

ExpertWeave: Efficiently Serving Expert-Specialized Fine-Tuned Adapters at Scale

Ge Shi^{1,*} Hanieh Sadri^{1,*} Qian Wang¹ Yu Zhang² Ying Xiong¹ Yong Zhang¹
Zhenan Fan¹

¹Huawei Technologies Canada ²Huawei Cloud

Abstract

Expert-Specialized Fine-Tuning (ESFT) adapts Mixture-of-Experts (MoE) large language models to enhance their task-specific performance by selectively tuning the top-activated experts for the task. Serving these fine-tuned models at scale is challenging: deploying merged models in isolation is prohibitively resource-hungry, while existing multi-adapter serving systems with LoRA-style additive updates are incompatible with ESFT’s expert-oriented paradigm. We present ExpertWeave, a system that serves multiple ESFT adapters concurrently over a single shared MoE base model, drastically reducing the memory footprint and improving resource utilization. To seamlessly integrate into existing inference pipelines for MoE models with non-intrusive modifications and minimal latency overhead, ExpertWeave introduces a virtual-memory-assisted expert weight manager that co-locates base-model and adapter experts without incurring memory overhead from fragmentation, and a fused kernel for batched rerouting to enable lightweight redirection of tokens to the appropriate experts at runtime. Our evaluations show that ExpertWeave can simultaneously serve multiple adapters of a 16B MoE model where the baseline runs out of memory, while providing up to $85\times$ more KV cache capacity and over 100% higher throughput with comparable resources, all without compromising model accuracy. ExpertWeave scales efficiently to tens of concurrent ESFT adapters with only marginal latency overhead.

1 Introduction

Large Language Models (LLMs) have recently attracted significant attention for their strong performance across a wide range of tasks (Radford et al., 2018; Bai et al., 2023; Meta, 2023; Team et al., 2023). These models are typically built on the Transformer architecture consisting of multi-head self-attention and feed-forward network (FFN) layers (Vaswani et al., 2017). In standard Transformers, each FFN layer uses one shared, dense parameter set for all tokens. This approach simplifies training and inference, but limits specialization and incurs substantial computational cost. Mixture-of-Experts (MoE) architectures (Jacobs et al., 1991; Jordan & Jacobs, 1994; Shazeer et al., 2017) address these challenges by replacing each FFN layer with multiple *experts* and using a learned router to activate only a few experts per token. This sparse computation increases model capacity while keeping per-token cost bounded (Fedus et al., 2022; Du et al., 2022; Zoph, 2022; Lepikhin et al., 2021).

As MoE models see wider adoption, efficiently adapting them to downstream tasks becomes increasingly important. Because full-parameter fine-tuning is often prohibitively expensive, Parameter-Efficient Fine-Tuning (PEFT) methods are preferred, among which Low-Rank Adaptation (LoRA) (Hu et al., 2022) is a prominent example. However, LoRA is designed primarily for dense models and is less suitable for MoE architectures (Wang et al., 2024). To address this gap, Wang et al. proposed Expert-Specialized Fine-Tuning (ESFT) (Wang et al., 2024). ESFT is motivated by an *expert specialization* pattern: routing for a specific task concentrates on a small expert subset, while different tasks activate different subsets.

*Equal contribution. Contact: {yong.zhang3,zhenan.fan1}@huawei.com

Accordingly, ESFT estimates per-layer expert relevance and fine-tunes only the most relevant experts. As a result, the number of fine-tuned experts varies across layers and adapters, reflecting task-specific relevance distributions. Although updating only a small fraction of experts (typically 5–15%), ESFT achieves performance comparable to full fine-tuning.

Despite its parameter efficiency, ESFT introduces new inference and serving challenges in modern Model-as-a-Service (MaaS) environments. A simple deployment strategy is to merge each adapter into a separate copy of the base model and serve the merged copies independently, but it scales poorly as the number of task-specific adapters grows, incurring substantial memory and compute overhead. Avoiding such duplication naturally motivates shared serving over a common base model, but this is fundamentally challenging for ESFT: unlike dense-model adapters with lightweight additive updates, ESFT yields heterogeneous, task-specific expert subsets that are structurally incompatible with existing MoE serving systems, leading to memory inefficiency and requiring additional runtime mechanisms to route tokens to the correct base-model or adapter-specific experts during batched inference. Accordingly, while systems like Punica (Chen et al., 2024) and S-LoRA (Sheng et al., 2024) enable concurrent serving of multiple LoRA adapters over a shared dense base model, efficient shared serving of ESFT adapters remains an open challenge.

To address this challenge, we present ExpertWeave, a system for shared inference and serving of ESFT adapters on MoE models, enabling multiple task-specific adapters to be served efficiently over a single shared base model. We make the following contributions:

- We present ExpertWeave, a system capable of serving multiple ESFT adapters concurrently over a shared base model, achieving scalable performance with minimal impact on throughput and latency.
- We design a unified and memory-efficient expert weight management mechanism that minimizes fragmentation via virtual memory, without requiring modifications to core computational kernels.
- We implement an efficient adapter-aware runtime with a batched rerouting operator that enables lightweight and accurate token dispatch across base-model and adapter experts.
- We validate ExpertWeave on Ascend NPUs (Liao et al., 2021) through end-to-end experiments, demonstrating scalability, efficiency, effective memory management, and serving accuracy across tasks.

2 Related Work

Multi-Adapter Serving for LoRA. The additive structure of LoRA has motivated multi-LoRA serving systems like Punica (Chen et al., 2024) and S-LoRA (Sheng et al., 2024) to allow multiple LoRA adapters to share a single base model by batching the base-model activation computations and then applying adapter-specific low-rank updates. Subsequent works explore scheduling (Li et al., 2025; Iliakopoulou et al., 2025; Wu et al., 2024), quantization (Xia et al., 2025), and caching (Zhang et al., 2025). CoLD combines multi-LoRA serving with contrastive decoding to enhance model performance in downstream task (Heisler et al., 2025). However, these methods fundamentally depend on additive low-rank updates. By contrast, ESFT fine-tunes subsets of experts within MoE layers, and different adapters may update different experts across layers. As a result, inference requires adapter-specific expert routing and execution instead of additive updates. This shift in execution pattern makes existing multi-LoRA batching and kernel optimizations inapplicable to ESFT-based multi-adapter serving.

Virtual Memory Management for LLMs. Beyond seminal work on PagedAttention (Kwon et al., 2023), recent work has also explored advanced virtual memory management (VMM) operations to improve memory efficiency in LLM training and inference (Guo et al., 2024; Prabhu et al., 2025; Xu et al., 2024; Yu et al., 2025). These methods use VMM mainly to reduce fragmentation in KV cache memory management in single- or multi-model serving scenarios. Our approach is orthogonal: it targets efficient model weight management for

multi-adapter serving and is potentially complementary to virtual-memory-assisted KV cache management techniques.

3 Problem Definition

Expert-Specialized Fine-Tuning. Consider a Transformer LLM with L MoE layers, where layer l contains M experts and a router that assigns each token to the top- k subset of experts. Expert-Specialized Fine-Tuning (ESFT) (Wang et al., 2024) is a parameter-efficient adaptation method that fine-tunes only a task-relevant expert subset in each MoE layer, while freezing all remaining experts and other model modules. Following the expert specialization pattern found in MoE models, ESFT identifies expert-task relevance by sampling task data and computing a relevance score for each expert, e.g., the average gate score or the token selection ratio. For each layer l , it then selects the smallest expert subset S_l whose cumulative relevance score exceeds a threshold p , and fine-tunes only the expert weights in S_l . Importantly, ESFT does not modify the routing mechanism itself: during training and inference, tokens may still be routed to any expert, but only the selected task-specific experts are fine-tuned.

Challenges of Serving ESFT Adapters. Consider a MaaS service provider serving multiple customers, each of whom may fine-tune a shared MoE base model for a different task, producing multiple ESFT adapters. To reduce inference cost, the provider would like to serve these adapters jointly over the shared base model, but doing so introduces system challenges beyond those in dense-model multi-adapter serving. In standard MoE inference pipelines, each layer routes tokens to their respective subsets of experts, which are then activated via grouped matrix multiplication (GMM) over a single contiguous expert weight tensor of shape $[M, H_{\text{out}}, H_{\text{in}}]$, where M is the number of experts and $H_{\text{in}}, H_{\text{out}}$ are the expert input and output dimensions. Existing serving systems are heavily optimized for this fixed and contiguous expert layout, but ESFT violates this assumption because each adapter fine-tunes a different subset of experts for each different layer. Consequently, the system must reconcile heterogeneous, dynamically loaded expert subsets with the static tensor abstraction required by optimized inference kernels.

A straightforward compatibility-preserving solution is to place both base-model and adapter experts in a unified tensor by reserving a fixed capacity of E_{max} experts for each adapter, yielding a padded tensor of shape $[M + N \cdot E_{\text{max}}, H_{\text{out}}, H_{\text{in}}]$ for up to N adapters. While this approach preserves the contiguous layout expected by existing kernels, most adapters only fine-tune a small fraction of experts, leading to substantial memory waste and fragmentation. Moreover, when requests for different adapters are batched together, the system must still route each token to the correct base-model or adapter-specific experts at runtime, and this routing must remain both accurate and lightweight, as it lies on the critical path of inference and must preserve the efficiency of existing MoE serving pipelines.

4 ExpertWeave: Scaling Multi-ESFT Serving

We design ExpertWeave to serve multiple ESFT adapters with these desired properties:

- Non-intrusive computation path modifications that enable the dynamic replacement of adapter experts while keeping the GMM operator unchanged;
- Memory efficiency that scales with the number of active adapter experts rather than fragmentation from worst-case padding;
- Negligible latency overhead relative to serving the base model.

The high-level architecture of ExpertWeave is as follows. Requests are dispatched with an associated ESFT adapter ID; a request may also target the base model using a special marker. Tokens from requests with up to N adapters can be packed together in the same batch. Layers not fine-tuned by ESFT are kept the same as before; batched tokens participate in computations in these layers with no modifications, enjoying advanced optimizations employed in LLM serving systems like continuous batching (Yu et al., 2022; Holmes et al.,

2024) and chunked prefill (Agrawal et al., 2024; Hu et al., 2024). In order to efficiently support ESFT adapters with minimal overhead and non-intrusive modifications, ExpertWeave relies on two core modules: virtual-memory-assisted expert weight management and a high-performance runtime through a batched rerouting operator.

ExpertWeave manages the weights of both base-model and adapter experts in a unified framework called the *virtual weight tensor*, a single three-dimensional tensor defined in a contiguous virtual address space. To accommodate the base model’s M experts as well as adapter-specific experts, the tensor is sized with padding to a fixed capacity that can hold the maximum number of experts ($E_{\max}$) for each of the N supported adapters. Concretely, it has shape $[M + N \cdot E_{\max}, H_{\text{out}}, H_{\text{in}}]$, where M is the number of base-model experts and $E_{\max}$ denotes the maximum number of experts allocated per adapter. This design decouples the logical tensor abstraction from its physical memory layout: although the tensor appears contiguous, its underlying memory is allocated independently and on demand. It provides a simple, unified view of all expert weights to the GMM operator while avoiding the memory fragmentation issues discussed in Section 3.

To manage expert computation at runtime, for each MoE layer l , ExpertWeave utilizes a *batched rerouting* operator with an *ESFT expert map* $\Pi^{(l)}$, a two-dimensional array of shape $[N, M]$. An entry $\Pi^{(l)}[i, j]$ stores the index of expert j of adapter i in the virtual weight tensor: If expert j is not fine-tuned by adapter i , $\Pi^{(l)}[i, j]$ simply holds the original index of j ; if it is fine-tuned, $\Pi^{(l)}[i, j]$ points to the specific location where the adapter expert weights are loaded in the virtual weight tensor:

$$\Pi^{(l)}[i, j] = \begin{cases} j, & \text{if } j \text{ is not fine-tuned in adapter } i, \\ \Delta_i + \delta_{ij}^{(l)}, & \text{if } j \text{ is fine-tuned in adapter } i \end{cases}$$

where $\Delta_i = M + i \cdot E_{\max}$ is the offset of adapter i in the first dimension of the virtual weight tensor and $\delta_{ij}^{(l)}$ is the offset of the fine-tuned expert j within the range $[\Delta_i : \Delta_i + E_{\max}]$ assigned to adapter i where $0 \leq \delta_{ij}^{(l)} < e_i^{(l)}$.

At inference time, batched rerouting directs tokens to their appropriate experts by leveraging the ESFT expert map Π to update the top- k expert IDs selected by the MoE router. For a token x from adapter $A(x)$, the router first selects the original set of top- k experts $\text{TopK}(x)$; batched rerouting then updates this set by replacing base-model expert indexes with their corresponding (fine-tuned or not) counterparts from the ESFT expert map Π :

$$\text{TopK}'(x) := \{\Pi[A(x), j] : j \in \text{TopK}(x)\}$$

This updated set of top- k experts, $\text{TopK}'(x)$, is then passed to the unmodified inference path with token dispatching and the GMM operator, which is executed on the virtual weight tensor without needing to be aware of the adapter-specific logic.

4.1 Virtual-Memory-Assisted Expert Weight Management

As described in Section 1, loading adapter expert weights following the padding approach can lead to severe memory fragmentation on device. To address this issue, ExpertWeave adopts a strategy that *decouples virtual memory reservation from physical memory allocation*.

Specifically, ExpertWeave reserves a contiguous virtual address space for the full weight tensor, but only allocates physical pages for the experts that are actually present in the adapters, with no allocation for the additional regions introduced by padding. When loading adapter i with $e_i^{(l)}$ experts in layer l , physical memory is mapped only for the range $[\Delta_i : \Delta_i + e_i^{(l)}]$. The leftover padding range $[\Delta_i + e_i^{(l)} : \Delta_i + E_{\max}]$ is intentionally kept not mapped, ensuring that no physical memory is wasted on unused areas of the tensor.

Memory Pool and Mapping. ExpertWeave implements the virtual weight tensor through two components:

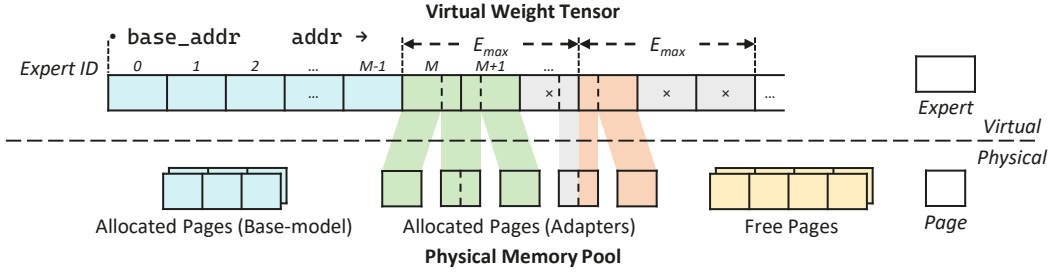


Figure 1: An illustration of a virtual weight tensor unfolded alongside its first dimension and a physical memory pool.

- A *physical memory pool* per device that manages physical memory pages with fixed sizes (e.g., 2 MB granularity). The physical memory pool pre-allocates pages from the device runtime and supplies them to the virtual weight tensor at adapter loading time; evicted adapters release their pages back to the pool, which are reused for subsequent adapters or eventually reclaimed by the device runtime.
- An *expert memory manager* per virtual weight tensor that handles physical pages allocated to the tensor by mapping them to the desired regions used by active experts, or the corresponding unmapping when adapters are evicted.

When a virtual weight tensor is instantiated, ExpertWeave reserves the contiguous virtual address space and returns a pointer to its base address (`base_addr`) without mapping any physical memory page. Upon the loading of a range of consecutive experts, either from the base model at system initialization time or from an adapter at runtime, the expert memory manager computes the starting virtual address for the range:

$$\text{start_addr} = \text{base_addr} + \Delta_i \times \text{expert_size}$$

Then, it interacts with the pool by requesting the required number of physical pages and mapping them to the appropriate offsets in the virtual address space starting at `start_addr`, before copying the actual expert weights into the mapped region. Unloading a range of experts works similarly in a backward manner by unmapping the physical pages and releasing them back to the physical memory pool for subsequent reuse.

Figure 1 illustrates a virtual weight tensor in an MoE layer l and its relationship with the physical memory pool, where the system-level $E_{\max} = 3$. The M experts in the base model and their associated physical pages (blue) are created at system initialization time (with the mapping omitted). There are $e_0^{(l)} = 2$ experts in the first (green) adapter and $e_1^{(l)} = 1$ expert in the second (orange) adapter. Each expert consumes 1.5 pages, and there are 3 pages mapped to the region of the 2 experts in the first adapter. Regions marked with "x" (gray) represent padding within the virtual weight tensor, without physical pages assigned to these regions. Free pages in the memory pool remain available for future allocations, enabling efficient reuse.

Expert-Page Alignment. A practical challenge of expert weight management is that the memory size of a model-defined expert may not be an exact multiple of the fixed granularity of physical memory pages. This misalignment can lead to conflicts where expert boundaries in virtual address space may not coincide with physical page boundaries, leading to internal fragmentation in partially filled pages. Figure 1 shows an example where the `start_addr` of the second (orange) adapter is not aligned with the page size. Inconsiderate implementations would therefore lead to runtime errors or wasted memory.

To prevent such conflicts and maximize memory utilization, ExpertWeave adopts a sub-page allocation strategy, where the unused segment of a mapped but partially filled page is intentionally made available for use by a subsequently loaded neighboring adapter. We implement this strategy through rigorous tracking of the expert-page relationship and reference counting.

4.2 Batched Rerouting

To efficiently leverage the experts in the virtual weight tensor, ExpertWeave introduces a batched rerouting operator in the forward path in each MoE layer. After the top- k expert IDs are computed by the MoE router in each layer, ExpertWeave additionally reroutes some selected experts to their fine-tuned counterparts if they are for tokens in requests to different adapters, utilizing the aforementioned ESFT expert map. This operator is performed at token granularity (Chen et al., 2024), ensuring seamless compatibility with token-level scheduling algorithms like continuous batching and chunked prefill.

Figure 2 shows an example of batched rerouting with a base model of $M = 64$ and $K = 6$, the maximum number of adapters $N = 2$, and $E_{\max} = 8$. In layer l , there are $e_0^{(l)} = 3$ experts in the first adapter and $e_1^{(l)} = 7$ experts in the second adapter. The hidden states of the scheduled tokens are paired with an *adapter ID* (AID) array indicating the associated adapter ID of each token; a special marker value of -1 represents a token from base-model requests (Kwon et al., 2023). Each entry in the top- k ID array is conditionally updated by looking up the ESFT expert map $\Pi^{(l)}$ using both its associated AID and its value, producing an updated top- k ID array corresponding to experts in the virtual weight tensor.

The batched rerouting operator is implemented as a series of vector-based operations, including broadcasting the AID array, computing offsets inside the ESFT expert map, and a gather operation. To reduce kernel launching overhead and redundant data copying, we implemented a fused kernel leveraging multiple vector cores on Ascend NPUs (Liao et al., 2021).

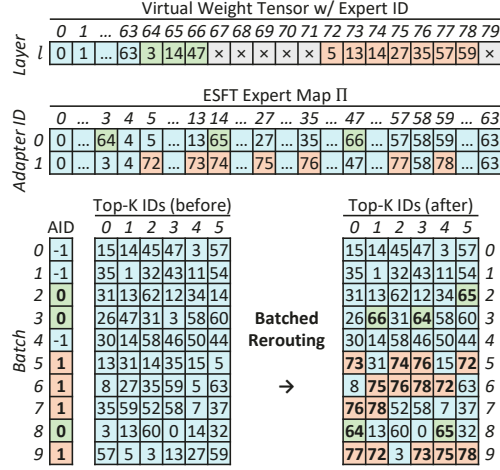


Figure 2: An example of batched rerouting.

5 Experiments

ExpertWeave is built on top of vLLM (Kwon et al., 2023), a state-of-the-art LLM inference and serving engine, and vLLM-Ascend (vLLM Community, 2024), a community-maintained plugin for vLLM on the Ascend (Liao et al., 2021) platform.

We evaluated the performance of ExpertWeave in online serving settings with real-time dynamic workload patterns, and then analyzed the memory efficiency of ExpertWeave.

5.1 Experimental Setup

Hardware. We evaluated ExpertWeave on a quad-socket server with 192 ARM cores, 2TB of main memory, and 16 Ascend NPUs, where each NPU has 64GB of memory.

Serving Framework and Setup. ExpertWeave is implemented on top of vLLM (Kwon et al., 2023) and vLLM-Ascend (vLLM Community, 2024) version v0.13.0rc1. We use Tensor Parallelism (TP) for self-attention layers and Expert Parallelism (EP) for MoE layers when applicable.

Models and Adapters. Unlike dense models that activate the full model for each token, MoE models exhibit different expert activation patterns for different tokens. Using synthetic adapters with dummy weights would incorrectly capture system behavior, especially under a multi-adapter setup. We therefore match real ESFT adapters with queries from their respective domains to accurately measure the performance of ExpertWeave. We use the 10

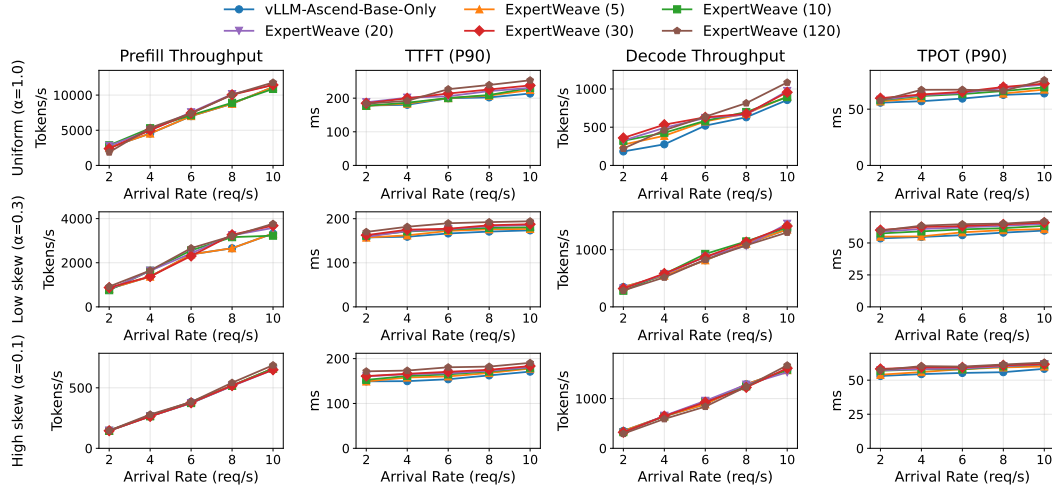


Figure 3: End-to-end performance for serving multiple adapters under uniform ($\alpha = 1$) and skewed ($\alpha = 0.3$ and 0.1) workloads on 16 Ascend NPUs; ExpertWeave scales with increasing numbers of ESFT adapters with minor overhead compared with vLLM-Ascend.

adapters (DeepSeek-AI, 2024a) illustrated in Table 3 in Appendix 3; they are replicated for experiments beyond 10 adapters. We use the ESFT vanilla model (DeepSeek-AI, 2024b) as the base model, a 16B model sharing the same architecture as DeepSeek-V2-Lite (DeepSeek-AI, 2024).

Baselines. To ensure fairness, we compared ExpertWeave with vLLM-Ascend on the same hardware setup. We used two baselines for vLLM-Ascend:

- *vLLM-Ascend-Merged*: As there is no native support for ESFT adapters in vLLM-Ascend, we first collected merged ESFT models offline by replacing the experts in the base model with the adapter’s fine-tuned experts, then served the merged model with vLLM-Ascend; for multiple ESFT adapters, we ran multiple vLLM-Ascend instances on different NPUs and dispatched requests of each domain to its instance.
- *vLLM-Ascend-Base-Only*: When deploying multiple full merged models was infeasible due to hardware resource constraints, we instead ran a single vLLM-Ascend instance serving only the ESFT vanilla base model and sent all requests to this instance.

Metrics. We reported prefill throughput, time-to-first-token (TTFT), decode throughput, and time-per-output-token (TPOT) as key performance metrics. For TTFT and TPOT, we report the 90th percentile (P90) tail latency.

5.2 End-to-End Performance

Workloads. For online inference, we construct prompts by sampling from real-world test sets spanning five tasks: math, intent recognition, summarization, law, and translation (Wang et al., 2024). These datasets are used to generate a synthetic workload that mimics realistic multi-domain traffic. Prompts from each dataset are routed exclusively to their corresponding fine-tuned adapters to preserve specialization. Prompt lengths vary across datasets from 190 to 2048 tokens.

To evaluate ExpertWeave under workload skew, we assign per-adapter request shares using a power-law distribution with shape parameter α (Sheng et al., 2024): smaller α induces heavier skew, while larger α approaches a uniform distribution ($\alpha = 1$). Each adapter i is assigned an arrival rate λ_i such that the total rate is $\lambda = \sum_{i=1}^N \lambda_i$. We then generate requests using independent Poisson processes with rate λ_i per adapter and execute all traces

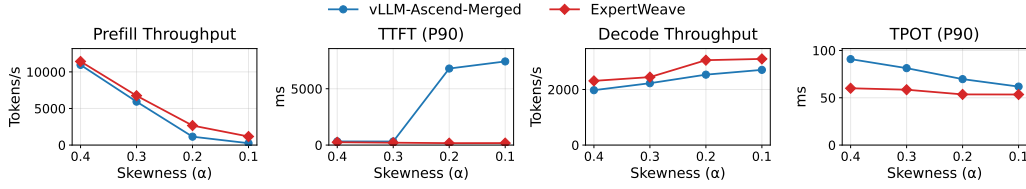


Figure 4: End-to-end performance for serving adapters vs. merged models under skewed workloads. ExpertWeave consistently outperforms vLLM-Ascend-Merged across skew levels.

concurrently over a 100-second horizon. We limit the maximum schedulable requests to 256 (through `--max-num-seqs` in vLLM) and vary the total request rate λ to control system load.

Serving Multiple ESFT Adapters. To evaluate the scalability of ExpertWeave with respect to the number of ESFT adapters N , we conduct experiments with $N = 5, 10, 20, 30$, and 120, compared against the vLLM-Ascend-Base-Only baseline. All experiments are performed on a single node with 16 Ascend NPUs, with TP=16 for attention and EP=16 for MoE, under an aggregate request rate $\lambda \in [2, 10]$ req/s.

As shown in Figure 3, under a uniform workload ($\alpha = 1.0$) with varying request arrival rates, serving ESFT adapters with ExpertWeave incurs low latency overhead as the number of adapters increases: approximately 5–8% for 5–10 adapters, rising to around 11–12% for 20–30 adapters, and reaching at most 18% when scaling to 120 adapters. TPOT exhibits a similar trend with approximately 4–14% overhead for 5–30 adapters and within 18% for 120 adapters, demonstrating stable performance when serving hundreds of adapters. This overhead mainly arises from (1) batched rerouting and (2) more diverse expert activation in the GMM operator, yet remains within typical SLO bounds.

Prefill throughput remains broadly comparable to the baseline for up to 30 adapters, with moderate variation across different request arrival rates. When scaling to 120 adapters, the variation becomes more pronounced, but no consistent degradation is observed. Decode throughput is generally higher than the baseline across all adapter configurations, although the improvement does not vary monotonically with the number of adapters. This behavior may be influenced by differences in output-length distributions across domains and therefore should not be interpreted solely as a direct performance benefit of ExpertWeave.

We observe the same qualitative behavior under skewed workloads ($\alpha = 0.3$ and 0.1), with bounded latency overhead and no systematic throughput degradation as N increases. The spread across different adapter configurations also appears generally smaller than under the uniform workload. Overall, ExpertWeave scales effectively to 120 adapters on a single node and can be further extended to larger-scale deployments through horizontal scaling with additional instances.

Comparison with Serving Merged Models. We compare ExpertWeave with merged-model serving under the same workload and resource constraints. We fix both the number of tasks (adapters) and the total memory budget for both systems. We deploy ExpertWeave on eight NPUs with TP=8 and EP=8. For vLLM-Ascend-Merged, we run eight independent instances (one per merged model), each on two NPUs with TP=2 and EP=2, resulting in a total of 16 NPUs. We allocate 90% of memory to ExpertWeave and 45% per vLLM-Ascend instance, ensuring similar aggregate memory usage. We evaluate 8 adapters at a fixed request rate $\lambda = 20$, and vary the workload skew using the parameter α , where smaller values of α indicate increasingly skewed request distributions.

As shown in Figure 4, ExpertWeave consistently outperforms vLLM-Ascend-Merged across all skew levels in both throughput and latency. For throughput, ExpertWeave achieves modest gains under mild skew (e.g., 4–14% prefill and 10–17% decode at $\alpha = 0.4$ – 0.3), and significantly larger improvements as skew increases, exceeding 100% improvement in prefill throughput under highly skewed workloads. Decode throughput also improves consistently, with gains of up to approximately 20%. ExpertWeave reduces TTFT by 22–34%

#Adapters	ExpertWeave		EW-Padding		vA-Merged		Token Gain (%)	
	Mem. (GB)	Tokens	Mem. (GB)	Tokens	Mem. (GB)	Tokens	EW vs. EW-Padding	EW vs. vA-Merged
2	36.6	1.327M	40.1	1.268M	58.6	944K	4.6	40.6
4	42.9	1.213M	50.8	1.080M	117.2	14K	12.2	8511.8
6	50.8	1.125M	61.6	893K	OOM	OOM	26.0	-
8	56.6	977K	72.3	706K	OOM	OOM	38.4	-
10	62.9	875K	83.1	516K	OOM	OOM	69.5	-
12	69.9	747K	93.9	328K	OOM	OOM	127.5	-
14	75.0	660K	104.6	141K	OOM	OOM	368.5	-
16	80.1	572K	OOM	OOM	OOM	OOM	-	-

Table 1: Memory usage and KV cache capacity across different numbers of adapters. ExpertWeave consistently requires less memory and supports more KV cache tokens.

under mild skew and by over 97% under high skew ($\alpha = 0.2$ and 0.1). TPOT is also reduced consistently, with improvements ranging from 13% to 34% across all settings. These gains become more pronounced as workload skew increases. In vLLM-Ascend-Merged, each merged model is served in isolation, causing the devices assigned to frequently requested adapters to become saturated while others remain underutilized. This imbalance leads to significant queueing delays, particularly impacting TTFT. In contrast, ExpertWeave utilizes all available devices and distributes computation across them via TP and EP, enabling resources to be shared across both adapters and the base model. As a result, ExpertWeave avoids idle devices, reduces stragglers, and sustains higher throughput and lower latency under skewed workloads.

5.3 Memory Efficiency

We evaluate the memory efficiency of ExpertWeave by comparing it against (1) vLLM-Ascend-Merged and (2) *ExpertWeave-Padding*, an implementation of ExpertWeave without virtual-memory-assisted expert weight management. Experiments were conducted using two NPUs with TP=2 and EP=2. We used the same set of adapters described in Section 5.2 and served up to 16 adapters.

As shown in Table 1, vLLM-Ascend-Merged scales poorly with the number of adapters and becomes infeasible beyond 4 adapters due to out-of-memory (OOM) errors, as each adapter requires a separate copy of the base model. Even at low adapter counts, it significantly reduces KV cache capacity. In contrast, ExpertWeave serves multiple adapters on top of a shared base model with no duplication, allowing adapters to be loaded with minimal memory overhead. As a result, ExpertWeave consistently uses less memory and maintains high KV cache capacity, with up to $85\times$ capacity compared to vLLM-Ascend-Merged.

Compared with ExpertWeave-Padding, ExpertWeave improves KV cache capacity by 4.6%–368.5%, with higher gains at larger adapter counts. ExpertWeave-Padding allocates memory based on the maximum number of experts across all adapters and layers, leading to substantial over-provisioning; in contrast, ExpertWeave materializes only the necessary weights, resulting in significantly better memory utilization. ExpertWeave remains feasible up to 16 adapters and supports up to 572K tokens, while both ExpertWeave-Padding and vLLM-Ascend-Merged encounter OOM at larger scales. These memory savings directly translate into higher throughput and larger KV cache capacity in online serving scenarios, where memory is often the primary bottleneck.

6 Conclusion

We presented ExpertWeave, a system for efficiently serving multiple Expert-Specialized Fine-Tuning (ESFT) adapters over a shared Mixture-of-Experts (MoE) base model. To improve memory utilization and reduce fragmentation, ExpertWeave introduces virtual-memory-assisted expert weight management to handle the efficient placement of base-model and

adapter experts in memory. Compared with baseline methods that merge adapters into the base model, ExpertWeave substantially reduces memory consumption, achieving up to $85\times$ larger KV cache capacity and continuing to scale at higher adapter counts when the baseline runs out of memory. To minimize runtime overhead, ExpertWeave incorporates a fused kernel for batched rerouting to enable lightweight adapter-aware token dispatch. Experiments show that ExpertWeave delivers over $2\times$ throughput compared to serving merged models with vLLM-Ascend under comparable resources, and ExpertWeave scales efficiently with bounded latency overhead: approximately 4–14% for up to 30 ESFT adapters and at most 18% when scaling to 120 adapters. Although evaluated through ESFT, the underlying systems abstraction supports batched serving of sparse expert adaptations that preserve most of a shared MoE base model and can complement additive adaptation mechanisms like multi-LoRA, which we leave for future work. By enabling efficient base-model sharing across ESFT adapters, ExpertWeave makes large-scale multi-tenant deployment of specialized MoE models practical and cost-effective.

References

- Advanced Micro Devices, Inc. HIP runtime API: Virtual memory management. https://rocm.docs.amd.com/projects/HIP/en/latest/reference/hip_runtime_api/modules/memory_management/virtual_memory_reference.html, 2026. Accessed 2026-08-04.
- Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, 2024.
- Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.
- Lequn Chen, Zihao Ye, Yongji Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant lora serving. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 1–13, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/054de805fcceb78a201f5e9d53c85908-Paper-Conference.pdf.
- DeepSeek-AI. Esft: Expert-specialized fine-tuning collection. <https://huggingface.co/collections/deepseek-ai/esft>, 2024a. Hugging Face collection, accessed 2026.
- DeepSeek-AI. Esft-vanilla-lite. <https://huggingface.co/deepseek-ai/ESFT-vanilla-lite>, 2024b. Hugging Face model repository, accessed 2026.
- DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.
- Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.
- William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.
- Cong Guo, Rui Zhang, Jiale Xu, Jingwen Leng, Zihan Liu, Ziyu Huang, Minyi Guo, Hao Wu, Shouren Zhao, Junping Zhao, et al. Gmlake: Efficient and transparent gpu memory defragmentation for large-scale dnn training with virtual memory stitching. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pp. 450–466, 2024.

- Morgan Lindsay Heisler, Linzi Xing, Ge Shi, Hanieh Sadri, Gursimran Singh, Weiwei Zhang, Tao Ye, Ying Xiong, Yong Zhang, and Zhenan Fan. Enhancing learned knowledge in lora adapters through efficient contrastive decoding on ascend npus. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 4499–4510, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3737215. URL <https://doi.org/10.1145/3711896.3737215>.
- Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.
- Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, et al. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024.
- Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- Nikoleta Iliakopoulou, Jovan Stojkovic, Chloe Alverti, Tianyin Xu, Hubertus Franke, and Josep Torrellas. Chameleon: Adaptive caching and scheduling for many-adapter llm inference environments. In *MICRO 2025: 58th IEEE/ACM International Symposium on Microarchitecture*, pp. 217–231. IEEE Computer Society, 2025. doi: 10.1145/3725843.3756083. URL <https://doi.org/10.1145/3725843.3756083>.
- Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.
- Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23*, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL <https://doi.org/10.1145/3600006.3613165>.
- Dmitry Lepikhin, HyounJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021. URL <https://openreview.net/forum?id=qrwe7XHTmYb>.
- Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. Toppings: Cpu-assisted, rank-aware adapter serving for llm inference. In *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, pp. 613–629. USENIX Association, 2025. URL <https://www.usenix.org/conference/atc25/presentation/li-suyi-toppings>.
- Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 789–801, 2021. doi: 10.1109/HPCA51647.2021.00071.
- AI Meta. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- NVIDIA. CUDA Driver API: Virtual memory management. https://docs.nvidia.com/cuda/cuda-driver-api/group_CUDA_VA.html, 2026. Version 13.3.1; accessed 2026-08-04.

- OpenAI. GSM8K: Grade School Math 8K Dataset. <https://huggingface.co/datasets/openai/gsm8k>, 2021. Accessed: 2025-06-02.
- Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 1133–1150, 2025.
- Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarczyk, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. URL <https://openreview.net/forum?id=B1ckMDqlg>.
- Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. Slora: Scalable serving of thousands of lora adapters. In P. Gibbons, G. Pekhimenko, and C. De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 296–311, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/file/906419cd502575b617cc489a1a696a67-Paper-Conference.pdf.
- Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL 2019*, pp. 10–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367196. doi: 10.1145/3315508.3329973. URL <https://doi.org/10.1145/3315508.3329973>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- vLLM Community. vllm ascend plugin. <https://github.com/vllm-project/vllm-ascend>, 2024. Accessed: 2025-06-05.
- Zihan Wang, Deli Chen, Damai Dai, Runxin Xu, Zhuoshu Li, and Yu Wu. Let the expert stick to his last: Expert-specialized fine-tuning for sparse architectural large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 784–801, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.46. URL <https://aclanthology.org/2024.emnlp-main.46/>.
- Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. dlora: dynamically orchestrating requests and adapters for lora llm serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation, OSDI’24, USA*, 2024. USENIX Association. ISBN 978-1-939133-40-3.
- Yifei Xia, Fangcheng Fu, Wentao Zhang, Jiawei Jiang, and Bin Cui. Efficient multi-task llm quantization and serving for multiple lora adapters. In *Proceedings of the 38th International Conference on Neural Information Processing Systems, NIPS ’24*, Red Hook, NY, USA, 2025. Curran Associates Inc. ISBN 9798331314385.
- Jiale Xu, Rui Zhang, Cong Guo, Weiming Hu, Zihan Liu, Feiyang Wu, Yu Feng, Shixuan Sun, Changxu Shao, Yuhong Guo, et al. vtensor: Flexible virtual tensor management for efficient llm serving. *arXiv preprint arXiv:2407.15309*, 2024.

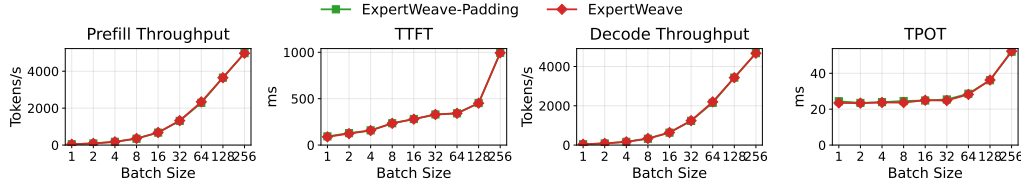


Figure 5: Effect of virtual weight tensors on TTFT and TPOT. Virtual weight tensors incur negligible overhead on inference latency.

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

Shan Yu, Jiarong Xing, Yifan Qiao, Mingyuan Ma, Yangmin Li, Yang Wang, Shuo Yang, Zhiqiang Xie, Shiyi Cao, Ke Bao, et al. Prism: Unleashing gpu sharing for cost-efficient multi-llm serving. *arXiv preprint arXiv:2505.04021*, 2025.

Hang Zhang, Jiuchen Shi, Yixiao Wang, Quan Chen, Yizhou Shan, and Minyi Guo. Improving the serving performance of multi-lora large language models via efficient lora and kv cache management. *arXiv preprint arXiv:2505.03756*, 2025.

Barret Zoph. Designing effective sparse expert models. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1044–1044. IEEE, 2022.

A Appendix

A.1 Extended Experimental Results

Impact of Virtual Weight Tensor. We evaluate the effect of virtual weight tensors (ExpertWeave) against the padding baseline (ExpertWeave-Padding). In this experiment, we select the gate-math adapter, fix both the prompt length and output response length to 1,024 tokens, and vary the batch size. We measure TTFT, TPOT, as well as prefill and decode throughput.

As shown in Figure 5, ExpertWeave and ExpertWeave-Padding achieve comparable performance, with differences of less than 2% in TTFT and less than 1% in TPOT across prompt lengths and batch sizes. These results indicate that virtual weight tensors do not introduce inference overhead. Furthermore, despite maintaining similar latency and throughput, virtual weight tensors provide significant memory savings, as discussed in Section 5.3. This improvement is particularly important in multi-adapter serving scenarios, where reduced memory pressure minimizes queuing delays caused by memory limitations, thereby increasing overall system throughput.

Impact on Accuracy. ExpertWeave only manipulates expert indexes rather than model weights, so its token-level activations have minimal accuracy differences compared with serving merged models. We validate this claim by evaluating the system on two downstream tasks: GSM8K math (OpenAI, 2021) and intent recognition (Wang et al., 2024). We configure ExpertWeave to load four adapters, gate-math, token-math, gate-intent and token-intent, and evaluate accuracy on both tasks. For vLLM-Ascend, we use the corresponding merged models.

Table 2 shows ExpertWeave is able to serve multiple adapters while matching per-task accuracy of the respective merged model for both tasks. This result validates that ExpertWeave is a robust system for multi-adapter serving without any accuracy loss.

Adapter Loading and Rerouting Overheads. We measure the adapter loading latency using the 10 adapters listed in Table 3; across these adapters, the average loading latency is

	GSM8K	Intent
Base Model	56.5	18.6
vLLM-Ascend + gate-math	62.3	-
vLLM-Ascend + token-math	64.7	-
vLLM-Ascend + gate-intent	-	78.8
vLLM-Ascend + token-intent	-	76.6
ExpertWeave + gate adapters	62.3	78.8
ExpertWeave + token adapters	64.8	76.6

Table 2: Accuracy of ExpertWeave with four adapters and their respective merged models. ExpertWeave preserves per-task accuracy on multiple downstream tasks. The numbers are not directly comparable to those reported in (Wang et al., 2024) due to differences in prompts, serving engines, hardware, etc.

1.86 ± 0.53 seconds. This cost is incurred only when an adapter is loaded in response to its first request, and we consider the resulting cold-start overhead acceptable for on-demand adapter serving. We further profile the batched rerouting kernel and find that it accounts for less than 1% of the total execution time in most cases. These results indicate that the latency difference between the baseline and ExpertWeave is primarily due to dynamic expert activation patterns at runtime rather than rerouting itself.

A.2 Memory Fragmentation Analysis

We analyze real ESFT adapters to understand the memory fragmentation of the padding approach. We selected 10 fine-tuned ESFT adapters that cover 5 different tasks as detailed in Table 3 from the original ESFT paper (Wang et al., 2024).

To quantify the deviation between the number of experts in each layer and the maximum number of experts in any layer of an adapter, we define the *adapter sparsity factor* S_i for adapter i as:

$$S_i = \frac{\sum_{l=1}^L (E_i - e_i^{(l)})}{L \cdot E_i},$$

where L is the total number of layers, $e_i^{(l)}$ is the number of experts fine-tuned in layer l , and $E_i = \max_{l=1}^L e_i^{(l)}$ denotes the maximum number of experts across all layers of the adapter. A value of $S_i = 0$ indicates a fully dense adapter with less inherent fragmentation, while larger values imply heavier intra-adapter padding. Table 3 reports the sparsity factor for the 10 adapters: while a small number of adapters are relatively dense, most exhibit moderate to high sparsity.

Even with dense adapters, memory fragmentation can still occur as a result of inter-adapter padding, as the tensors must be padded to the largest expert count observed across all adapters. To assess the memory efficiency of serving multiple adapters at the same time, we define the *memory fragmentation factor* as:

$$F_{\text{mem}} = \frac{L \cdot (M + N \cdot E_{\text{max}})}{\sum_{l=1}^L (M + \sum_{i=1}^N e_i^{(l)})},$$

The memory fragmentation factor captures the ratio between allocated memory and actual memory usage for adapter expert weights, with $F_{\text{mem}} = 1.0$ indicating no fragmentation and $F_{\text{mem}} > 1.0$ indicating proportional overhead. For the 10 ESFT adapters in Table 3, the smallest feasible $E_{\text{max}} = 13$ yields an associated memory fragmentation factor of $F_{\text{mem}} = 1.51$, indicating a 51% memory overhead beyond the necessary adapter weights due to padding alone. As the memory on device is already a scarce resource, limiting this fragmentation overhead to an acceptable degree becomes a practical challenge.

Domain	Adapter	Max. #Experts	Avg. #Experts	Sparsity
Math	gate-math	12	7.04	0.41
	token-math	9	6.12	0.32
Intent	gate-intent	12	9.50	0.21
	token-intent	8	7.12	0.11
Summary	gate-summary	11	7.73	0.30
	token-summary	8	5.15	0.36
Law	gate-law	12	7.35	0.39
	token-law	10	6.58	0.34
Translation	gate-translation	13	4.69	0.64
	token-translation	6	3.85	0.36

Table 3: Expert configuration and sparsity of 10 selected ESFT adapters across 5 domains. *Max. #Experts* is the maximum experts in any layer, *Avg. #Experts* is the average experts across all layers, and *Sparsity* is the adapter sparsity factor S_i .

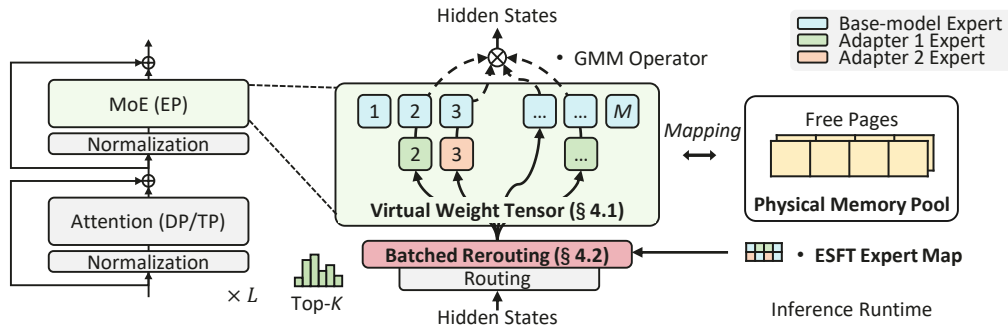


Figure 6: High-level architecture of ExpertWeave. Expert weights are organized in a *virtual weight tensor* with decoupled physical page allocation managed by a *physical memory pool*. An *ESFT expert map* records the locations of fine-tuned experts within this tensor, enabling ExpertWeave to perform *batched rerouting* of tokens by replacing base-model expert IDs with their adapter-specific counterparts during inference time.

A.3 Implementation

We describe how multi-adapter ESFT serving is implemented in our ExpertWeave inference stack, focusing on the concrete system components and their interaction during adapter loading and execution. Figure 6 shows the high-level architecture of ExpertWeave. The ExpertWeave implementation is designed to fit into an optimized serving engine like vLLM and vLLM-Ascend (Kwon et al., 2023; vLLM Community, 2024) with minimal disruption to the original forward path. In particular, we preserve the existing routing, dispatch, and expert activation computation pipeline, and introduce adapter support through additional weight management logic and a lightweight runtime rerouting step.

We implement our prototype on the Ascend NPU platform, and we distinguish the mechanisms of ExpertWeave from the platform-specific implementation details. Section 4 defines the general serving mechanisms of ExpertWeave including request batching, adapter management, virtual weight tensors, sub-page allocation, and batched rerouting. In our prototype, virtual-memory operations are implemented using Ascend Computing Language (AscendCL) APIs, while batched rerouting is implemented as a fused Ascend C kernel. They are specific to the Ascend NPU backend rather than requirements of the ExpertWeave system design. We describe the concrete implementation below and discuss its compatibility with other hardware backends at the end of this section.

Inference Runtime and Expert Weight Management. Each request is associated with an adapter ID, and tokens from different adapters may be packed into the same batch. ExpertWeave manages adapters using a three-tier storage hierarchy. Hot adapters reside in device memory to support efficient GMM execution, recent adapters that are not immediately required may be offloaded to CPU memory and retained as warm adapters, and inactive adapters remain in persistent storage. When the device-resident adapter set exceeds the configured capacity, ExpertWeave applies a least-recently-used (LRU) replacement policy to select adapters for offloading or eviction. Before executing a batch, the worker first determines the set of adapters required by the active requests. Adapters that are already device-resident remain active, while missing adapters are loaded on demand from persistent storage or host memory and then transferred to accelerator memory for device-side execution. Adapter management is therefore handled outside the model forward pass, and the model only sees the adapters active in the current batch.

To handle adapter experts within the existing MoE kernels, each supported MoE layer is extended with adapter expert storage. Concretely, the layer is initialized with an expanded expert tensor that can hold both the base experts and the reserved expert slots for all adapter regions. Loading of an adapter then proceeds by selecting a slot for the adapter and copying its fine-tuned expert weights into the corresponding region of this expanded tensor, which is managed using the virtual weight tensor design described in Section 4.1.

Before writing a new adapter into a slot, the implementation resets the corresponding region. This reset clears any stale expert weights left by a previously active adapter and restores the layer state to its default configuration. The new adapter is then loaded by assigning consecutive expert slots within its reserved region to the fine-tuned experts present in that layer and copying their weights into those locations. Only the experts that appear in the adapter are materialized.

To avoid invasive changes to the original MoE implementation, each target MoE layer is wrapped by an ESFT-layer that manages this additional adapter state while delegating all actual computation to the base MoE layer. The wrapper maintains the ESFT expert map, a per-layer lookup table that records, for each active adapter and each base-model expert ID, which expert ID in the virtual weight tensor should be used during execution. Initially, this table is an identity mapping, so all routed experts refer to the original base-model experts. When an adapter is activated, only the entries corresponding to its fine-tuned experts are changed to point to the newly loaded expert slots; all remaining entries continue to point to the base-model experts. As a result, unchanged experts are automatically shared across all adapters, while modified (fine-tuned) experts are selectively rerouted.

Batched Rerouting. During the forward pass, the router still computes top- k expert selections exactly as in the original model. At this point, the selected expert IDs are still expressed in the base-model expert space. The implementation then applies the lightweight batched rerouting step that rewrites these expert IDs according to the adapter associated with each token. Operationally, the runtime reads the adapter ID of each token, uses it to select the appropriate row of the layer-local lookup table, and replaces each routed expert ID with the corresponding execution-time expert ID. If the routed expert was not fine-tuned by the adapter, the lookup returns the original base-model expert. If it was fine-tuned, the lookup returns the ID of the slot where the adapter expert was loaded. This indirection is implemented as a batched device-side kernel and is performed after routing but before dispatch to the expert computation kernel. The resulting top- k expert indices are then passed to the standard MoE dispatch and computation pipeline, which executes them without any modification to the underlying kernels or communication logic.

The batched rerouting kernel overwrites the top- k expert IDs from the router for all tokens in a batch using the adapter-specific ESFT expert map. It takes as input the top- k expert ID array, a per-token adapter ID array, and the ESFT expert map stored as a 2D lookup table. The output is an updated top- k array of the same shape, where each expert ID is replaced by its mapped counterpart when the token activates an adapter-specific expert, or unchanged otherwise.

The kernel processes tokens in tiles and first stages the adapter IDs and top- k expert IDs into on-chip memory. For each token, its adapter ID is expanded to match the layout of the padded top- k entries, and a mask is generated to identify tokens from requests to the base model using a special marker value. The kernel then computes flattened lookup offsets of the form $\text{adapter ID} \times M + \text{expert ID}$ for each routed expert ID, where M is the number of base experts. These offsets are used to gather the mapped expert IDs from the expert map. Finally, the mask is applied to restore the original top- k expert IDs for base-model tokens, ensuring that rerouting only affects adapter-associated tokens.

All steps, including adapter ID expansion, masking, offset computation, gather, and conditional selection, are fused into a single tiled kernel. This avoids intermediate memory traffic and eliminates the need for multiple kernel launches, allowing rerouting to be efficiently integrated into the MoE forward path.

Expert-Parallel Execution. Under expert parallelism (EP) for distributed inference, ExpertWeave reuses the existing global-to-local expert mapping of the underlying MoE runtime. For each global expert ID, this mapping records the corresponding local expert index on the EP rank that stores the expert and uses a value of -1 on all other ranks. The standard EP dispatch mechanism uses this mapping to ensure that each rank processes only the token-expert activation pairs assigned to its locally stored experts. ExpertWeave extends the same mechanism to the expanded global expert ID space containing both base-model and adapter-specific experts.

Base-model experts retain their original EP placement. For adapter experts, the current implementation uses a coarse-grained placement policy in which all fine-tuned experts of adapter a in a given layer are assigned to EP rank $a \bmod N_{\text{EP}}$, where N_{EP} denotes the EP degree. Only the assigned rank stores valid VMM-backed physical weights for those adapter experts. On all other ranks, their global-to-local mapping entries are set to -1 , and the corresponding reserved virtual address ranges remain unmapped. Therefore, adapter expert weights are not replicated across EP ranks, although this simple placement policy does not guarantee perfectly balanced memory usage across ranks.

Batched rerouting operates on global expert IDs and is performed before EP dispatch. Consequently, an adapter-specific expert selected after rerouting may reside on a different EP rank from the corresponding base-model expert. After the expert IDs are rewritten using the adapter-specific ESFT expert map, the standard EP global-to-local translation determines the destination rank and local expert index for each token-expert pair. The existing EP communication and expert-computation pipeline then dispatches and executes these pairs without modification.

Memory Management APIs. The key AscendCL APIs enabling the virtual weight tensor of Section 4.1, including reserving contiguous virtual address spaces, mapping and unmapping physical pages, and managing device-level physical memory pages, are summarized in Table 4. As shown in Section 5, the use of virtual-memory-assisted expert weight management introduces negligible overhead in ExpertWeave.

API	Short Description
<code>aclrtReserveMemAddress</code>	Reserve virtual memory address space.
<code>aclrtMallocPhysical</code> <code>aclrtFreePhysical</code>	Create physical memory pages on the NPU. Free physical memory pages on the NPU.
<code>aclrtMapMem</code> <code>aclrtUnmapMem</code>	Map physical memory pages to virtual addresses. Unmap physical memory pages from virtual addresses.

Table 4: Key AscendCL APIs used in virtual-memory-assisted expert weight management.

Hardware Compatibility. Our implementation is developed on Ascend NPUs and leverages device-specific APIs for memory management and kernel execution. However, the design of ExpertWeave is hardware-agnostic and follows the abstractions described in Section 4, which do not depend on particular accelerator architectures as long as similar APIs are

provided by the accelerator runtime. While quantitative efficiency gains may vary across hardware backends, the underlying mechanisms of virtual-memory-assisted expert weight management and batched rerouting are not fundamentally tied to Ascend NPUs.

In particular, the virtual weight tensor is implemented using the design described in Section 4, which relies on reserving a contiguous virtual address space and dynamically mapping physical memory for active experts. While our implementation uses AscendCL VMM APIs, a CUDA implementation could use similar VMM primitives including `cuMemAddressReserve`, `cuMemCreate`, `cuMemMap`, and `cuMemUnmap` (NVIDIA, 2026), and a HIP implementation could use `hipMemAddressReserve`, `hipMemCreate`, `hipMemMap`, and `hipMemUnmap` (Advanced Micro Devices, Inc., 2026). The underlying algorithm and layout remain unchanged, and only the platform-specific memory management API calls differ.

Similarly, the batched rerouting kernel is expressed as a sequence of standard parallel operations, including expansion of per-token adapter IDs, arithmetic offset computation, gather from the expert map, and masked selection. These operations are widely supported across accelerator programming models and can be implemented in Triton (Tillet et al., 2019), CUDA, or another accelerator-specific kernel framework without modifying the algorithm.