vExpert: Virtualizing Expert Storage for Adaptive Load Balancing in Distributed MoE Inference

Wenxun Wang[1,3] **Xiuhong Li**[2] **Yida Wang**[2] **Chen Tang**[1] **Zongle Huang**[1,3] **Ke Hong**[1]
Yu Wang[1] **Yongpan Liu**[1,3*]
[1]Tsinghua University [2]Infinigence-AI [3]BNRist
{wx-wang23}@mails.tsinghua.edu.cn {ypliu}@tsinghua.edu.cn

Abstract

Mixture-of-Experts (MoE) models have become prevailing, yet their distributed deployment suffers from severe dynamic load imbalance due to the conflict between static expert placement and inherent routing dynamism. While host DRAM offers capacity to avoid static device storage for load balancing, we identify existing offloading works fundamentally fail in distributed inference due to two overlooked challenges: (1) coupled performance impacts of runtime Host-to-Device transfers due to load dynamism and shared PCIe topology; (2) system-level memory overhead incurred by DRAM usage at scale. In this paper, we propose vEXPERT that shifts DRAM offloading paradigm from partial residency to full redundancy through virtualized expert storage. We pioneer the first systematic performance model that jointly captures H2D latency, PCIe contention, and load imbalance, building a lightweight allocator to dynamically map physical experts to virtual slots runtime. A novel disaggregated manager is designed to reduce memory overhead in multi-process environments. Evaluated on DeepSeek-V3.1 within an 8-node H100 cluster, vEXPERT improves load balance ratio by up to 55% and achieves 1.2× end-to-end speedup.

1 Introduction

The scaling of Large Language Models (LLMs) [3, 22] has established Mixture-of-Experts (MoE) [10, 25, 28] as the *de facto* architecture for efficient inference. To handle massive model sizes, Expert Parallelism (EP) is universally adopted for distributed inference, sharding experts across multiple GPUs [15, 39, 18].

However, the static expert placement contrasts with the inherent dynamism of MoE [9, 34]. As shown in Figure 1, activated experts fluctuates drastically depending on the input [21], leading to severe load imbalance where GPUs hosting hot experts become bottlenecks. Existing systems mitigate this by replicating redundant experts [16, 8, 24]. Yet, this approach exposes a fatal flaw: **the inherent conflict between redundancy capacity and memory constraints.** Redundancy capacity can be viewed as the ratio between the number of experts visible to a GPU and the total number of experts. While GPU

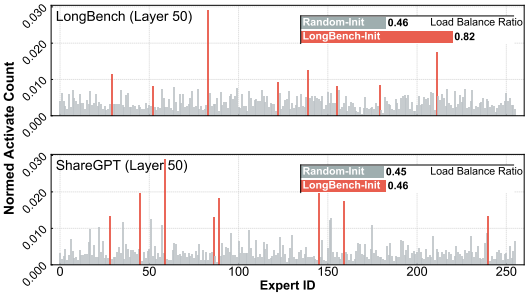


Figure 1: Distribution of activated expert for DeepSeek-V3.1. Random-init and LongBench-init denote expert placement initialized randomly and via LongBench statistics, respectively.

*Corresponding Author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

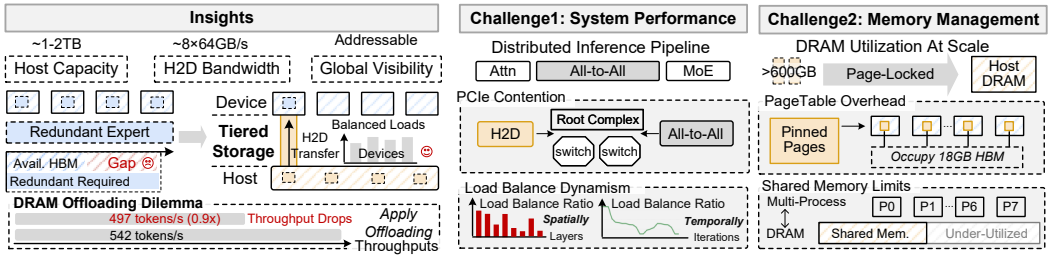


Figure 2: Insights and challenges of utilizing host resources for distributed inference at scale.

HBM is heavily occupied by model parameters and KV caches, it restricts redundancy to merely 1–2 experts per GPU, stripping the system of the elasticity required to handle rapid load drifts.

We identify a *tiered expert storage hierarchy* as the key to decouple redundancy capacity from HBM limits. This insight is motivated by three observations in modern AI servers: First, the underutilized TB-scale host DRAM provides an abundant capacity for full-model redundancy. Second, the aggregated Host-to-Device (H2D) unidirectional bandwidth (e.g., 512GB/s via PCIe Gen5) is competitive with Device-to-Device interconnects (e.g., 450GB/s via NVLink 4.0) [37, 36], rendering runtime weight streaming viable. Third, host memory offers topology symmetry and global visibility, enabling uniform access by all intra-node GPUs. Together, these factors motivate treating host DRAM as a second-level storage to break the coupling between redundancy capacity and device HBM.

A straightforward approach to exploiting host resources is DRAM offloading [13, 4, 14], which adopts a partial residency where a subset of experts resides in HBM. It fetches GPU-missed experts runtime and manages H2D transfers based on overlapping, assuming that H2D transfers incur negligible overhead when masked by concurrent computation. However, applying offloading for distributed inference incurs fundamental challenges as shown in Figure 2.

First, the impact of H2D transfers on **system performance** is inherently complex and structurally coupled. On one hand, distributed MoE inference relies on inter-node All-to-All (A2A) communication. Integrating H2D transfers into inference pipeline causes contention for node-level PCIe resources, leading to negative latency impacts. On the other hand, load balance exhibits dynamism both spatially (layers) and temporally (iterations). DRAM offloading fundamentally fails as it incurs constant PCIe traffic for GPU-miss regardless of actual load distribution and PCIe contention, underscoring the necessity to model the coupled impact on system performance.

Second, leveraging host DRAM at scale incurs **system-level memory overheads** in multi-process servers. Achieving peak H2D bandwidth requires page-locked weights. As each GPU process must maintain its own page table mappings, serving a 671B model consumes GBs of HBM, limiting KV cache capacity—which in turn constrains concurrency and degrades throughput. Besides, exposing the host-resident experts across multiple processes relies on OS-level shared memory, whose default quotas are strictly capped. It reduces the accessible host capacity and prevents full-model redundancy.

To address these challenges, we present vExpert, a virtualized expert storage system for MoE decoding. It proposes a paradigm shift from partial residency to full redundancy by storage abstraction, virtual Expert (vExpert), that decouples expert residency across a tiered Device-HBM and Host-DRAM hierarchy. To guarantee system performance, we pioneer the first systematic analysis of H2D transfers in distributed inference and build a fine-grained performance model that quantifies the coupling impact of H2D latency, PCIe contention, and load imbalance. Guided by this model, we design a lightweight allocator that dynamically remaps physical experts to virtual slots in response to shifting workloads across layers and iterations. To mitigate system-level memory overheads, we implement a novel memory management that decouples data provisioning (coordinator) from consumption (worker). It reduces OS-level addressing overhead and bypasses shared memory limits.

We implement vExpert and evaluate it on an 8-node H100 cluster. Compared to SOTA baselines, vExpert improves load balance by up to 55% under dynamic workloads, translating to up to 20% higher end-to-end decoding throughput. These results demonstrate that vExpert is a practical and effective path toward elastic and robust MoE inference. Code is available at https://github.com/FlyingCloud/vExpert.

2 Related Work

2.1 MoE Serving At Scale

Inference Timeline. MoE layers follow a structured pipeline of four stages in distributed inference [39]: (1) `Attention`, (2) `Dispatch` that scatters tokens to the selected experts, (3) `MoE` that computes the tokens through its local experts, (4) `Combine` that gathers the processed tokens back to their original GPUs. To hide the overhead of communication, two-batch-overlap (TBO) is commonly employed to enhance throughput [8] as shown in Figure 3.

Load Imbalance. The critical path is dominated by MoE and All-to-All. Load imbalance directly impacts the latency of these operations and greatly affects system throughput [17]. Maintaining workload balance in distributed system is crucial yet challenging [27]. An expert placement that is well balanced for one workload can quickly become imbalanced under load drift, rendering current static or offline optimization insufficient. This gap necessitates a system that can adapt to dynamic load shifts which is commonly observed in online serving.

2.2 DRAM Offloading

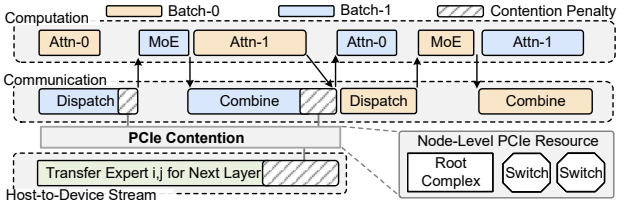


Figure 3: Distributed inference timeline with H2D transfers.

DRAM offloading has been explored to utilize host resources for efficient LLM inference [19, 40, 12, 35, 20]. These systems typically adopt a partial residency paradigm [4, 26], partitioning weights between GPU HBM and host DRAM, where H2D transfers are triggered by GPU misses. To minimize overhead, they rely on overlapping that assumes H2D transfers incurs zero overhead if masked by concurrent computation. While viable for single-node or single-device setups, it fails in distributed inference as H2D transfers compete for limited node-level PCIe resources with `Dispatch` and `Combine`. It exacerbates physical contention on the critical path, making constant H2D transfer sub-optimal for system performance. [5] furher offloads both storage and computation to CPUs, exploiting available CPU compute resources to meet latency requirements. Though suitable for small-batch edge serving, the limited CPU capabilities cannot sustain the strict latency demands of online serving under massive concurrency (e.g., batch size > 100). Consequently, DRAM offloading has primarily been restricted to the latency-tolerant prefillling or training [13, 23, 33]. Our work pioneeringly leverages host resources for distributed MoE decoding, revealing and addressing challenges in system performance and memory management.

3 vExpert Paradigm

3.1 System Architecture

Overview. We introduce vExpert abstraction to decouple expert residency from GPU memory constraints. vExperts are logically visible to GPUs as locally available experts but physically reside in host DRAM. In this paradigm, the entire expert pool is exposed and addressable to every GPU process, thereby enabling non-blocking, dynamic configuration of redundant experts to adapt to shifting workloads. GPUs maintain a minimal set of vExpert Slots—HBM buffers that serve as destination targets for H2D transfers. The vExpert architecture establishes a tiered expert storage hierarchy (Figure 4):

L1:Device HBM. L1 does not store static expert replicas. Instead, it serves as a dynamic cache for weights required by the current iteration and layer.

L2:Host DRAM. Leveraging the high capacity of DRAM, L2 maintains a full-coverage physical expert pool in the model.

Pre-Allocating Pipeline. vExperts are managed via a layer-wise pre-allocating pipeline to mask the H2D latency in tiered hierarchy. During the execution of layer i, system asynchronously initiates H2D transfers for the vExperts required by layer $i + 1$. By overlapping H2D transfer with the computation of the preceding layer, vExperts are available in their designated slots just-in-time for execution. As a

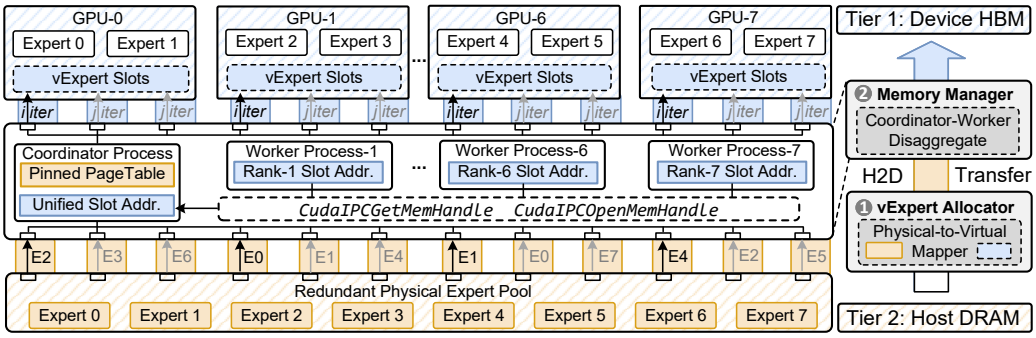


Figure 4: Overview of the proposed VEXPERT storage hierarchy and system design.

result, GPUs only need to maintain buffers sufficient to hold the virtual experts of two layers (current and next).

vExpert Allocator. The system utilizes the allocator to dynamically resolve the optimal mapping between L2 physical weights and L1 virtual slots, specifying the per-layer vExpert number and the physical expert ID assigned to each GPU. Figure 4 shows an example: in iteration i, each device hold a single vExpert and slot of GPU-0 is mapped to physical expert E2. In later iteration j, allocator configures two vExperts per device, with GPU-0 mapping to E3 and E6.

Memory Manager. The manager implements a disaggregated scheme. A single coordinator process registers the global pinned memory for L2 physical pool and orchestrates H2D transfers. Worker processes allocate their local vExpert slot buffers at L1, which are made visible to coordinator via CUDA Inter-Process Communication (IPC). It allows coordinator to perform transfers into slots of any workers. Details of each component are presented in Section 4.

3.2 Performance Modeling and Optimization

We develop a principled performance model that quantifies the impacts of vExperts.

3.2.1 Latency Model Formulation

The latency of MoE layer is typically dominated by three sequential operations: `Dispatch`, `MoE`, and `Combine`. Their aggregate duration become the primary determinant of overall latency in the pipeline. We therefore define the baseline per-layer latency as:

$$T_i(L_i) = T_{i,\text{dispatch}}(L_i) + T_{i,\text{moe}}(L_i) + T_{i,\text{combine}}(L_i) \tag{1}$$

$L_i = \frac{\text{AvgTokensPerGPU}}{\text{MaxTokensPerGPU}} \in [0,1]$ represents the load balance ratio, which determines the token number $B_{\max} = B/L_i$ in the most heavily loaded GPU (B is the batch size). Both communication and computation are bottlenecked by $B_{\max}$ in the cluster.

Extension with Virtual Experts. Introducing V virtual experts per layer modifies the latency by: (1) improved load balance ratio (L_i), and (2) injecting H2D transfer overhead. To simplify the analysis without loss of generality, we aggregate the `Dispatch` and `Combine` into a single term $T_{i,\text{a2a}}$, the enhanced latency model becomes:

$$T_i^*(L_i,V) = T_{i,\text{a2a}}(L_i) + T_{i,\text{moe}}(L_i) - \underbrace{T_{i,\text{a2a}} \cdot \gamma_1(L_i,V)}_{\text{communication gain}} - \underbrace{T_{i,\text{moe}} \cdot \gamma_2(L_i,V)}_{\text{computation gain}} + \underbrace{T_{\text{H2D}}(V) \cdot \eta}_{\text{H2D overhead}} \tag{2}$$

γ_1 and γ_2 represent the latency reduction ratios. $T_{\text{H2D}}(V)$ is the latency to transfer V experts, and η captures the negative impact of PCIe contention. We proceed to analyze and profile each component based on setup in Section 5.1.

3.2.2 Positive Impact: Load Balancing Gains

All-to-All Communication Gain (γ_1). The latency of All-to-All scales with the maximum token number $B_{\max}$ handled by any rank [16]. A higher balance ratio L decreases $B_{\max}$, leading to the

reduction of latency as:

$$\gamma_1(L_i, V) \approx 1 - \frac{B/(L_i + \Delta_L)}{B/L_i} = \frac{\Delta(L_i, V)}{L_i + \Delta(L_i, V)} \tag{3}$$

$\Delta(L_i, V)$ denotes the improvements in load balance.

MoE Computation Gain (γ_2). L_i directly affects the TokensPerExpert (TPE, referred to $B_{\max}$) for MoE GEMM kernels. The computation gain of MoE computation exhibits workload-dependent behavior [1]: **(1) Compute-Bound:** For large TPE (TPE $\geq$ 160 in our profiling), MoE GEMMs are compute-bound, and latency scales proportionally with the maximum token count. In this regime, $\gamma_2 \approx \gamma_1$. **(2) Memory-Bound:** For small TPE, MoE GEMMs are memory-bound where memory access overheads dominate. Reducing the token count yields diminishing returns as $\gamma_2 \approx 0$.

We provide detailed profiling for these operations in Appendix B.2 for validation.

Load Balance Improvement (Δ). $\Delta(L_i, V)$ is a complex function of: distribution of activated experts, initial load imbalance L_i, and vExpert number V. Closed-form modeling is intractable due to the stochastic nature of online request distributions. Instead, we characterize this relationship through offline profiling on representative datasets. Our observations indicate that for a fixed V, Δ can be approximated via a linear function as shown in Figure 5. Therefore, we approximate Δ via a first-order linear model:

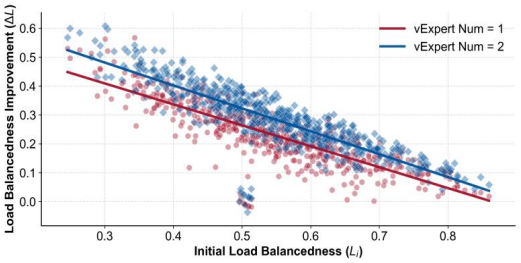


Figure 5: Analysis of load balance improvement (Δ) through virtual experts on DeepSeek-V3.1.

$$\Delta(L_i, V) = \alpha \cdot L_i + \beta \cdot V + k \tag{4}$$

Profiled on representative datasets, the fitted parameters are $\alpha, k, \beta = -0.75, 0.1, 0.53$. This empirical characterization captures load balancing effects for each iteration and layer.

3.2.3 Negative Impact: H2D Transfer Overhead

Path Topology and Interference. An non-intuitive challenge is the physical resource contention between H2D transfer and All-to-All communication [38]. All-to-All typically traverses the GPU $\leftrightarrow$ PCIe Switch $\leftrightarrow$ NIC $\leftrightarrow$ PCIe Switch $\leftrightarrow$ GPU. H2D transfer traverses the CPU $\rightarrow$ PCIe Switch $\rightarrow$ GPU. While PCIe supports full-duplex, both traffic flows compete for the resources of PCIe Switch and Root Complex. We model this interference via an additive penalty $T_{\text{H2D}}(V) \cdot \eta$, where T_{H2D} is both the time of H2D transfer and the duration of bandwidth contention, η reflects the contention loss for All-to-All. We derive η and T_{H2D} via benchmarking concurrent bandwidth in Appendix B.3.

H2D Latency (T_{H2D}). Transferring V experts from Host to Device costs $T_{\text{H2D}}(V) = \frac{V \cdot M_{\text{expert}}}{\text{BW}_{\text{H2D}}}$, where M_{expert} is the expert weight size and BW_{H2D} is the achievable PCIe bandwidth under contention. On H100-servers with PCIe Gen5 x16, we achieve $\text{BW}_{\text{H2D}} \approx 26$ GB/s per GPU.

Contention Factor (η). We observe an asymmetry in interference sensitivity: while contention degrades H2D transfer by 40%, its impact on All2All is relatively minor, yielding $\eta \approx 0.19$. This asymmetry can be attributed to the underlying physical topology. Cross-node All2All is a multi-stage process that depends not only on PCIe but also on network fabric (e.g., InfiniBand) traversal. Consequently, the local latency penalty introduced by contention is amortized.

3.2.4 Optimization Problem Definition

We formulate the selection of virtual experts as a constrained optimization problem in Equation 5. The objective is to minimize the layer latency T_i. A critical constraint is that the H2D transfer time must be hidden within the execution time of the previous layer to prevent stalls. This yields an upper bound $V_{\max}$ on the number of virtual experts. Since the search space $[0, V_{\max}]$ is small (typically $V_{\max} < 3$), we employ a greedy traversal strategy at runtime to solve for V^* efficiently without incurring solver overhead.

$$V^* = \underset{V \in \mathbb{Z}_{\geq 0}}{\text{argmin}}\ T_i^*(L_i, V) \quad \text{s.t.} \quad T_{\text{H2D}}(V) \leq T_i \implies V \leq V_{\max} \tag{5}$$

Generalizability. The gain (γ_1, γ_2, Δ) and PCIe contention (BW_{H2D}, η) parameters are inherently hardware- and model-specific, with numerical values calibrated based on the setup in Section 5.1. Crucially, the allocator adapts to diverse setups with minimal effort and without any architectural modifications. For new models, load-balancing parameters α, β, and k are rapidly re-calibrated via offline forward-pass tracing in under 30 minutes. For new hardware platforms, contention parameters η and BW_{H2D} are seamlessly updated using microbenchmarks. A detailed evaluation of cross-model and cross-platform generalizability is provided in Appendix B.5 and Section 5.3.

4 System Implementation

4.1 vExpert Allocator

We detail the allocator design from three perspectives: (1) group-wise budgeting strategy that expands the optimization space beyond single-layer constraints; (2) physical-to-virtual mapping algorithm; and (3) non-blocking pipeline integration that applies these decisions with negligible runtime overhead.

Group-wise Budgeting. Per-layer vExpert budget is constrained by the overlap requirement: H2D latency must be masked by the time of previous layer, typically restricting $V_{max} \approx 1$. Our insight is that load distribution are heterogeneous across layers. Layers with high L_i (e.g. $L_i > 0.7$) gains little from vExpert, while imbalanced layers (e.g. $L_i < 0.5$) benefit greatly as shown in Figure 5.

Therefore, we introduce **group-wise slots** that partitions G consecutive layers into a single group. The constraint is relaxed from a per-layer limit to a group-wise budget: $\sum_{i=1}^{G} V_i \leq V^*_{\max} = G \cdot V_{\max}$, V_i is the number of virtual experts for layer i. It masks H2D latency behind the computation time of G layers, allowing well-balanced layers to donate their slots to imbalanced layers and expanding the solution space. This strategy requires a GPU buffer size of $G \cdot V_{\max} \cdot M_{\text{expert}}$. The choice of G represents a trade-off between optimization space and memory overhead. In practice, we set $G = 2$ to achieve load balance with architectural simplicity and modest overhead (see Appendix D).

Physical-to-Virtual Mapper. We develop a three-step mapping with details provided in Appendix E.

Step 1: Number Assignment. To determine the optimal budget, we first evaluate the L_i of each layer within a group based on historical statistics. Well-balanced layers ($L_i > 0.7$) are assigned $V_i = 0$, and their slots are redistributed to imbalanced layers, whose V_i are generated via solving Equation 5.

Step 2: Candidate Selection. To identify the expert candidates for the assigned budget, for each layer i, we select the top $V_i \times N_{gpu}$ hottest physical experts from host DRAM based on historical statistics.

Step 3: Greedy Rebalance. To ensure load balance across devices, we employ a greedy bin-packing approach to map candidates to physical GPUs: (i) Rank GPUs by their current token loads; (ii) Iteratively assign the virtual expert with the highest pending load to the GPU with the lowest current load; (iii) Repeat until all V_i slots for the layer are filled.

Pipeline Integration. To minimize interference with critical path, we integrate vExpert allocation into inference pipeline as:

Lightweight vExpert Allocation. The system triggers allocation every k iterations. The Allocator consumes the latest activation statistics and determines the optimal mapping. We implement the allocation using efficient CUDA kernels, enabling the entire allocation to complete within 10 ms.

Non-Blocking Graph Update. When new vExpert info is ready, the system updates CUDA graph by modifying the `SrcPtr` field of `MemCpy` nodes through CUDA runtime API to avoid expensive recapturing. Since CUDA graphs support asynchronous updates, this process is non-blocking.

Synchronization. Once current batch finishes, GPU switches to the updated graph and executes the next batch. This pipeline enables adaptation to dynamic workloads with near-zero overhead. Concretely, it introduces at a 10 ms overhead every k iterations; with $k \geq 10$, the per-iteration overhead is below 1 ms, negligible in practice. Thus, we set $k = 10$ in our default implementation.

4.2 Coordinator–Worker Disaggregated Management

Deploying tiered storage in multi-process environments faces two critical system-level problems:

Pinned memory overhead. Achieving peak H2D bandwidth requires registering weights as pinned (page-locked) to enable DMA zero-copy transfers, which forces GPUs to maintain a page table mapping physical pages to accessible addresses. This incurs a non-trivial memory overhead. For instance, DeepSeek-V3.1 consumes ∼2GB VRAM per rank, totaling 16GB for an 8-GPU node.

Shared memory limits. To make CPU-resident experts visible across multiple processes, standard implementations rely on shared memory, which is typically capped at 50% of physical RAM. This is insufficient for full redundancy of massive models, especially with NUMA-duplicating to avoid cross-socket penalties: DeepSeek-V3.1 requires 1.3TB while the usable shared memory is 900GB.

The root cause of these overheads is the enforcement of a uniform access model: every GPU process requires both read and write access —read for its own GEMM operations and write (initiating H2D) to populate HBM vExpert slots.

Disaggregated Architecture. We propose a **Coordinator–Worker** architecture to bypass these limits. The core principle is an IO-centralized coordinator to hold the responsibility for data provisioning (pinning memory, initiating H2D copies). The worker processes only require read access to their local GPU buffers for computation. This separation eliminates redundant host memory pinning and bypasses the need for OS shared memory. As shown in Figure 4, we designate Rank 0 as the coordinator process and Ranks $1-7$ as workers within a node:

Coordinator: The coordinator allocates and pins the expert pool in host memory. It uses CUDA IPC to import worker GPU buffers via `cudaIpcOpenMemHandle` and writes via `cudaMemcpyAsync`.

Worker: Each worker allocates its local GPU ping-pong buffer for caching vExpert weights, exports handles via `cudaIpcGetMemHandle` and shares them with coordinator during initialization. It focuses exclusively on kernel execution once coordinator signals data readiness.

A critical design choice is the number of pinned physical experts (N_p) maintained in the L2 storage. Increasing N_p enhances redundancy capacity and improves load balance. However, it comes at the cost of higher DRAM consumption. NUMA locality further complicates this trade-off: duplicating experts across NUMA domains can mitigate cross-NUMA penalties but doubles the DRAM footprint. We provide a detailed analysis of this trade-off in Section 5.3 and store a single copy of every expert in L2 storage without NUMA duplication as default.

5 Evaluation

5.1 Evaluation Setup

Hardware. Experiments are conducted on an 8-node cluster; each node features 8×H100-NVLink GPUs, 1.8 TB of DDR5-4800 RAM, and 4×400 Gbps InfiniBand interconnects. This setup represents a typical high-end production environment. Additional details are provided in Appendix A.1.

Workloads and Model. To emulate dynamic online serving and load drifts, we construct a synthetic workload from multiple publicly available datasets: ShareGPT [30], LongBench [2] and LiveCodeBench [11]. ShareGPT provides real-world conversations. LongBench is used to simulate long-context scenarios with lengthy system prompts. LiveCodeBench is employed for code-generation workloads. Experiments are conducted using 671B DeepSeek-V3.1 [8] by default.

Baseline. We use SGLang, a state-of-the-art open-source LLM serving system for large-scale cluster inference, as our primary baseline. All decode instances are configured following the best-practice strategies recommended in [31] to ensure a strong baseline (refer to Appendix A.2 for details). vEXPERT is compared against:

(1) **B0 (Vanilla)**: Vanilla SGLang deployment without explicit load balancing; (2) **B1 (EPLB)**: SGLang with Expert-Parallelism Load Balancing [7], which rebalances placement periodically based on historical statistics; (3) **B2 (Redundant)**: EPLB augmented with one GPU-resident redundant expert per device, representing a typical HBM-constrained redundancy strategy; (4) **Offload**: Offloads one expert per device to CPUs, triggering H2D transfers of one expert per layer [13].

The choice of rebalance frequency for B1 and B2 involves a trade-off between load balance and rebalance-blocking overhead. Given that EPLB incurs a stall of $\geq$ 1s per rebalance [39], we configure B1 and B2 with rebalance iterations of 500 and 200 as low and high-frequency.

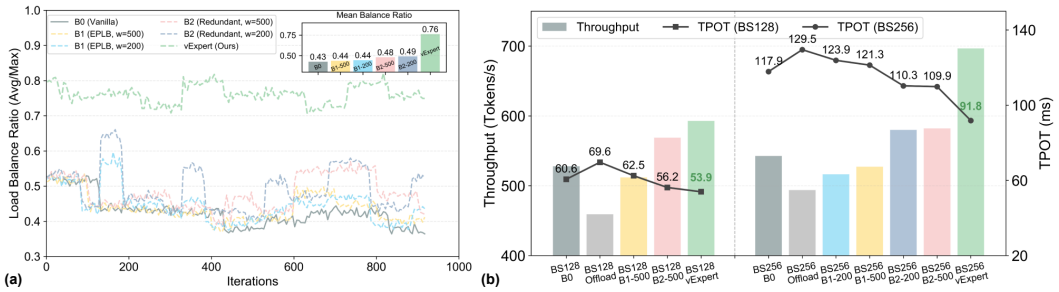


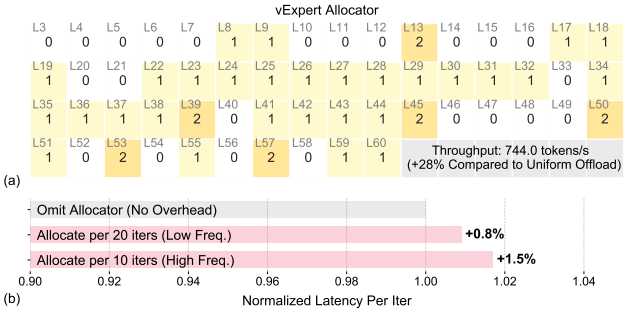
Figure 6: (a) Comparison of load balance ratio over iterations at batchsize 256, w=w denotes rebalancing every w iterations.(b) End-to-end throughput (*bars, left axis*) and Time-Per-Output-Token (TPOT) (*lines, right axis*) across different batch sizes, B-w denotes rebalancing every w iterations.

5.2 Evaluation Results

Effectiveness in Load Balance. We evaluate vEXPERT in maintaining load balance under dynamic load drifts, using **load balance ratio** as metric. As shown in Figure 6 (a), the vanilla baseline (B0) suffers from severe imbalance, with balance ratio at 0.43. While B1 and B2 provide marginal improvements (0.44 and 0.49), they are constrained by HBM capacity and the stalls induced by periodic rebalancing, failing to handle rapid load drifts. In contrast, vEXPERT increases redundancy capacity and maintains load balance ratio at 0.76, improving by 55% compared to the best baseline.

End-to-end Performance. Figure 6 (b) highlights the end-to-end throughput gains. Due to constant H2D transfers, the penalty reduces throughput of Offload by 10% compared to B0. B1 suffers degradation compared to B0 since the overhead of rebalancing stalls outweigh the load-balancing benefits. B2 achieves a modest 7% gain via HBM-resident redundancy but faces diminishing returns at higher rebalancing frequencies, thus limiting its adaptability to dynamic workloads. By contrast, vExpert allocator enables seamless adaptation, translating to 28.4% improvement at batch size 256 and 12.4% at batch size 128 over B0.

The performance variance across batch sizes is attributed to H2D transfer overhead. Since T_{H2D} depends only on the number of virtual experts V, its contribution to inference latency increases as batch size and TPOT decrease. Despite this, vEXPERT consistently outperforms all baselines, demonstrating its robustness in distributed environments.



(c) Layer-wise allocation

Slots per Layer	# Layers	Before/After
0	23	0.697/0.697
1	29	0.438/**0.763**
2	6	0.347/**0.768**

Figure 7: Allocator evaluation for DeepSeek-V3.1 on an 8-node H100 cluster at batch size 256. (a) Layer-wise assignment and throughput. (b) Allocator overhead. (c) Allocation across Layers 3–60. Before/After denotes the mean balance ratio before and after allocation.

5.3 Ablation Study

Efficiency of Allocator. Compared to uniformly offloading ($V_i = 1$), our allocator yields a 28% throughput gain by dynamically assigning virtual experts across layers, reflecting the layer-wise variation in load imbalance (Figure 7 (a)). To illustrate how the allocator adapts to this variation, Figure 7 (c) summarizes the allocation across Layers 3–60 in one DeepSeek-V3.1 iteration. The allocator assigns 41 virtual expert slots to 35 imbalanced layers while leaving the other 23 relatively balanced layers unchanged, avoiding unnecessary H2D transfers for these layers. Layers receiving

Table 1: Effect of ablating memory manager on memory usage and system throughput measured at batchsize 256, "N/A" indicates a system crash or OOM due to memory constraints, "NUMA" denotes duplicating expert weight across two NUMA domains.

Method	Metric (Per Node)	Pinned Physical Expert Configuration			
		128	128-NUMA	256	256-NUMA
Disable Memory Manager	GPU Mem Alloc.	6.9G	10.4G	11.9G	18.4G
	CPU Mem Alloc./Avail.	313G/900G	626G/900G	646G/900G	1.3T/900G
	Throughput	1.00×	1.02×	1.05×	**N/A(OOM)**
Enable Memory Manager	GPU Mem Alloc.	0.9G	1.3G	1.5G	2.3G
	CPU Mem Alloc./Avail.	313G/1.8T	626G/1.8T	646G/1.8T	1.3T/1.8T
	Throughput	1.00×	1.02×	**1.05**×	**1.06**×

one and two slots improve their mean balance ratios from 0.438 to 0.763 and from 0.347 to 0.768, respectively. It demonstrates the effectiveness of adapting the number of virtual experts to each layer's load imbalance.

We also evaluate the overhead of allocator as shown in Figure 7 (b). The penalty of enabling allocator is negligible (less than 2%). In addition, we provide detailed breakdown of allocator overhead in Appendix F. While both 10-iter and 20-iter introduce minimal overhead, 10-iter provides superior adaptability to rapid load shifts. Consequently, we select 10-iter as default to ensure both agility and efficiency.

Efficiency of Memory Manager. We evaluate the manager on resource utilization and system throughput across different configurations.

1. GPU Memory Savings. As shown in Table 1, enabling the manager leads to an $8\times$ reduction in GPU memory overhead. By centralizing management within the coordinator, vExpert avoids per-process pagetable-metadata overhead, with negligible HBM usage of less than 300MB per GPU.

2. CPU Memory Utilization. The manager enhances scalability by bypassing shared-memory limits. Without the manager, the system is constrained by the typical 50% allocation of physical RAM for shared memory. vExpert utilizes the full DRAM capacity, allowing for higher redundancy capacity, which is critical for dynamic workloads.

3. Redundancy Capacity and NUMA Awareness. Increasing the number of pinned experts improves throughput as a larger redundant expert pool increases the hit rate of hot experts in L2. Furthermore, replicating experts across NUMA domains also enhances performance. Given that it doubles CPU memory usage, we select 256 as default with modest memory usage and throughput.

Scalability with EP. Increasing the EP degree distributes expert ownership across more ranks, making execution more sensitive to routing skew: a rank owning a hot expert can become a more pronounced straggler in A2A communication and MoE computation. We conduct experiments by evaluating vExpert at EP32 and EP64 on DeepSeek-V3.1 with batch size 256. When scaling from EP32 to EP64, B2's balance ratio decreases from 0.57 to 0.49, and its throughput drops by 11%. vExpert improves the balance ratio to 0.81 and 0.76 and increases throughput over B2 by 11% and 20%, respectively. These results show that vExpert remains effective as EP scales and expert ownership becomes more fragmented, with larger relative gains under more severe load imbalance.

Performance Model Robustness. Table 2 presents the sensitivity to $\pm 20\%$ perturbations in each performance model parameter on DeepSeek-V3.1. Remarkably, even with substantial parameter errors, throughput degradation remains below 3% in all cases. It demonstrates that vExpert does not require high-precision calibration to deliver near-optimal performance. The root cause is that the allocator's primary decision variable—the number of virtual experts—is a **discrete integer** that naturally absorbs continuous perturbations.

Cross-Model Generalizability. Table 3 reports the results on different model KIMI-K2 [29]. We also implement a Simplified model that omits γ_2 and fixes the load-balance parameters α, β, k to constants derived from DeepSeek-V3.1 profiling. The Simplified Model improves throughput by $+\mathbf{21}\%$. With full calibration, the gain rises to $+\mathbf{29}\%$. These results confirm that the structural insights captured by performance model are transferable across models. Results of Simplified Model further show that a coarse approximation of H2D impacts is sufficient to outperform offloading, underscoring that the

Table 2: Throughput impact of parameter errors.

Para.	-20% Error	$+20\%$ Error
α	-1.2%	-2.7%
β	-2.5%	-0.7%
k	-2.4%	-2.8%
BW_{H2D}	-0.9%	-1.4%
η	-1.2%	-2.6%

Table 3: Evaluation on KIMI-K2 (BatchSize 128, 4-Node H100 Cluster).

Method	Balance Ratio	Throughput(tok/s)
B2-500	0.42	545
Offload [13]	0.59	529
vEXPERT(Simplified)	0.64	659 ($+\mathbf{21}\%$)
vEXPERT(Calibrated)	0.69	703 ($+\mathbf{29}\%$)

joint modeling of PCIe contention, H2D latency, and load distribution is the critical enabler of robust performance.

6 Conclusion and Limitation

This work pioneeringly reveals and resolves challenges for exploiting host resource in distributed MoE inference. By building systematic performance model and memory manager, it enables fine-grained and adaptive load balance, achieving up to 20% throughput improvement and highly stable load balance under dynamic workloads. While vEXPERT targets the decode phase where host resources are underutilized, it complements GPU-based load-balancing methods (e.g., EPLB) in the prefill phase, as the longer per-forward latency in prefilling amortizes expert reordering overheads and host memory can be repurposed for prefix caching. Additionally, our analysis assumes conventional server topologies with PCIe/NVLink hierarchies. Emerging supernode architectures [41, 6] featuring advanced packaging and high-bandwidth interconnects may reshape the host–device landscape. For example, GB200 NVL72 introduces a fundamentally different hierarchy: coherent NVLink-C2C connects Grace CPUs and Blackwell GPUs, while up to 72 GPUs form a rack-scale NVLink domain. This higher-bandwidth CPU–GPU path could reduce host-expert transfer overhead, making host-backed expert virtualization promising on such systems. Adapting vExpert to advancing topology is an important direction for our future work.

Acknowledgments

This work was supported by National Key Research and Development Program of China under Grant (2024YFB4405400).

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve, 2024.

[2] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[4] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 715–730, New York, NY, USA, 2025. Association for Computing Machinery.

[5] Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan

Ai, Yongwei Wu, and Mingxing Zhang. Ktransformers: Unleashing the full potential of cpu/gpu hybrid inference for moe models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, page 1014–1029, New York, NY, USA, 2025. Association for Computing Machinery.

[6] NVIDIA Corporation. *Nvidia Blackwell Architecture Technical Brief*. NVIDIA Technical Documentation, 2024.

[7] DeepSeek-AI. Expert parallelism load balancer (eplb). GitHub repository, 2025.

[8] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025.

[9] Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale, 2023.

[10] 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.

[11] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024.

[12] Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models, 2025.

[13] Yechan Kim, Hwijoon Lim, and Dongsu Han. Scaling beyond the gpu memory limit for large mixture-of-experts model training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[14] Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Xiaozhou Ye, Ye Ouyang, Linghe Kong, and Yunxin Liu. Swapmoe: Serving off-the-shelf moe-based large language models with tunable memory budget. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6710–6720, 2024.

[15] 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, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[16] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed MoE training and inference with lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 945–959, Boston, MA, July 2023. USENIX Association.

[17] Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating distributed moe training and inference with lina, 2024.

[18] Yan Li, Zhenyu Zhang, Zhengang Wang, Pengfei Chen, and Pengfei Zheng. Semantic parallelism: Redefining efficient moe inference via model-data co-scheduling, 2026.

[19] Jiacheng Liu, Peng Tang, Wenfeng Wang, Yuhang Ren, Xiaofeng Hou, Pheng-Ann Heng, Minyi Guo, and Chao Li. A survey on inference optimization techniques for mixture of experts models, 2025.

[20] Qingxiu Liu, Yongchao He, Runhan Jiang, Zion Wang, Bohan Zhao, Mi Zhang, and Patrick P. C. Lee. Fluxmoe: Decoupling expert residency for high-performance moe serving, 2026.

[21] Ziming Liu, Boyu Tian, Guoteng Wang, Zhen Jiang, Peng Sun, Zhenhua Han, Tian Tang, Xiaohe Hu, Yanmin Jia, Yan Zhang, He Liu, Mingjun Zhang, Yiqi Zhang, Qiaoling Chen, Shenggan Cheng, Mingyu Gao, Yang You, and Siyuan Feng. Expert-as-a-service: Towards efficient, scalable, and robust large-scale moe serving, 2025.

[22] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report, 2024.

[23] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA, February 2025. USENIX Association.

[24] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation AI scale. *CoRR*, abs/2201.05596, 2022.

[25] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017.

[26] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. FlexGen: High-throughput generative inference of large language models with a single GPU. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 31094–31116. PMLR, 23–29 Jul 2023.

[27] StepFun, :, Bin Wang, Bojun Wang, Changyi Wan, Guanzhe Huang, Hanpeng Hu, Haonan Jia, Hao Nie, Mingliang Li, Nuo Chen, Siyu Chen, Song Yuan, Wuxun Xie, Xiaoniu Song, Xing Chen, Xingping Yang, Xuelin Zhang, Yanbo Yu, Yaoyu Wang, Yibo Zhu, Yimin Jiang, Yu Zhou, Yuanwei Lu, Houyi Li, Jingcheng Hu, Ka Man Lo, Ailin Huang, Binxing Jiao, Bo Li, Boyu Chen, Changxin Miao, Chang Lou, Chen Hu, Chen Xu, Chenfeng Yu, Chengyuan Yao, Daokuan Lv, Dapeng Shi, Deshan Sun, Ding Huang, Dingyuan Hu, Dongqing Pang, Enle Liu, Fajie Zhang, Fanqi Wan, Gulin Yan, Han Zhang, Han Zhou, Hanghao Wu, Hangyu Guo, Hanqi Chen, Hanshan Zhang, Hao Wu, Haocheng Zhang, Haolong Yan, Haoran Lv, Haoran Wei, Hebin Zhou, Heng Wang, Heng Wang, Hongxin Li, Hongyu Zhou, Hongyuan Wang, Huiyong Guo, Jia Wang, Jiahao Gong, Jialing Xie, Jian Zhou, Jianjian Sun, Jiaoren Wu, Jiaran Zhang, Jiayu Liu, Jie Cheng, Jie Luo, Jie Yan, Jie Yang, Jieyi Hou, Jinguang Zhang, Jinlan Cao, Jisheng Yin, Junfeng Liu, Junhao Huang, Junzhe Lin, Kaijun Tan, Kaixiang Li, Kang An, Kangheng Lin, Kenkun Liu, Lei Yang, Liang Zhao, Liangyu Chen, Lieyu Shi, Liguo Tan, Lin Lin, Lin Zhang, Lina Chen, Liwen Huang, Liying Shi, Longlong Gu, Mei Chen, Mengqiang Ren, Ming Li, Mingzhe Chen, Na Wang, Nan Wu, Qi Han, Qian Zhao, Qiang Zhang, Qianni Liu, Qiaohui Chen, Qiling Wu, Qinglin He, Qinyuan Tan, Qiufeng Wang, Qiuping Wu, Qiuyan Liang, Quan Sun, Rui Li, Ruihang Miao, Ruosi Wan, Ruyan Guo, Shangwu Zhong, Shaoliang Pang, Shengjie Fan, Shijie Shang, Shilei Jiang, Shiliang Yang, Shiming Hao, Shuli Gao, Siming Huang, Siqi Liu, Tiancheng Cao, Tianhao Cheng, Tianhao Peng, Wang You, Wei Ji, Wen Sun, Wenjin Deng, Wenqing He, Wenzhen Zheng, Xi Chen, Xiangwen Kong, Xianzhen Luo, Xiaobo Yang, Xiaojia Liu, Xiaoxiao Ren, Xin Han, Xin Li, Xin Wu, Xu Zhao, Yanan Wei, Yang Li, Yangguang Li, Yangshijie Xu, Yanming Xu, Yaqiang Shi, Yeqing Shen, Yi Yang, Yifei Yang, Yifeng Gong, Yihan Chen, Yijing Yang, Yinmin Zhang, Yizhuang Zhou, Yuanhao Ding, Yuantao Fan, Yuanzhen Yang, Yuchu Luo, Yue Peng, Yufan Lu, Yuhang Deng, Yuhe Yin, Yujie Liu, Yukun Chen, Yuling Zhao, Yun Mou, Yunlong Li, Yunzhou Ju, Yusheng Li, Yuxiang Yang, Yuxiang Zhang, Yuyang Chen, Zejia Weng, Zhe Xie, Zheng Ge, Zheng Gong, Zhenyi Lu, Zhewei Huang, Zhichao Chang, Zhiguo Huang, Zhirui Wang, Zidong Yang, Zili Wang, Ziqi Wang, Zixin Zhang, Binxing Jiao, Daxin Jiang, Heung-Yeung Shum, and Xiangyu Zhang. Step-3 is large yet affordable: Model-system co-design for cost-effective decoding, 2025.

[28] 5 Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, Yean Cheng, Yifan An, Yilin Niu, Yuanhao Wen, Yushi Bai, Zhengxiao Du, Zihan Wang, Zilin Zhu, Bohan Zhang, Bosi Wen, Bowen Wu, Bowen Xu, Can Huang, Casey Zhao, Changpeng Cai, Chao Yu, Chen Li, Chendi Ge, Chenghua Huang, Chenhui Zhang, Chenxi Xu, Chenzheng Zhu, Chuang Li, Congfeng Yin, Daoyan Lin, Dayong Yang, Dazhi Jiang, Ding Ai, Erle Zhu, Fei Wang, Gengzheng Pan, Guo Wang, Hailong Sun, Haitao Li, Haiyang Li, Haiyi Hu, Hanyu Zhang, Hao Peng, Hao Tai, Haoke Zhang, Haoran Wang, Haoyu Yang, He Liu, He Zhao, Hongwei Liu, Hongxi Yan, Huan Liu, Huilong Chen, Ji Li, Jiajing Zhao, Jiamin Ren, Jian Jiao, Jiani Zhao, Jianyang Yan, Jiaqi Wang, Jiayi Gui, Jiayue Zhao, Jie Liu, Jijie Li, Jing Li, Jing Lu, Jingsen Wang, Jingwei Yuan, Jingxuan Li, Jingzhao Du, Jinhua Du, Jinxin Liu, Junkai Zhi, Junli Gao, Ke Wang, Lekang Yang, Liang Xu, Lin Fan, Lindong Wu, Lintao Ding, Lu Wang, Man Zhang, Minghao Li, Minghuan Xu, Mingming Zhao, Mingshu Zhai, Pengfan Du, Qian Dong, Shangde Lei, Shangqing Tu, Shangtong Yang, Shaoyou Lu, Shijie Li, Shuang Li, Shuang-Li, Shuxun Yang, Sibo Yi, Tianshu Yu, Wei Tian, Weihan Wang, Wenbo Yu, Weng Lam Tam, Wenjie Liang, Wentao Liu, Xiao Wang, Xiaohan Jia, Xiaotao Gu, Xiaoying Ling, Xin Wang, Xing Fan, Xingru Pan, Xinyuan Zhang, Xinze Zhang, Xiuqing Fu, Xunkai Zhang, Yabo Xu, Yandong Wu, Yida Lu, Yidong Wang, Yilin Zhou, Yiming Pan, Ying Zhang, Yingli Wang, Yingru Li, Yinpei Su, Yipeng Geng, Yitong Zhu, Yongkun Yang, Yuhang Li, Yuhao Wu, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yuxuan Zhang, Zezhen Liu, Zhen Yang, Zhengda Zhou, Zhongpei Qiao, Zhuoer Feng, Zhuorui Liu, Zichen Zhang, Zihan Wang, Zijun Yao, Zikang Wang, Ziqiang Liu, Ziwei Chai, Zixuan Li, Zuodong Zhao, Wenguang Chen, Jidong Zhai, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025.

[29] Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Chenxiao Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Qizheng Gu, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Yunjia He, Chao Hong, Hao Hu, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yang Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Haoyu Lu, Lijun Lu, Yashuo Luo, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Zeyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Lin Sui, Xinjie Sun, Flood Sung, Yunpeng Tai, Heyi Tang, Jiawen Tao, Qifeng Teng, Chaoran Tian, Chensi Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Si Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Haoning Wu, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Jing Xu, Jing Xu, Junjie Yan, Yuzi Yan, Hao Yang, Xiaofei Yang, Yi Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Siyu Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Shaojie Zheng, Longguang Zhong, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2026.

[30] ShareGPT Team. Sharegpt. Website, 2023.

[31] The SGLang Team. Deploying deepseek with pd disaggregation and large-scale expert parallelism on 96 h100 gpus. Blog post, May 2025.

[32] Yusong Gao Yun Zhang Tianyu Zhang, Peng Zhang. Together with sglang: Best practices for serving deepseek-r1 on h20-96g. Blog post, 2025.

[33] Xinyi Wan, Penghui Qi, Guangxing Huang, Min Lin, and Jialin Li. Pipeoffload: Improving scalability of pipeline parallelism with memory optimization, 2025.

[34] Jun Wang, Yunxiang Yao, Wenwei Kuang, Runze Mao, Zhenhao Sun, Zhuang Tao, Ziyang Zhang, Dengyu Li, Jiajun Chen, Zhili Wang, Kai Cui, Congzhi Cai, Longwen Lan, and Ken Zhang. Omniinfer: System-wide acceleration techniques for optimizing llm serving throughput and latency, 2025.

[35] Wenxun Wang, Likai Ma, Zongle Huang, Chen Tang, and Yongpan Liu. Rapidmoe: Exploiting cross-asymmetry via adaptive residual offloading for large-scale moe inference. *arXiv preprint arXiv:2610.01265*, 2026.

[36] Yida Wang, Ke Hong, Xiuhong Li, Yuanchao Xu, Wenxun Wang, Guohao Dai, and Yu Wang. Tasp: Topology-aware sequence parallelism, 2025.

[37] Xinjun Yang, Qingda Hu, Junru Li, Feifei Li, Yicong Zhu, Yuqi Zhou, Qiuru Lin, Jian Dai, Yang Kong, Jiayu Zhang, Guoqiang Xu, and Qiang Liu. Beluga: A cxl-based memory architecture for scalable and efficient llm kvcache management, 2025.

[38] Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Huazuo Gao, Jiashi Li, Liyue Zhang, Panpan Huang, Shangyan Zhou, Shirong Ma, Wenfeng Liang, Ying He, Yuqing Wang, Yuxuan Liu, and Y.X. Wei. Insights into deepseek-v3: Scaling challenges and reflections on hardware for ai architectures. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, SIGARCH '25, page 1731–1745. ACM, June 2025.

[39] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62557–62583. Curran Associates, Inc., 2024.

[40] Yuxin Zhou, Zheng Li, Jun Zhang, Jue Wang, Yiping Wang, Zhongle Xie, Ke Chen, and Lidan Shou. Floe: On-the-fly moe inference on memory-constrained gpu, 2025.

[41] Pengfei Zuo, Huimin Lin, Junbo Deng, Nan Zou, Xingkun Yang, Yingyu Diao, Weifeng Gao, Ke Xu, Zhangyu Chen, Shirui Lu, Zhao Qiu, Peiyang Li, Xianyu Chang, Zhengzhong Yu, Fangzheng Miao, Jia Zheng, Ying Li, Yuan Feng, Bei Wang, Zaijian Zong, Mosong Zhou, Wenli Zhou, Houjiang Chen, Xingyu Liao, Yipeng Li, Wenxiao Zhang, Ping Zhu, Yinggang Wang, Chuanjie Xiao, Depeng Liang, Dong Cao, Juncheng Liu, Yongqiang Yang, Xiaolong Bai, Yi Li, Huaguo Xie, Huatao Wu, Zhibin Yu, Lv Chen, Hu Liu, Yujun Ding, Haipei Zhu, Jing Xia, Yi Xiong, Zhou Yu, and Heng Liao. Serving large language models on huawei cloudmatrix384, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Please refer to Abstract and Introduction. The main contribution are explained detailed in Section 3 and Section 4

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Please refer to Section 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Main theoretical results are presented in Section 3 with all variables and proofs provided in Section 3 and Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.

- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Please refer to Section 5.1 and Appendix A.1 for the experimental settings. We also provide an anonymous repository containing code and supplementary materials to facilitate reproduction.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide an anonymous repository containing code and supplementary materials to facilitate reproduction.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).

- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Please refer to Section 5.1 and Appendix A.1 for the experimental settings.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: This paper does not include error bars. However, results are averaged over 3 open-source and complete datasets. We also include evaluation for error perturbation in performance model in Section 5.3 to demonstrate the robustness.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Please refer to Section **??** and Appendix A.1 and A.2.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: There is no societal impact of the work performed.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cite the sglang framework as its GitHub main page suggests.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

A Experimental Details

A.1 Hardware Configurations

All experiments are conducted on a multi-node GPU cluster representative of modern production inference environments. Table 4 summarizes the hardware configuration of each node.

Each node is equipped with 8 NVIDIA H100 GPUs (80GB HBM each), interconnected via NVLink within the node to provide high-bandwidth device-to-device communication. Nodes are connected through 4×400 Gbps InfiniBand links. On the host side, each node contains dual Intel Xeon Platinum 8468V CPUs with 96 physical cores in total, and 1.8 TB of DDR5-4800 host memory split evenly across two NUMA domains. All GPUs are connected to the host via PCIe Gen5 ×16 links, yielding a peak per-GPU H2D bandwidth of up to 64 GB/s.

A.2 Deployment Setup

We deploy all methods using SGLang in disaggregated decode mode (version v0.5.1), following best-practice configurations for large-scale MoE inference [31]. All experiments focus exclusively on the decode stage, where strict latency constraints and dynamic expert activation patterns make load balancing particularly challenging.

Decode Cluster Configuration. Each node runs 8 GPU worker processes, corresponding to one process per GPU. Decode workers across nodes are organized using data parallelism attention to maximize throughput while preserving expert parallelism semantics with parallel size equaling to total number of decode GPUs.

We enable **DeepEP** as the All-to-All backend with the low-latency mode and **DeepGEMM** as MoE operation backend. Two-batch overlap is enabled to overlap attention computation with MoE communication, following the execution timeline described in Section 3. To improve batchsize and throughput of decode cluster while meeting service-level objective, all configurations are deployed with Multi-Token Prediction (MTP) speculative decoding enabled, configured with *speculative-num-draft-tokens=4* with averaged accept length at 3.3 [32].

We generate workload by injecting datasets into a standard Prefill-Decode disaggregattion pipeline. We sample requests from ShareGPT, LongBench, and LiveCodeBench. Injecting them sequentially to Prefill instances, simulating intra- and inter-dataset shifts. Decode instance fetches requests from Prefill instances to meet its target batch size. Following SGLang's stress-test protocal, we initially pause Decode instance to build a Prefill backlog, ensuring Decode queue remains fully saturated.

Table 4: Detailed Hardware Specifications per Computing Node.

Component	**Specification**
CPU	Dual Intel Xeon Platinum 8468V (96 Physical Cores)
GPU	8 × NVIDIA H100 (80GB HBM3)
Host Memory	1.8 TB DDR5-4800
Intra-node Interconnect	NVLink 4.0
Inter-node Interconnect	4 × 400 Gbps InfiniBand
PCIe Interface	PCIe Gen5 × 16
Peak H2D Bandwidth	64×8 GB/s

B Performance Model: Characterization, Calibration and Generalizability

In Section 3.2, we introduced a performance model to guide the vExpert allocator. This appendix details (1) the micro-benchmarking protocols used to calibrate hardware- and model-specific parameters ($\gamma_1, \gamma_2, \Delta, \eta$), (2) a lightweight calibration protocol for new deployment environments and (3) a rigorous evaluation of the model's robustness and generalizability across diverse models and hardware platforms.

B.1 Experimental Setup

Profiling is conducted on an 8-node H100 cluster as detailed in Appendix A.1. We use `torch.profiler` to capture CUDA graph execution timelines and kernel-level latencies. Unless otherwise specified, each H2D transfer in PCIe contention tests (B.3) moves a payload of $7168 \times 2048 \times 3 \times 2$ bytes per rank, corresponding to the weights of two experts in DeepSeek-V3.1.

B.2 Characterizing Load Balancing Gains ($\gamma_1, \gamma_2, \Delta$)

We quantify how the load balance ratio L_i impacts the latency of the critical path operations: All-to-All communication and MoE computation. We artificially skew expert assignments to vary L_i and measure the resulting execution metrics using DeepEP and DeepGEMM backends.

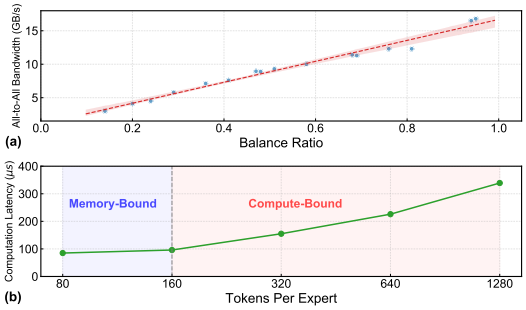


Figure 8: **Impact of load balance ratio on All-to-All communication and MoE computation.** (a) Effective All-to-All bandwidth, measured as total data volume divided by latency. (b) MoE Kernel latency across different Tokens Per Expert (TPE) settings.

Impact on All-to-All Communication (γ_1). Figure 8(a) reports the effective All-to-All bandwidth. As load imbalance increases (lower Balance Ratio), the effective bandwidth drops sharply, even though the aggregate data volume remains constant. This behavior stems from the "straggler effect": the latency is dominated by the rank transmitting the maximum data volume. GPUs hosting "hot" experts saturate their links while lightly loaded GPUs idle early. Consequently, improving the balance ratio directly reduces the maximum per-rank token load, yielding a near-linear reduction in communication latency. This validates our approximation in Equation 3, where $\Delta(L_i, V)$ represents the improvement in load balance.

Impact on MoE Computation (γ_2). Variations in load balance result in different *Tokens Per Expert* (TPE) assignments, which significantly affect MoE GEMM kernel performance. Figure 8(b) illustrates two distinct regimes: **1. Compute-Bound Regime (TPE ≥ 160):** Kernel execution scales proportionally with the maximum token count. Reducing the straggler load yields substantial gains, with $\gamma_2 \approx \gamma_1$. **2. Memory-Bound Regime (TPE < 160):** Execution is dominated by kernel launch overheads and memory access patterns. Reducing the maximum token load provides diminishing returns; latency remains largely flat despite TPE improvements, leading to $\gamma_2 \approx 0$.

Modeling Load Balance Improvement (Δ). The improvement term $\Delta(L_i, V)$ is a complex function of the activation distribution, the initial imbalance L_i, and the redundancy budget V. Deriving a closed-form solution is intractable due to stochastic online request distributions. We instead characterize this relationship through offline profiling on representative datasets (ShareGPT, LongBench). Our observations indicate that for a fixed V, Δ approximates a linear function of the initial balance ratio. We therefore model Δ via a first-order linear approximation:

$$\Delta(L_i, V) = \alpha \cdot L_i + \beta \cdot V + k \tag{6}$$

B.3 Characterizing PCIe Contention (η and BW_{H2D})

This section quantifies the hardware-level interference between Host-to-Device (H2D) expert streaming and cross-node All-to-All (A2A) communication.

Timeline Analysis and Observations. Figure 9 illustrates the execution timeline for both the baseline (A2A only) and concurrent (A2A + H2D) scenarios. In the baseline scenario, the `Combine` and `Dispatch` operators on A2A Stream exhibit stable latencies. In the concurrent scenario, H2D MemCpy operations across multiple streams reveal two key effects:

1. Intra-node H2D Variation: H2D completion times are non-uniform across devices within the same node. This variance stems from shared PCIe tree topologies, where concurrent DMA requests from multiple GPUs compete for the finite arbitration bandwidth of PCIe switches and the CPU's Root Complex.

2. Cross-Operation Interference: Both A2A and H2D transfers on auxiliary streams suffer from increased latency. Although PCIe is theoretically full-duplex, control signaling and aggregate data pressure on internal switch fabrics lead to measurable throughput degradation for both RDMA-based A2A and PCIe-based H2D.

Quantitative Modeling. We define the contention penalty η based on the degradation of A2A latency:

$$\eta = \frac{T^*_{a2a} - T_{a2a}}{T_{h2d}} \tag{7}$$

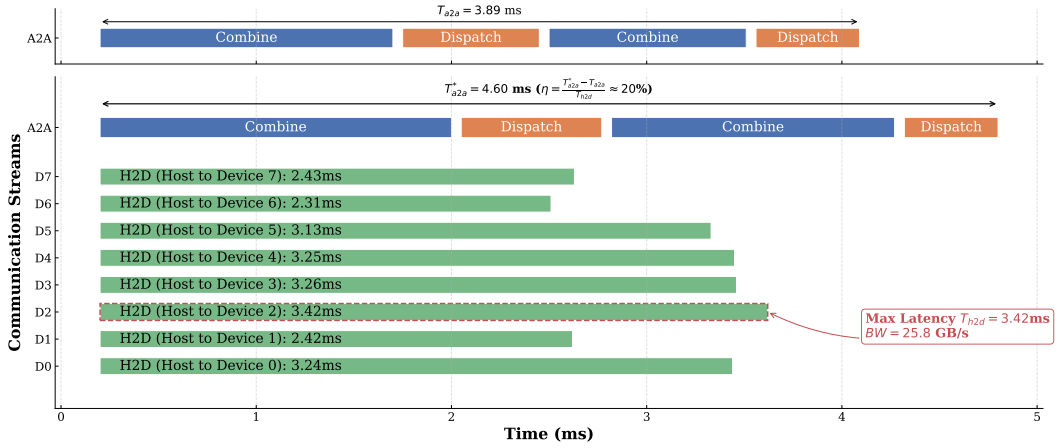


Figure 9: **Micro-benchmark of PCIe contention on H100 cluster (Batch Size 128). Top:** Baseline execution timeline (A2A only) showing stable latencies. **Bottom:** Concurrent execution (A2A + H2D). The overlap introduces resource contention, leading to heterogeneous H2D throughput and increased latency for both operations.

where T_{a2a} is the baseline latency, T^*_{a2a} is the latency under contention, and T_{h2d} is the duration of the concurrent transfer. Our measurements yield $\eta \approx 20\%$ and an effective BW_{H2D} that is degraded compared to the theoretical peak. These values are integrated into the vExpert Allocator (Eq. 5) to predict whether improved load balancing outweighs PCIe contention delays.

B.4 Calibration Protocol for New Environments

While the parameter values presented above are specific, the performance modeling methodology (Section 3.2) is hardware- and model-agnostic. A key advantage of VEXPERT is the minimal recalibration effort required for deployment on new model architectures or hardware platforms. We prescribe the following lightweight, two-path calibration protocol:

1. Model-Specific Path (α, β, k): For a new model architecture, run the target model's gating network on representative validation datasets to capture expert activation distribution. Vary the number of virtual experts V and record the resulting load balance ratio L_i for each layer. A linear regression on these data points extracts the model-specific coefficients α, β, and k. This tracing completes in under 30 minutes.

2. Hardware-Specific Path (BW_{H2D}, η): For a new hardware cluster, only the PCIe contention parameters require re-characterization. Run the concurrent micro-benchmark described in Section B.3 to measure the effective H2D bandwidth (BW_{H2D}) and calculate η using Equation 8. This single-step procedure captures the interference patterns of the deployment cluster's PCIe switches and CPU Root Complexes, completing in under 10 minutes.

New models only require Step 1 (tracing α, β, k), while new hardware only requires Step 2 (measuring $\text{BW}_{\text{H2D}}, \eta$). Once calibrated, parameters are loaded as a static configuration, ensuring the runtime optimization reflects the physical constraints.

B.5 Generalizability Evaluation

A critical concern for performance model is whether its benefits persist under imperfect calibration or when deployed on unseen models and hardware. We systematically evaluate robustness along: cross-model validation, and hardware generalizability. The cross-model generalizability is evaluated at Section 5.3, here we discuss the cross-platform generalizability in detail.

B.5.1 Hardware Generalizability

While our primary evaluation targets NVIDIA H100 clusters, the vExpert performance model is architecturally portable. Modern servers—including NVIDIA H800/H20 and AMD MI300X—universally adopt **PCIe Gen5** interconnects. The consistent physical topology (shared PCIe switch trees, CPU Root Complex arbitration)

dictates that **PCIe contention behavior remains structurally similar**, and parameters BW_{H2D} and η remain stable across platforms with comparable PCIe generations.

For older PCIe Gen4 hardware, we provide a quantitative sensitivity analysis showing that vExpert's benefits are not merely preserved but in fact *amplified*. From Section 3.2, the normalized H2D overhead is:

$$O = \frac{T_{\text{H2D}} \cdot \eta}{T_{\text{layer}}} = \eta \cdot \frac{V_m}{\text{BW}_{\text{H2D}} \cdot T_{\text{layer}}} \tag{8}$$

where V_m is the per-expert transfer size, BW_{H2D} is the PCIe bandwidth, and T_{layer} is the layer computation time. Halving the PCIe bandwidth would, in isolation, double the overhead. However, **PCIe bandwidth and compute capability scale together** across server generations:

- **H100 Server:** PCIe Gen5 (≈64 GB/s) with 1,979 TFLOPS (FP16).
- **A100 Server:** PCIe Gen4 (≈32 GB/s) with 312 TFLOPS (FP16).

Because T_{layer} is approximately inversely proportional to peak FLOPS, the relative overhead on Gen4 compared to Gen5 is:

$$\frac{O_{\text{Gen4}}}{O_{\text{Gen5}}} \approx \frac{\text{BW}_{\text{Gen5}}}{\text{BW}_{\text{Gen4}}} \cdot \frac{T_{\text{layer}}^{(\text{Gen5})}}{T_{\text{layer}}^{(\text{Gen4})}} \approx \frac{64}{32} \cdot \frac{312}{1979} \approx 0.32 \tag{9}$$

Thus, VEXPERT's gains are preserved on lower-bandwidth platforms since slower compute provides more overlap headroom for H2D transfers.

C End-to-end Timeline Comparison

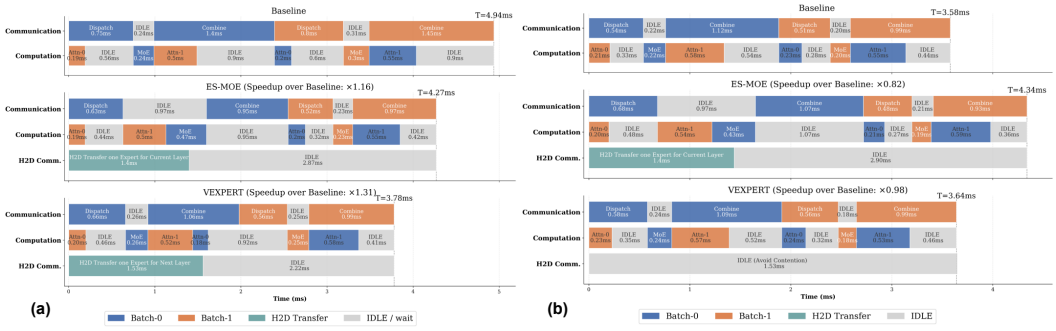


Figure 10: End-to-end timeline comparison on DeepSeek-V3.1 using a 4-node H100 cluster with a batchsize of 256. (a) Timeline of an imbalanced layer with a balance ratio of 0.45. (b) Timeline of a balanced layer with a balance ratio of 0.68.

Figure 10 presents a detailed per-layer timeline comparison among **vExpert**, **Baseline** (B2, as described in Section 5.1), and **ES-MoE** [13] (offloading-based load balancing, labeled as "Offload" in Section 5.1). The traces are captured on a 4-node H100 cluster running DeepSeek-V3.1 at batch size 256. We select two representative layers to illustrate the spatial dynamism of load distribution.

Limitations of Baseline (B2). Due to the inherent **spatial dynamism** (layer-to-layer variation) and **temporal dynamism** (iteration-toiteration fluctuation) of expert activation distributions, the Baseline struggles to achieve global load balance. While static expert placement may yield acceptable balance for some layers, it inevitably leaves others severely skewed.

Limitations of Offloading (ES-MoE). ES-MoE leverages host memory to re-balance tokens across devices via offloading partial experts. However, this approach suffers from two fundamental shortcomings. **First**, it ignores the **PCIe contention** introduced by H2D transfers. Streaming experts via PCIe shares the interconnect fabric with RDMA-based A2A communication. **Second**, ES-MoE applies a *uniform* offloading policy across *all* layers, unable to exploit inter-layer and inter-inter heterogeneity in load distribution. Even for naturally balanced layers (Figure 10 (b)), it incurs **constant H2D transfers** that introduce unnecessary synchronization overhead and contention penalties. Consequently:

For Imbalanced Layer: VEXPERT utilizes H2D to prevent load imbalance that blocks All-to-All, yielding to a 30% latency reduction.

For Balanced Layer: VEXPERT skips unnecessary H2D to avoid contention while offloading requires constant transfers.

D Analysis on vExpert Group

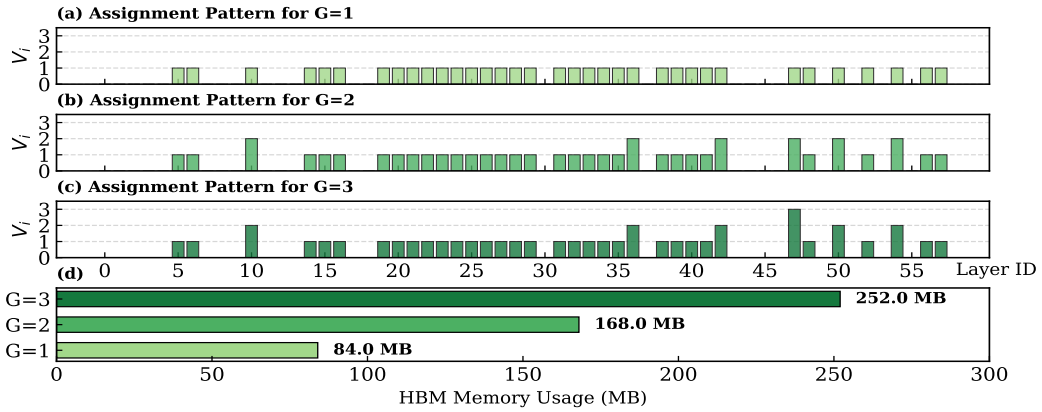


Figure 11: **Analysis of vExpert group size G on assignment pattern and memory overhead.** (a)-(c) illustrate the layer-wise assignment of virtual experts (V_i) across different group sizes. (d) The GPU HBM overhead for required slot buffers scales linearly with G.

In this section, we analyze the impact of group size G. The group-wise slot strategy is designed to relax the strict per-layer H2D bandwidth constraint, allowing the system to redistribute vExpert slots from "easy-to-balance" layers to "bottleneck" layers.

Trade-off between Flexibility and Memory. The total GPU memory overhead for vExpert slots is defined as $M = G \cdot V_{\max} \cdot W_{\text{expert}}$. While a larger G expands the search space for the optimization problem (Eq. 5), it also increases the HBM footprint. We evaluate $G \in \{1, 2, 3\}$ as shown in Figure 11.

$G = 1$ (Rigid Constraint): Each layer is strictly limited by its own TPOT. If a layer is highly imbalanced but its TPOT is short, it cannot acquire more than $V_{\max}$ experts, even if adjacent layers have significant idle PCIe bandwidth.

$G = 2$ (Elastic Allocation): By grouping two layers, the allocator can assign $V_i = 2$ to an imbalanced layer by setting $V_{i+1} = 0$ for its balanced neighbor. This slot donation mechanism improves the load-balancing effectiveness for critical layers.

$G = 3$ (Diminishing Returns): While $G = 3$ further relaxes the constraint, our experiments show that the assignment pattern remains largely similar to $G = 2$. Most imbalanced layers are effectively covered by 2 experts, and the additional memory cost for $G = 3$ does not translate into higher throughput.

As shown in our evaluation, the layer-wise assignment for $G = 2$ differs from $G = 1$, specifically targeting imbalanced layers. In contrast, the assignment for $G = 3$ provides marginal gains over $G = 2$. Beyond hardware resource consumption, we consider the system orchestration overhead as a decisive factor:

Implementation Complexity: A larger G necessitates a more sophisticated scheduler to manage multi-layer dependencies and a more complex memory manager to track asynchronous H2D transfers across an extended window. Such complexity risks bloating the runtime logic and increasing the host-side CPU overhead. Maintaining a small G ensures that the vExpert allocator remains lightweight and agile, allowing it to complete reconfigurations within the strict latency constraints of real-time decoding. Given that $G = 2$ achieves the optimal balance between balancing flexibility and implementation simplicity—while only incurring a modest HBM footprint of $\sim$168 MB (a negligible fraction of the 80GB HBM on an H100 GPU)—we select $G = 2$ as the default configuration for deployment.

E Physical-to-virtual Mapping Design

This appendix details the proposed physical-to-virtual mapping strategy. The rebalancing procedure is summarized in Algorithm 1 and consists of three stages: group-wise vExpert number assignment, vExpert selection, and greedy physical rebalancing.

Algorithm 1 Physical-to-Virtual Rebalancing

Require: Expert load statistics W; group size G; per-layer vExpert budget $V_{\max}$; number of GPUs N_{gpu}; Inference constant C;
Ensure: Physical-to-virtual map Φ;

// Step 1: Group-wise vExpert number assignment
for each layer group $\mathcal{G}$ of size G **do**
  Compute group budget $V^*_{\max} \leftarrow G \cdot V_{\max}$
  for each layer $L_i \in \mathcal{G}$ **do**
    Compute GPU-wise load $\mathbf{W}_i$
    $\mathcal{L}_i \leftarrow \text{mean}(\mathbf{W}_i)/\max(\mathbf{W}_i)$
  end for
  $\{V_i\}_{L_i \in \mathcal{G}} \leftarrow$ Algorithm 2$(\{\mathbf{W}_i, \mathcal{L}_i\}_{L_i \in \mathcal{G}}, V^*_{\max}, C)$ // vExpert Number Assignment
end for
// Step 2: vExpert Candidate selection
for each layer L_i **do**
  Rank physical experts by activation frequency
  Select top $V_i \times N_{\text{gpu}}$ physical experts
  Generate vExpert-to-physical Candidate Ψ
end for
// Step 3: Greedy physical rebalancing
for each layer L_i **do**
  Initialize GPU loads with base experts
  for each virtual expert v in descending load order of Ψ **do**
    Select GPU g with minimum current load
    Assign expert v with the largest load to GPU g
    Update GPU load and capacity
    Update mappings Φ
  end for
end for
return Φ

E.1 Step 1: Group-wise vExpert Number Assignment

Layers are partitioned into groups of size G. Each layer is initially allocated a per-layer vExpert budget $V_{\max}$ which is determined by TPOT SLO. Within a group $\mathcal{G}$, the total vExpert budget is therefore $V^*_{\max} = G \cdot V_{\max}$.

For each layer $L_i \in \mathcal{G}$, we compute the layer balance ratio $\mathcal{L}_i$ using expert load statistics $\mathbf{W}_i$ from previous iterations:

$$\mathcal{L}_i = \frac{\mathbb{E}[\mathbf{W}_i]}{\max(\mathbf{W}_i)}. \tag{10}$$

Well-balanced layers are filtered to assign zero slot, allowing their unused budget to be redistributed to more imbalanced layers within the same group. The final vExpert count $\{V_i\}_{L_i \in \mathcal{G}}$ is determined by Algorithm 2, which minimizes an estimated per-layer iteration cost based on Eq. 5. The optimization is parameterized by an inference constant C, which captures runtime-dependent factors such as batch size and hardware communication characteristics.

E.2 Step 2: vExpert Candidate Selection

Given the vExpert count V_i for each layer, we select physical experts to replicate. Physical experts are ranked by their historical activation frequency. For layer L_i, the top $V_i \times N_{\text{gpu}}$ experts are selected and replicated to form virtual experts. This produces a vExpert-to-physical mapping candidates that is reused in the subsequent rebalancing stage.

E.3 Step 3: Greedy Physical Rebalancing

Virtual experts are assigned to GPUs using a greedy bin-packing strategy. Each GPU is treated as a bin whose capacity equals the number of base experts plus the allocated vExperts. For each layer, virtual experts are processed in descending order of their expected load. Each virtual expert is assigned to the GPU with the smallest current load that still has available capacity. After each assignment, GPU load statistics and expert

Algorithm 2 Group-wise vExpert Number Assignment

Require: Layer-wise GPU load $\{\mathbf{W}_i\}$; layer balanceness $\{\mathcal{L}_i\}$; group vExpert budget $V^*_{\max}$; inference constant C
Ensure: vExpert count per layer $\{V_i\}$

Initialize $V_i^{\max} \leftarrow V^*_{\max}/|\mathcal{G}|$ for all $L_i \in \mathcal{G}$
// Filter well-balanced layers
for each layer $L_i \in \mathcal{G}$ **do**
  if $\mathcal{L}_i > \tau$ **then**
    $V_i^{\max} \leftarrow 0$
  end if
end for
// Redistribute slots within the group
for each layer $L_i \in \mathcal{G}$ **do**
  Adjust $\{V_i\}$ to satisfy $\sum_{L_i \in \mathcal{G}} V_i = V^*_{\max}$
end for
// Optimization-based redistribution
Initialize $V_i \leftarrow V^*_{\max}/|\mathcal{G}|$ for all $L_i \in \mathcal{G}$
for each remaining layer $L_i \in \mathcal{G}$ **do**
  $t_{\min} \leftarrow +\infty$
  $V_i^\star \leftarrow 0$
  for $v = 0$ to $V_i^{\max}$ **do**
    Estimate communication time $t_{\text{comm}}(v; C)$
    Estimate MoE compute time $t_{\text{moe}}(v; C)$
    Estimate H2D transfer overhead $t_{\text{H2D}}(v, C)$
    Compute objective $t(v)$ using Eq. (2)
    if $t_i(v) < t_{\min}$ **then**
      $t_{\min} \leftarrow t_i(v)$
      $V_i^\star \leftarrow v$
    end if
  end for
  $V_i \leftarrow V_i^\star$
end for
return $\{V_i\}$

mappings are updated accordingly. This greedy strategy is efficient and empirically achieves near-optimal load balance.

F Analysis of Allocator Overhead

In this section, we provide a detailed breakdown of the latency overhead introduced by the vExpert Allocator. To maintain high throughput in real-time decoding, it is imperative that the reconfiguration logic does not become a bottleneck. We evaluate the overhead on our cluster consisting of 8 computing nodes (64 GPUs total).

The allocation process occurs every k iterations and consists of two primary stages:

1. Global Statistics Aggregation (4 ms): Each rank maintains a local `logical_count` tensor, recording the activation frequency of experts. Before allocation start, system performs `All-Reduce` operation across all GPUs to synchronize these statistics. Despite the cluster scale, this collective communication is highly optimized via InfiniBand, completing in approximately **4 ms**.

2. Physical-to-Virtual Mapping (7 ms): Once the global activation distribution is available, each rank executes the physical-to-virtual mapping (as detailed in Appendix E). This computational step consumes only **7 ms**.

The total latency for a single reconfiguration event is:

$$T_{\text{alloc}} = T_{\text{all_reduce}} + T_{\text{mapping}} = 11\text{ms} \tag{11}$$

In our default configuration, we set the allocation frequency at $k = 10$. This allows the system to remain responsive to load drifts while significantly diluting the overhead at $\frac{T_{\text{alloc}}}{k} = 1.1$ms per iter. This confirms that vExpert is a lightweight and scalable solution suitable for high-frequency online adaptation.