Expert-as-a-Service: Towards Efficient, Scalable, and Robust Large-scale MoE Serving

Ziming Liu[*]
National University of Singapore
Shanghai Qiji Zhifeng Co., Ltd.
liuziming@comp.nus.edu.sg

Boyu Tian
Shanghai Qiji Zhifeng Co., Ltd.
tianboyu@qijizhifeng.com

Guoteng Wang
Shanghai Qiji Zhifeng Co., Ltd.
wangguoteng@qijizhifeng.com

Zhen Jiang
Shanghai Qiji Zhifeng Co., Ltd.
jiangzhen@qijizhifeng.com

Peng Sun
Shanghai Qiji Zhifeng Co., Ltd.
sunpeng@qijizhifeng.com

Zhenhua Han
Shanghai Qiji Zhifeng Co., Ltd.
hanzhenhua@qijizhifeng.com

Tian Tang
Shanghai Qiji Zhifeng Co., Ltd.
tangtian@qijizhifeng.com

Xiaohe Hu
Infrawaves
huxiaohe@infrawaves.com

Yanmin Jia
Infrawaves
jiayanmin@infrawaves.com

Yan Zhang
Infrawaves
zhangyan@infrawaves.com

He Liu
Infrawaves
liuhe@infrawaves.com

Mingjun Zhang
Infrawaves
Mingjun.Zhang02@gmail.com

Yiqi Zhang
National University of Singapore
yiqi.zhang@u.nus.edu

Qiaoling Chen
Nanyang Technology University
qiaoling.chen@ntu.edu.sg

Shenggan Cheng
National University of Singapore
shenggan@comp.nus.edu.sg

Mingyu Gao
Tsinghua University
gaomy@tsinghua.edu.cn

Yang You[†]
National University of Singapore
youy@comp.nus.edu.sg

Siyuan Feng[†]
Shanghai Innovation Institute
Shanghai Qiji Zhifeng Co., Ltd.
syfeng@sii.edu.cn

Abstract

Mixture-of-Experts (MoE) models challenge serving infrastructures with dynamic, sparse expert utilization, causing instability on conventional systems designed for dense architectures. We propose EAAS, a novel serving system to enable efficient, scalable, and robust MoE deployment. Our system disaggregates MoE modules into independent, stateless services. This design enables fine-grained resource scaling and provides inherent fault tolerance by decoupling compute units. The architecture is powered by a high-performance, CPU-free peer-to-peer communication library that ensures minimal overhead and high throughput. Experiments confirm EAAS's scalability and efficiency, achieving performance comparable to monolithic systems while providing robust fault tolerance and strong scalability. EAAS incurs less than a 2% throughput reduction under simulated hardware failures that would otherwise halt monolithic architectures. It further saves up to 37.5% of computing resources through dynamic fine-grained adaptation to serving traffic, demonstrating strong resilience for large-scale MoE deployment in production.

1 Introduction

The rapid advancement of Large Language Models (LLMs) has revolutionized natural language processing, enabling unprecedented capabilities but simultaneously introducing significant computational challenges for deployment. Serving these increasingly large and complex models efficiently is paramount for their practical application. Mixture-of-Experts (MoE) architectures [4, 9, 13, 14, 16, 33] have emerged as a promising direction, mitigating computational costs by leveraging sparsity. Unlike dense models where all parameters participate in every computation, MoE models activate only a subset of specialized sub-models ("experts") per input token, potentially reducing computational load significantly while maintaining high performance. Furthermore, recent fine-grained-expert MoE models, such as the DeepSeek series [5, 13, 14], push sparsity further by employing numerous smaller experts, enabling scaling to near-trillions of parameters without a proportional increase in inference cost.

Despite the architectural advantages, efficiently serving MoE models presents unique challenges distinct from those

of dense LLMs. While several frameworks (e.g., vLLM [10], SGLang [35], and TensorRT-LLM [21]) excel at serving dense models, they often struggle with the dynamic, sparse nature of MoE computation. First, existing systems can exhibit poor elasticity, requiring large, monolithic deployment units (e.g., DeepSeek-V3/R1 requires 320 GPUs as a basic scaling unit for high efficiency [5, 14]), which suffer from coarse scaling granularities. Second, mainstream approaches rely heavily on large-scale collective communication primitives (like All-to-All for expert parallelism) established within static process groups at initialization. This tight coupling makes the system brittle: a single device failure can necessitate restarting the entire group. Third, the inherent dynamism of MoE leads to workload imbalance. The distribution of activated experts can vary drastically depending on the input data and downstream task, yet existing systems often employ static sharding strategies that cannot adapt, resulting in inefficient resource utilization where some expert-hosting devices are overloaded while others remain idle. While recent attempts have explored [32, 36] disaggregating the attention and MoE layers, they often fail to resolve these fundamental issues, retaining static group structures and tightly coupled control logic that limits flexibility and robustness.

To address these limitations, we propose EaaS, a scalable, robust, and efficient serving system designed for MoE LLMs. The core principle of EaaS is to re-architect the system around the concept of disaggregating experts into independent services. We decouple the MoE layers from the rest of the model, such as the attention layers, treating the pool of experts as dynamically accessible and independent services. This fundamental shift from monolithic deployment to a service-oriented architecture offers several key advantages:

Fine-grained Elasticity: This service-oriented approach eliminates the rigid structure of large serving units. EaaS allows expert capacity to be scaled almost linearly and independently from the attention computation components. For instance, serving a model like DeepSeek-V3 [14] can begin with a practical base unit (e.g., 16x 80GB GPUs) and scale incrementally, potentially one GPU at a time, to precisely match demand. Furthermore, EaaS removes the need for static communication group establishment between GPUs, enabling more precise scaling according to the current workload.

Improved Robustness: By replacing collective communication with peer-to-peer (P2P) interactions between attention clients and expert servers, EaaS removes the dependency on fragile, static communication groups for expert parallelism. If an expert server fails, clients can transparently failover to a replica or an alternative instance after a timeout with minimal disruption. Recovery is also simplified; since experts operate as services, a new server only needs to register its availability to be integrated into the system, avoiding the costly rebuild of a global communication group.
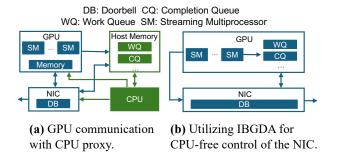

(a) GPU communication with CPU proxy. **(b)** Utilizing IBGDA for CPU-free control of the NIC.

Figure 1. EaaS leverages InfiniBand GPUDirect Async (IBGDA) to achieve low communication latency and minimize kernel launch overhead through full CUDA graph capture, enabled by CPU-free control.

Dynamic Load Balancing: The independence of each expert server enables real-time load balancing. If specific experts are frequently requested by more tokens, they can be duplicated to balance the computation. Furthermore, these new expert instances can be added to the serving pool without interrupting the overall service, allowing the system to adjust to shifting workloads and prevent performance bottlenecks.

Additionally, efficiently implementing this disaggregated model requires a flexible, high-performance communication substrate for dynamic, asymmetric P2P interactions. Existing libraries often fall short: NCCL [19] lacks suitable single-sided P2P operations, while RDMA-based libraries like NVSHEMEM [23] impose restrictive symmetric buffer requirements. While recent works such as StepMesh [32] and Megascale-Infer [36] leverage CPU-controlled communication, introducing extra overhead and preventing end-to-end CUDA graph capture. To overcome these limitations, we developed a novel asymmetric asynchronous P2P communication library using low-level IBGDA (InfiniBand GPUDirect Async) primitives. Our library offers flexible buffer management, native support for efficient single-sided operations, minimal overhead through CPU-free and group-free network operation, perfectly aligning with the requirements of EaaS.

In summary, our contributions are:

1. We propose EaaS, a novel serving system architecture for fine-grained MoE models based on disaggregating experts into independent services. This design enhances elasticity, robustness, and load balancing compared to monolithic approaches.
2. We design and implement a high-performance, asymmetric, asynchronous CPU-free P2P communication library based on IBGDA, tailored for dynamic client-server interactions in GPU clusters, overcoming limitations of existing libraries for this use case.

3. Through extensive experiments, we show that EAAS achieves performance comparable to state-of-the-art monolithic systems while providing superior fault tolerance and scalability. EAAS shows less than 2% decoding throughput drop during hardware failures that would halt conventional architectures and allows for incremental, cost-effective scaling not possible with rigid, group-based designs, saving up to 37.5% computing resources.

2 Background and Motivation

This section provides the necessary background on Mixture-of-Experts models and current serving systems. We then analyze the fundamental limitations of existing monolithic designs, which motivate our new approach.

2.1 Mixture-of-Experts: Scaling with Sparsity

Mixture-of-Experts (MoE) has become a dominant architecture for scaling Large Language Models efficiently. The core idea is to replace dense feed-forward network (FFN) layers with a larger set of smaller, specialized "expert" sub-networks. For each input token, a lightweight gating network dynamically selects a small subset of these experts to perform the computation. This conditional execution allows models to grow to trillions of parameters in total capacity without a proportional increase in the computational cost for any single inference pass.

Recent architectural trends have moved from models with a few large experts (e.g., Mixtral-8x7B [9]) toward fine-grained MoE, featuring thousands of much smaller experts (e.g., DeepSeek-V3 [14], Kimi-K2 [28], Qwen3 [33]). As shown in Table 1, this approach enables enormous model scale while keeping the number of activated parameters manageable. However, this fine-grained sparsity introduces two significant challenges for serving systems:

- **Low Computational Intensity**: During batched decoding, tokens are scattered across a vast number of experts. This means each expert processes only a small fraction of the tokens in a batch, leading to poor hardware utilization unless extremely large, often impractical, batch sizes are used.
- **Dynamic Workload Imbalance**: Token-level expert assignments are highly dynamic and input-dependent. As illustrated in Figure 2, the distribution of expert activations can vary dramatically across different tasks and datasets. This data-dependent skew creates a difficult load-balancing problem that static resource allocation schemes cannot solve.

2.2 The Monolithic MoE Serving Architecture

To handle the immense size of modern MoE models (Table 1), distributed serving is a necessity. The predominant approach, adopted by systems like vLLM [10], SGLang [35],

Model	Total Param. (B)	Active Param. (B)	Min. Unit (80GB GPUs)
Qwen3-235B-A22B	235	22	8
DeepSeek V3	671	37	16
Kimi-K2	1000	32	32
Llama 4 Behemoth	2000	288	64

Table 1. While MoE models activate only a fraction of their total parameters, housing the entire model still demands substantial GPU resources, necessitating distributed serving.
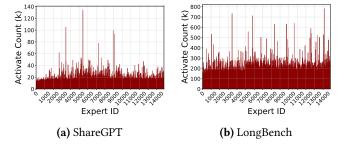

(a) ShareGPT **(b)** LongBench

Figure 2. The distribution of expert activations varies significantly across different downstream tasks, posing a dynamic load balancing challenge.

and TensorRT-LLM [21], can be characterized as a monolithic architecture. In this design, the entire model, including both attention and expert layers, is deployed as a single, tightly-coupled distributed application.

At the heart of this architecture lies *Expert Parallelism (EP)*, a technique where experts are sharded across a group of GPUs. To route tokens from the attention layers to their designated experts, these systems rely heavily on large-scale *collective communication* primitives, most notably `All-to-All`. During an MoE layer computation, each GPU sends the tokens destined for a specific expert to the GPU hosting it. This requires a collective data exchange where every GPU in the group communicates with every other GPU.

This reliance on collective communication forces the creation of a *static process group* at initialization. All participating GPUs must be known beforehand, and the communication patterns are pre-determined. While effective for static workloads, this monolithic, group-based design creates a rigid structure that is ill-suited for the dynamic and demanding nature of production LLM serving. This rigidity gives rise to three fundamental limitations:

- **Coarse-Grained Elasticity.** The static process group is the indivisible unit of deployment and scaling. To add capacity, an entire new group of identical size and configuration must be provisioned. It's impossible to add just a few GPUs to handle a marginal increase in load. This leads to inefficient resource allocation, as systems must be over-provisioned for peak demand. The 320-GPU "basic
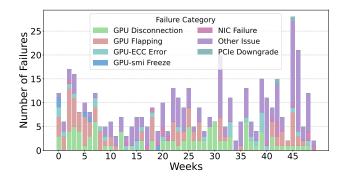
Figure 3. Weekly failure frequencies and categories observed in our computing cluster, consisting of 4096 NVIDIA Hopper GPUs with each node connected via 8×400 Gbps RoCE network. An average of 8.8 failures per week indicates that robust fault tolerance and rapid recovery are essential in such large-scale systems.

scaling unit" recommended for DeepSeek-V3 [14] is a clear example of this coarse granularity.

- **Brittleness and Low Fault Tolerance.** Collective communication is fragile. The failure of a single GPU or network link within the static group will cause the All-to-All operation to hang or fail, bringing the entire serving instance to a halt. Recovery requires restarting the entire group, a slow and disruptive process that reduces service availability.
- **Static and Inefficient Load Balancing.** The mapping of experts to GPUs is fixed when the service is launched. This static assignment cannot adapt to dynamic workload patterns, such as those shown in Figure 2. If certain experts become "hot" (frequently activated), the GPUs hosting them become performance bottlenecks, while GPUs with "cold" experts sit idle. The monolithic design lacks the flexibility to mitigate these imbalances by, for instance, dynamically replicating popular experts on underutilized hardware.

Recent works like MegaScale-Infer [36] and StepMesh [32] have explored disaggregating the attention and MoE layers into separate tiers. However, they still rely on static, group-based collective communication between these tiers. They essentially create two coupled monolithic groups, thereby inheriting the same fundamental limitations of poor elasticity, brittleness, and static load management.

These challenges highlight a fundamental architectural mismatch between the static, rigid nature of monolithic serving systems and the dynamic, sparse computational patterns of fine-grained MoE models. This motivates our departure from the monolithic paradigm toward a more flexible, robust, and service-oriented design.

2.3 Challenges of Efficiency and Scalability

While the monolithic architecture is functional, its rigidity creates severe practical challenges when deployed in production environments, particularly for large-scale Model-as-a-Service (MaaS) providers. These challenges manifest as operational inefficiencies, poor reliability, and performance bottlenecks.

Inefficient Provisioning and High Operational Costs. The monolithic design, with its coarse-grained and indivisible scaling units, imposes a high "buy-in" cost and forces inefficient resource provisioning. MaaS providers must allocate resources for peak demand, meaning a large cluster of GPUs (e.g., the 320-GPU unit for DeepSeek-V3) remains committed even during periods of low traffic. This leads to significant resource stranding and drives up operational costs, as the provider pays for idle, tightly-coupled hardware. Furthermore, this inflexibility prevents providers from dynamically adjusting the ratio of attention-to-expert compute resources to match shifting workload characteristics, leading to a one-size-fits-all allocation that is rarely optimal.

Poor Fault Tolerance and High Recovery Overhead. The tight coupling of resources within a static process group makes the entire serving instance extremely brittle. Collective communication primitives like `All-to-All` require every participant to be healthy; the failure of a single GPU or network link can cause the collective to hang indefinitely, triggering a cascading failure that brings down the entire service. As shown in Figure 3, hardware failures are a common occurrence in large-scale clusters. In a monolithic system, recovery from such an event is a costly, disruptive process, requiring the restart of the entire multi-hundred-GPU deployment. This leads to significant downtime, dropped user requests, and a severe impact on service-level objectives (SLOs).

Static Load Balancing and Performance Bottlenecks. The assignment of experts to specific GPUs is fixed at deployment time. This static allocation is fundamentally incapable of adapting to the dynamic, input-dependent nature of expert activation patterns (as illustrated in Figure 2). Consequently, when certain experts become "hot spots" due to the workload, the GPUs hosting them are overwhelmed, creating bottlenecks. Simultaneously, GPUs hosting "cold" or less-frequently used experts remain underutilized. The overall throughput is thus throttled by the single most overloaded device, not the aggregate capacity. This static imbalance prevents the system from achieving its potential performance and leads to inefficient use of expensive GPU resources.

2.4 The Architectural Pivot: Exploiting the Stateless Nature of Expert

The severe challenges of elasticity, robustness, and load balancing detailed above are not independent issues; they are
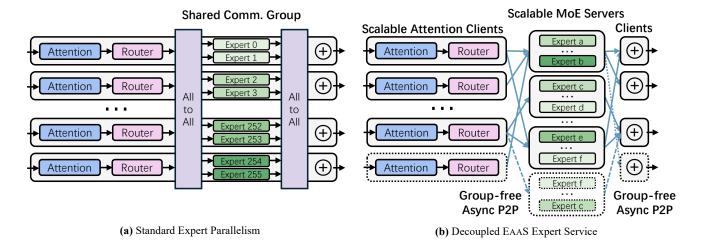
Figure 4. Overview of EAAS approach. Existing Expert Parallelism requires sharing communication groups with other collective parallelisms. EAAS disaggregates the MoE layers and other dense components, enabling flexible and scalable deployment of attention and MoE layer. Our approach extracts the experts in MoE layers as independent services. Once the attention clients finish the computation, the router will send the token to the corresponding expert server and gather the result back via async peer-to-peer connection, and then applies the weighted sum with router scores.

symptoms of a single, fundamental architectural flaw in the monolithic design: the tight coupling of stateful and stateless computation. An LLM serving process has two distinct components:

- **Stateful Attention Layers:** These layers maintain the Key-Value (KV) cache, which stores the contextual state for each individual generation sequence. This component is inherently stateful and session-specific.
- **Stateless MoE Layers:** The expert networks within MoE layers perform a pure, idempotent function. They take a token's hidden state as input and produce an output, without retaining any memory or state from previous tokens in the same sequence.

The monolithic architecture shackles the stateless, flexible MoE layers to the stateful, rigid attention components. This forces the entire system to be managed as a single, stateful, indivisible unit, thereby inheriting the worst properties of both. This core insight—the statelessness of experts—presents a clear architectural pivot. By decoupling these components, we can design a system that treats experts as the independent, stateless services they truly are. This disaggregation directly unlocks solutions to the previously identified challenges:

Independent, Fine-Grained Scaling. Decoupling allows the pool of expert servers to scale independently from the attention-processing frontends. Instead of provisioning rigid, monolithic 320-GPU blocks, a provider can add or remove expert-hosting GPUs one at a time to precisely match the computational demand for experts, eliminating the massive resource stranding and high operational costs associated

with over-provisioning. The ratio of attention-to-expert compute can be dynamically tuned to suit the workload, rather than being fixed by a rigid hardware topology.

Robustness through Replication. The stateless nature of experts makes them perfectly suited for replication. They become interchangeable, fungible resources. If an expert server fails, requests can be transparently rerouted to a replica without disrupting the stateful KV cache on the attention GPU. This replaces the brittle, all-or-nothing failure model of collective communication with a robust, gracefully degrading service model. Recovery is as simple as a new expert server registering its availability, eliminating the need for a costly, service-halting restart of the entire group.

Dynamic Load Balancing via Service Instantiation. A service-oriented design breaks the static mapping of experts to specific GPUs. When the system detects that certain experts have become "hot spots" (as shown in Figure 2), it can dynamically instantiate new copies of those specific experts on underutilized GPUs in the cluster. These new instances can immediately begin serving requests, distributing the load and alleviating bottlenecks in real-time. This transforms load management from a static, pre-deployment guess into a dynamic, adaptive system optimization.

This architectural pivot from a monolithic, tightly coupled system to a disaggregated, service-oriented one forms the cornerstone of our approach. Based on these principles, we designed EAAS, a system for large-scale MoE model serving that is inherently elastic, robust, and efficient. We will detail its design in the following section.

3 System Design

This section details the architecture of EAAS, a serving system designed for large-scale MoE models. It presents the system's disaggregated architecture, the shared communication buffer design, and the asynchronous expert server operation, which is critical for achieving scalability and robustness.

3.1 Architectural Overview

As depicted in Figure 4, the EAAS framework represents a paradigm shift for MoE model inference. Unlike conventional expert parallelism (Figure 4 (a)), which uses a monolithic deployment that tightly couples attention and expert layers within a static group, EAAS employs a disaggregated, service-oriented architecture (Figure 4 (b)). This design is rigid, vulnerable to single-point failures, and scales suboptimally.

The EAAS architecture decouples the stateless MoE layers from stateful components like the attention layers, creating two independently scalable entities: **1) Attention Clients:** These units handle dense computations like the attention mechanism. When an MoE layer is reached, the client's router dispatches activations to the appropriate expert servers via asynchronous peer-to-peer (P2P) communication. **2) Scalable Expert Servers:** Each server hosts a subset of experts and operates as an independent, stateless service focused solely on executing expert computations.

This architectural decoupling obviates the need for a shared, static communication group, which facilitates fine-grained elasticity and enhanced fault tolerance. Consequently, both clients and servers can be scaled independently to meet dynamic workloads, ensuring that an individual component failure does not cause a system-wide outage.

3.2 Shared Communication Buffer Design

The interaction between clients and expert servers is orchestrated through a shared communication buffer designed for efficient, asynchronous data exchange. As shown in Figure 5, each client is allocated a dedicated buffer slot on the server, which is composed of a state flag, a header, and a data payload.

Buffer State: A single-byte flag denotes the buffer's status to mediate client-server interaction. The states are "0" (Empty) for client writing, "1" (Client Write Done) for server processing, 2 (Server Computation Done) for client reading, and "3" (Offline) to mark an inactive client.

Header: This section contains metadata for request processing, including the `layer_id` and `batch_size`. It can also include data size for memory management.

Data Payload: This section holds the core data for computation, including the token activation (hidden states), the `expert_id` mapping for each token, and the `router_score` for weighting outputs.

This structured buffer design creates a clear separation of responsibilities, where the client manages writing requests and reading results, while the server only reads requests and writes results. The state flag ensures proper synchronization without direct communication.

3.3 Asynchronous Server Operation

A fundamental principle of EAAS is the complete operational independence and asynchronicity of the expert servers. It is critical to note that the server does not initiate any synchronization or direct communication with the clients. Instead, it functions within a continuous, stateless loop, responding exclusively to the state of the shared communication buffers.

The server's operational workflow proceeds through a sequence of distinct steps. First, the server continuously and asynchronously polls the state flags of all client buffers allocated to it. Upon identifying all buffers in the "1" (Client Write Done) state, it aggregates these ready-to-process requests from multiple clients to form a dynamic batch. Subsequently, the tokens from this aggregated batch are reorganized and processed with high efficiency, utilizing optimized kernels such as grouped GEMM to achieve maximum hardware utilization. Following the completion of the computation, the server writes the results—specifically, the weighted sum of expert outputs—back into the data payload section of the corresponding client buffers. As a final step, the server updates the state flag of each processed buffer to "2" (Server Computation Done), thereby signaling to the respective clients that the results are available for retrieval.

This fully independent mode of operation is integral to the system's robustness and scalability. The stateless nature of the server and its independence from a static client group render it resilient to client failures. New servers can be dynamically integrated into the pool to manage increased loads, and clients can perform transparent failover to replica servers without service interruption. This design effectively transforms the expert modules into a scalable, robust, and on-demand computational service.

3.4 Fault Tolerance Mechanisms

The disaggregated and asynchronous nature of the EAAS architecture, supported by a central monitor, provides inherent fault tolerance and ensures high service availability. As illustrated in Figure 6, the system is designed to handle failures of both clients and servers gracefully using a heartbeat-based health tracking mechanism.

Client Failure: In the event that an attention client goes offline, the monitor detects the loss of the client's heartbeat signal. It subsequently notifies all relevant expert servers of the failure. Upon receiving this notification, the servers release the communication buffer allocated to the offline client, preventing resource locks and ensuring that the failure remains localized. This event only affects the inference requests being processed by that specific client; all other
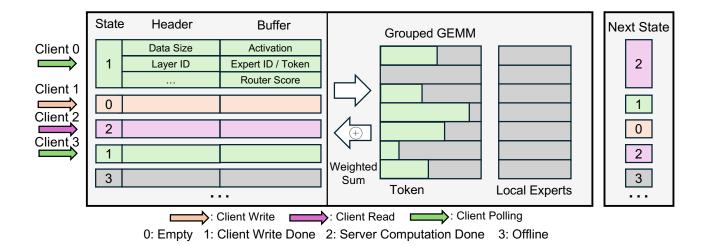
Figure 5. Dynamic batching for EAAS expert servers. The expert server continuously monitors and aggregates incoming requests from attention clients whose buffer state is "1" (indicating the client has finished writing and the data is ready for server-side computation). To ensure high GPU utilization, the server waits to aggregate a sufficient number of client requests before processing them as a batch. Tokens are then reorganized and processed using grouped GEMM operations. Upon completion of computation, the server writes the results back to the corresponding locations in the shared communication buffers and updates the buffer state to "2" (signaling that computation is finished and the output is ready for the client to read). Both read and write operations are performed as one-sided RDMA operations initiated by the client; the server does not initiate RDMA operations, allowing it to remain stateless and focused solely on computation.
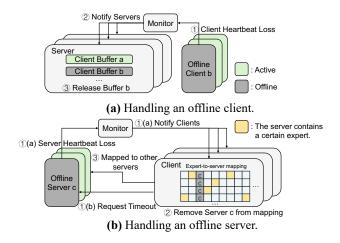

(a) Handling an offline client.

(b) Handling an offline server.

Figure 6. EAAS utilizes a monitor to listen to the heartbeat of all running workers and send notifications for servers to release buffer or for clients to modify the expert-to-server mapping mask when detecting an offline client/server. A client can detect an offline server either through the monitor (①(a)) or a timeout request (①(b)).

clients and servers continue their operations without interruption, preserving overall system stability.

Server Failure: The system's resilience to server failure is achieved through the duplication of stateless expert services and is handled via two detection pathways. First, the central monitor can detect a server's heartbeat loss and proactively

notify all clients. Second, a client can independently infer a server failure if a request's polling period exceeds a predefined timeout threshold. In either case, the client immediately updates its local expert-to-server mapping to remove the failed server from its list of available resources. It then automatically re-transmits the original request to an alternative server that hosts a replica of the required expert. This failover mechanism is designed to be transparent and have a negligible impact on the final inference outcome, ensuring continuous and robust service delivery.

4 Optimizations

EAAS is an integrated, end-to-end MoE serving system that combines SGLang [35]-based attention clients, an RDMA-based asymmetric asynchronous peer-to-peer (P2P) communication library based on IBGDA [20], and a self-designed [24] MoE server. Custom Triton and CUDA kernels have been developed to reduce CPU involvement and to utilize CUDA Graph [18] for decreasing kernel launch overhead.

4.1 Kernel Optimization for Expert Server

A key challenge in the EAAS architecture is managing the non-uniform workloads from imbalanced batches in large-scale MoE serving, which often leads to underutilized GPUs. We address this with specialized kernel optimizations (Algorithm 1) that use a static GPU grid. Each block independently iterates over valid tokens, skips empty batches, and

Algorithm 1 Iterate tokens with various length of batches.

Require: *num_tokens*: Array of token counts per batch
Require: *batch_size*: Number of batches
Require: *compute_func*: The function to process a single token
 1: **Alloc shared memory:** $s\_num\_tokens[MAX\_BATCH]$
 2: **Initialize:** $token\_id \leftarrow blockIdx.x$
 3: **Load** $s\_num\_tokens \leftarrow num\_tokens$
 4: **for** $batch\_id = 0$ **to** $batch\_size - 1$ **do**
 5:     $num\_token\_this\_batch \leftarrow s\_num\_tokens[batch\_id]$
 6:     **while** $token\_id < num\_token\_this\_batch$ **do**
 7:         **Call** $compute\_func(batch\_id, token\_id)$
 8:         $token\_id \leftarrow token\_id + gridDim.x$
 9:     **end while**
10:     $token\_id \leftarrow token\_id - num\_token\_this\_batch$
11: **end for**

---

uses shared memory to distribute tokens in a strided fashion. This design effectively balances workloads across GPU threads, allowing servers to adapt to imbalances without idle threads or synchronization overhead.

A second challenge arises from hosting many experts on each server. The standard `DeepGEMM` implementation creates performance bottlenecks by inefficiently iterating over all potential expert groups, most of which are inactive, and incurring a separate, low-throughput global memory read for each one. To resolve this, we introduce a `group-shrink` kernel that uses a GPU prefix scan to isolate active groups and move their metadata to the front so that we can early-stop. The `DeepGEMM` scheduler is then modified to iterate over this compacted tensor, which is loaded into shared memory once to eliminate inefficient, repeated memory reads.

These kernel-level optimizations, in concert with our dynamic batching and token reorganization strategies, enable expert servers to sustain high GPU utilization and throughput. By effectively managing these dynamic and sparse MoE workloads, our system overcomes the traditional bottlenecks of load imbalance and resource underutilization.

4.2 Double-Batch-Overlap in Attention Client

In EaaS architecture, a critical performance bottleneck is the communication latency inherent in dispatching tokens from attention clients to remote expert servers. This network round-trip time can force the client's GPU into an idle state while it awaits results, creating bubbles in the execution pipeline that directly degrade overall throughput. To counteract this inefficiency, we implement the Double-Batch-Overlap [14] strategy, a pipelining technique specifically designed to overlap this communication and remote computation latency.

This strategy ensures continuous GPU engagement by maintaining two active batches in the client's pipeline. The process unfolds as follows: while a previously dispatched batch (Batch A) is being processed by the remote expert
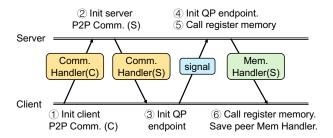

Figure 7. The process of establishing IBGDA connection

servers, the client's GPU does not wait. Instead, it immediately commences the computationally intensive attention calculations for the subsequent batch (Batch B). This temporal overlap is the key mechanism; the attention computation for Batch B effectively hides the network and processing latency of Batch A. As a result, by the time the expert results for Batch A are returned to the client, Batch B is fully prepared for dispatch. This seamless handover creates a continuous, overlapping workflow that transforms potential idle periods into productive computation. By minimizing GPU stalls, this approach directly increases resource utilization, sustains a higher operational intensity for improved system throughput, and reduces the overall end-to-end latency for each request.

4.3 CUDA Graph for End-to-End Acceleration

High overhead from launching numerous small CUDA kernels can limit LLM inference performance. CUDA graphs address this by capturing the entire sequence of kernel launches into a single, reusable unit, which reduces CPU launch overhead.

Our use of a CPU-free communication library based on IBGDA is instrumental in maximizing the benefits of this technique. Because all network operations are initiated from the GPU without CPU intervention, we can capture the complete end-to-end workflow, from attention computation and data dispatch to result reception and final processing, within a single CUDA graph on the attention client. This holistic capture eliminates nearly all CPU-side launch overhead for the client's entire operational loop.

By enabling graph-based execution on both clients and servers, EaaS minimizes kernel launch overhead across the distributed system, enhancing overall efficiency and reducing latency.

4.4 Flexible IBGDA Connection

We utilize a client-server architecture to manage IBGDA connections, which ensures flexible deployment of EaaS services. The process for establishing a new IBGDA connection is illustrated in Figure 7. Firstly, the client initiates an HTTP request to the server ①, transmitting its IBGDA P2P communication handler (P2P Comm). Upon receiving this request,

the server performs a mirroring operation ②. Meanwhile, the client begins to change its QP state and notifies the server ③④. Finally, the client and server complete the registration of the GPU buffer and exchange their respective MR and buffer addresses ⑤⑥, marking the peer's readiness state in their mappings.
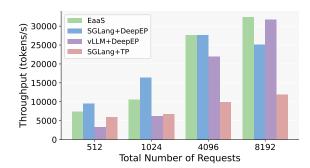
To maintain a unified global view of the connection state, we introduce a central monitor. This monitor tracks the health of all clients and servers and broadcasts when a rank comes online or goes offline. It can be implemented using a distributed consistency component like ZooKeeper [7]. Figure 6 illustrates the system's fault-tolerance mechanisms for both client and server failures. When a server stops serving, the monitor's heartbeat mechanism detects the change, allowing its clients to update their mappings and switch to another available server. Conversely, a server will release the buffer of any client that has lost its heartbeat. This enables changing the number of instance without requiring a restart. Thanks to the monitor and client-server based IBGDA connections, EaaS only needs to rebuild connections on demand. This avoids the costly overhead of rebuilding the entire communication group during dynamic scaling or fault tolerance events.

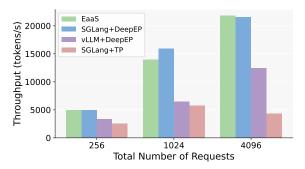## 4.5 Dynamic Load Balancing for Experts

EaaS is compatible with current mainstream MoE load balancing schemes, such as EPLB [3], which balances the load among instances of expert parallelism by reordering and adding redundant experts. Furthermore, EaaS can use existing load-balancing strategies to provide a wider array of balancing methods. Specifically, three categories are possible. First, unlike expert parallelism, EaaS does not restrict the number of experts on each expert server to the same. This provides an opportunity for load balancing by adjusting the number of experts on each server. Second, based on the flexible IBGDA connection mechanism discussed above, EaaS can dynamically balance the load by scaling up or down the number of service instances for 'hot' and 'cold' experts. Finally, due to the decoupled deployment architecture, it is also possible to balance the load by changing the compute/memory specifications of the instances on which the experts are hosted. In conclusion, on top of existing load-balancing strategies, EaaS provides a richer set of load-balancing methods for deploying MoE model inference services, particularly in scenarios like cloud services. We will also explore load-balancing algorithms specifically designed for EaaS in future works.

5 Evaluation

In this section, we evaluate the performance of EaaS in comparison with state-of-the-art solutions for large-scale MoE model serving. Our experiments are designed to address the following questions:

(a) 128 GPUs (64 prefilling and 64 decoding).
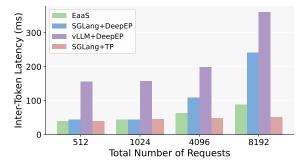
(b) 48 GPUs (16 prefilling and 32 decoding).

Figure 8. This figure compares the decoding throughput of EaaS against baseline systems across various request loads for two different GPU cluster sizes. The results demonstrate EaaS's competitive end-to-end performance and scalability.
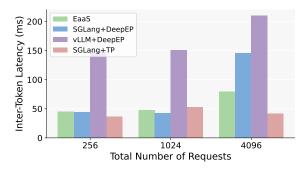
- What is the overall performance of EaaS in realistic serving scenarios? (§5.2)
- How is EaaS's scalability with various amounts of workloads? (§5.3)
- Does EaaS maintain acceptable performance during the recovery period after GPU failures? (§5.4)
- Is the CPU-free design necessary for MoE serving? How efficient is the P2P communication library proposed in this work, compared with existing solutions? (§5.5)
- How much do key design components and implemented techniques contribute to the system's overall performance? (§5.6)

5.1 Evaluation Setup

Hardware and frameworks. We use 16 computing nodes, each equipped with 8 Nvidia Hopper GPUs (80GB), connected via NVLink within the node and 8×400Gbps RoCE links across nodes. For frameworks, we use PyTorch 2.7.1, Python 3.12, and CUDA 12.6.

(a) 128 GPUs (64 prefilling and 64 decoding).

(b) 48 GPUs (16 prefilling and 32 decoding).

Figure 9. EaaS successfully balances high throughput with low inter-token latency, a key challenge in large-scale model serving. The system's architecture allows it to achieve strong processing speeds while maintaining acceptable responsiveness for users. Evaluations confirm that EaaS delivers competitive throughput and latency compared to state-of-the-art systems.

Model and workloads. We evaluate EaaS on the DeepSeek-R1 [5] 671B model under large-scale scenarios. For workloads, we adopt ShareGPT [29]. To mitigate long-tail effects, we cap the maximum response length at 768 tokens.

Baselines. We compare EaaS with the state-of-the-art inference engine SGLang [35] (commit ID: 51cdd81f) and vLLM [10] (commit ID: 9b01870). We also adopt relevant techniques from the SGLang codebase (e.g., KV caching) that are orthogonal to our contributions. Both SGLang and EaaS implement PD (Prefill-Decode) disaggregation with Moon-cake [25], while vLLM does not use disaggregation as this feature of its is currently experimental. SGLang supports two serving modes for DeepSeek models: Tensor Parallelism (TP) and Expert Parallelism (EP) via DeepEP [14], a high-performance communication library. We include both modes to provide a comprehensive comparison across different communication patterns. Similarly, we use vLLM+DeepEP as another baseline. Since the same expert load balancing algorithm yields different expert distributions under EaaS and the baselines due to their distinct arrangements, we do not

incorporate load balancing in the main evaluations. Instead, we include it in the ablation study to examine EaaS's compatibility with these algorithms.

In addition, MegaScale-Infer [36] and StepMesh [31] propose asymmetric communication libraries based on GDR-Copy [22], which rely on the CPU during communication. Since MegaScale-Infer's code is not publicly available, we compare our end-to-end communication overhead against StepMesh to highlight the necessity of our CPU-free design.

Key metrics. We measure two primary metrics: throughput and inter-token latency (ITL). For throughput, we focus on the decoding phase, as EaaS is applied to disaggregated decoding. We provision sufficient prefilling resources to ensure decoding throughput is not bottlenecked. ITL is measured to assess the ability of each method to deliver acceptable per-request latency. All metrics are measured using SGLang's original serving benchmark scripts, and we report the end-to-end results across all submitted requests.

5.2 End-to-End Evaluation

We evaluate the overall performance of EaaS under two settings: (1) 16 nodes (128 GPUs) with 8 nodes for prefilling and 8 nodes for decoding, and (2) 6 nodes (48 GPUs) with 2 nodes for prefilling and 4 nodes for decoding, as shown in Figure 8 and Figure 9.

For the decoding stage, SGLang+DeepEP (referred to as SGL-EP) employs full Expert Parallelism (EP sizes of 64 and 32, respectively). In contrast, EaaS adopts a client-server architecture with 32 clients–32 servers and 16 clients–16 servers. SGLang+TP (referred to as SGL-TP) is constrained to units of 16 GPUs due to the limitation of tensor parallelism. Consequently, the model must be duplicated 4 times and 2 times in the two settings when using TP.

SGL-EP achieves competitive throughput and inter-token latency when the request volume is relatively small. However, under heavy traffic, we observe pronounced long-tail latencies, which we attribute to limitations in its request transportation and load balancing across the Data Parallel Attention [30] ranks. In contrast, SGL-TP maintains low and stable ITL due to its small inference unit size (16 GPUs per unit). Yet, this small unit size also requires multiple replications of the model weights, reducing the available memory per GPU and limiting the maximum batch size. As shown in Figure 8, this results in substantially lower throughput compared to other methods.

The current PD disaggregation in vLLM is still experimental and relatively unstable. Thus, for vLLM, we allocate the same nodes for prefilling and decoding and, for fairness, evaluate decoding on 64 and 32 GPUs while excluding prefilling overhead. From the results, vLLM delivers high throughput under large-scale configurations but suffers from higher inter-token latency relative to other methods.

Finally, EaaS benefits from its efficient communication library and balanced resource allocation between attention
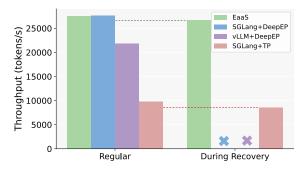
Figure 10. In our system fault tolerance experiment, EₐₐS proved resilient by maintaining high throughput without crashing, exhibiting only a slight performance decrease during recovery. DeepEP solutions, however, cannot continue operating under such failures. Because their monolithic architecture depends on a single communication group, the loss of one GPU forces the entire system to restart, causing a total service outage.
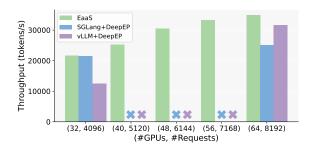

Figure 11. Weak scaling with fine granularity. SGLang and vLLM can only scale with GPU counts that evenly divide 7168, whereas EₐₐS supports arbitrary GPU counts, enabling more flexible and cost-efficient scaling.

and expert computation. As a result, it achieves strong throughput while maintaining acceptable latency. Overall, EₐₐS demonstrates competitive end-to-end performance compared to state-of-the-art systems such as SGLang and vLLM, while additionally offering features like fault tolerance, which we will evaluate in the following sections.

5.3 Scalability

The scalability of EₐₐS was evaluated through a weak scaling experiment designed to demonstrate its capacity for dynamic, cost-effective resource allocation. As illustrated in Figure 11, the disaggregated architecture of EₐₐS exhibits a proportional, near-linear increase in throughput when scaling from 32 to 64 GPUs, indicating efficient resource utilization without significant overhead. This fine-grained scalability presents a considerable advantage over monolithic systems such as SGL-EP and vLLM-EP. These baseline

systems are unable to operate on configurations with non-standard GPU counts (e.g., 40, 48, 56) due to their inherent architectural rigidity. Such inflexibility constrains operators to inefficient, coarse-grained scaling increments, resulting in resource stranding and elevated operational expenditures. Conversely, the service-oriented design of EₐₐS facilitates precise, demand-based provisioning by allowing the number of GPUs to be dynamically adjusted in response to workload fluctuations. For example, when the traffic decreases from 8192 to 5120, SGL-EP and vLLM-EP must still provision 64 decoding GPUs. In contrast, EₐₐS can scale down to 40 or 48 GPUs while maintaining the same level of decoding throughput as the baselines, thereby saving up to 37.5% of computing resources.
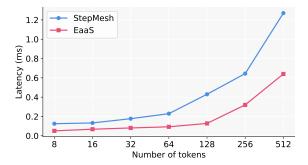
5.4 Fault Tolerance

To evaluate fault-tolerance capabilities, we randomly disable ten GPUs, one at a time, and measure the average decoding throughput during recovery. The experimental setup follows the previous section, using 64 GPUs for decoding with a total of 4096 requests. As shown in Figure 10, both SGL-EP and vLLM-EP are unable to continue serving during recovery because all decoding GPUs belong to a single communication group; thus, the failure of one GPU necessitates restarting the entire group. In contrast, SGL-TP benefits from its smaller execution unit: only a group of 16 GPUs must be restarted upon a failure, while the remaining units remain operational.

EₐₐS further improves resilience by pre-duplicating experts on servers, enabling expert requests originally routed to an offline server to be redirected to another server with a backup copy. Moreover, since attention clients in EₐₐS operate independently, client failures can also be efficiently handled by the routing mechanism of the PD disaggregation engine. Consequently, EₐₐS incurs less than 2% drop in decoding throughput when a GPU failure occurs.
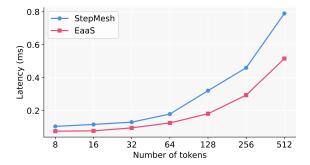
5.5 Comparison with Other Asymmetric Communication Libraries

To evaluate communication library performance in realistic scenarios, we construct PyTorch tensors with the same shape used during the decoding phase of DeepSeek R1 models: (batch size, 1, 7168). In this setup, the client sends the tensor to the server, which immediately returns it to the client. We measure the end-to-end latency from the initiation of the client transmission to the completion of the response. Two scenarios are considered: a symmetric setting (2 clients and 2 servers) and an asymmetric setting (1 client and 3 servers).

As shown in Figure 12, EₐₐS consistently achieves lower latency than StepMesh, reducing overhead by 49.6% and 34.7% in the two scenarios when the batch size is 512. The performance gains of EₐₐS stem from two key factors. First, by leveraging IBGDA to bypass the CPU, we eliminate the overhead associated with accessing host memory queues and CPU-issued control orders. Second, as the process is entirely

(a) Latency under symmetric scenario.

(b) Latency under asymmetric scenario.

Figure 12. End-to-end overhead of round-trip communication, measured from sending a tensor via the communication API to receiving the returned result.

CPU-free, the launch overhead of communication kernels can be further reduced through CUDA graph capture.

These results highlight the importance of CPU-free communication libraries, which existing systems such as StepMesh and MegaScale-Infer have yet to provide. More importantly, they demonstrate that reducing communication overhead at the library level directly translates into lower end-to-end latency for large-scale inference serving, thereby improving user-perceived responsiveness in real deployments.

5.6 Ablation Study

To better understand the contribution of each key component in EAAS, we conduct an ablation study by selectively disabling them and measuring the resulting throughput. Specifically, we examine three optimizations: (1) CUDA Graph capture, which reduces kernel launch overhead by eliminating CPU involvement; (2) kernel shrinking, which compacts group metadata to improve expert server efficiency; and (3) double batching, which overlaps attention computation with communication to mask latency.

As shown in Figure 13, removing any of these optimizations results in a clear performance degradation. Using batch size 4096 as an example, disabling CUDA Graph causes a dramatic **91.9%** throughput drop due to repeated CPU-side
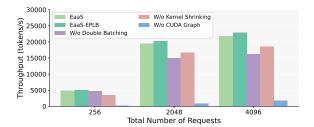

Figure 13. This ablation study demonstrates that EAAS's key optimizations are all critical for performance, as disabling any of them causes a severe drop in system throughput.

kernel launch overhead. Without kernel shrinking, throughput decreases by **14.9%** because of inefficient scheduling over sparse expert groups. Finally, removing the double-batching strategy leads to a **25.9%** decline, as communication latency is no longer masked by overlapping attention computation. Together, these results confirm that all three optimizations are critical for achieving the overall efficiency of EAAS.

We also evaluated EAAS in combination with the expert load balancing algorithm proposed by DeepSeek [14]. The results show performance improvements of 2.4%, 4.4%, and 5.1%, respectively. In future work, we plan to investigate expert load balancing strategies specifically tailored for EAAS.

6 Related Works

LLM Serving Frameworks. For efficient LLM serving, several frameworks have been proposed. vLLM [10] introduces PagedAttention for efficient KV cache memory management. SGLang [35] achieves efficient serving with RadixAttention for prefix caching. MLC-LLM [17] enables serving on diverse platforms through a machine learning compiler, while TensorRT-LLM [21] provides flexible LLM APIs with state-of-the-art optimizations and emphasizes ease of use. These frameworks are optimized for dense models and do not address the dynamic sparsity characteristic of MoE models. In this work, we leverage the inherent sparsity in MoE models to develop an efficient and robust serving system tailored specifically for them.

Parallelisms For LLMs. To facilitate large-scale LLM inference and training, various forms of parallelism such as Tensor Parallelism (TP) [27], Pipeline Parallelism (PP) [6, 8, 11, 12, 15], and Data Parallelism (DP) [26, 34] have been used. In adapting the sparsity inherent in MoE models, Expert Parallelism (EP) was introduced to distribute experts across devices. In this work, we decouple EP from the dense portion of the model and from other parallelism dimensions, thereby enabling flexible scheduling and enhanced robustness. This decoupling permits group-free and resilient MoE serving.

Recent works such as Megascale-Infer [36] and the serving system of Step-3 [32] have also explored disaggregating

attention and MoE layers into separate tiers. However, their approaches rely on CPU-controlled communication libraries based on technologies like GDRCopy. This reliance on the CPU for managing communication introduces additional overhead and prevents the end-to-end capture of CUDA graph, and further limiting potential optimizations, despite their disaggregated architecture.

Communication Libraries. Efficient and stable communication libraries are imperative to support these parallelism strategies. NCCL [19] is widely used in both model training and inference for its ease of integration, while NVSHE-MEM [23] provides scalable and efficient communication for NVIDIA GPUs. Moreover, systems such as Centauri [1] and Concerto [2] have demonstrated that overlapping techniques can reduce distributed communication overhead. In our work, we introduce a novel asymmetric asynchronous peer-to-peer communication library that supports our decoupled MoE servers by offering CPU-free communication with IBGDA [20].

Conclusion

We presented EaaS, a serving system designed for large-scale Mixture-of-Experts (MoE) models. By disaggregating MoE layers into independent services, EaaS achieves fine-grained elasticity, robust fault tolerance, and high efficiency. At its core, EaaS introduces a CPU-free, asynchronous P2P communication library that provides additional acceleration compared with existing solutions. Our evaluation demonstrates that EaaS delivers competitive throughput and latency compared to state-of-the-art systems while maintaining resilience under failures and strong scalability. This service-oriented paradigm establishes a flexible and efficient foundation for deploying large-scale MoE models in production.

References

[1] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. 2024. Centauri: Enabling Efficient Scheduling for Communication-Computation Overlap in Large Model Training via Communication Partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24).* Association for Computing Machinery, New York, NY, USA, 178–191. doi:10.1145/3620666.3651379

[2] Shenggan Cheng, Shengjie Lin, Lansong Diao, Hao Wu, Siyu Wang, Chang Si, Ziming Liu, Xuanlei Zhao, Jiangsu Du, Wei Lin, and Yang You. 2025. Concerto: Automatic Communication Optimization and Scheduling for Large-Scale Deep Learning. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25).* Association for Computing Machinery, New York, NY, USA, 198–213. doi:10.1145/3669940.3707223

[3] deepseek ai. 2025. Expert Parallelism Load Balancer (EPLB). https://github.com/deepseek-ai/EPLB

[4] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948* (2025).

[6] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2018. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. doi:10.48550/ARXIV.1811.06965

[7] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (Boston, MA) *(USENIXATC'10).* USENIX Association, USA, 11.

[8] Arpan Jain, Ammar Ahmad Awan, Asmaa M. Aljuhani, Jahanzeb Maqbool Hashmi, Quentin G. Anthony, Hari Subramoni, Dhableswar K. Panda, Raghu Machiraju, and Anil Parwani. 2020. GEMS: GPU-Enabled Memory-Aware Model-Parallelism System for Distributed DNN Training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–15. doi:10.1109/SC41405.2020.00049

[9] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23).* Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[11] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.* 1–14.

[12] Junfeng Lin, Ziming Liu, Yang You, Jun Wang, Weihao Zhang, and Rong Zhao. 2025. WeiPipe: Weight Pipeline Parallelism for Communication-Effective Long-Context Large Model Training. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming* (Las Vegas, NV, USA) *(PPoPP '25).* Association for Computing Machinery, New York, NY, USA, 225–238. doi:10.1145/3710848.3710869

[13] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434* (2024).

[14] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).

[15] Ziming Liu, Shenggan Cheng, Haotian Zhou, and Yang You. 2023. Hanayo: Harnessing Wave-like Pipeline Parallelism for Enhanced Large Model Training Efficiency. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23).* Association for Computing Machinery, New York, NY, USA, Article 56, 13 pages. doi:10.1145/3581784.3607073

[16] Meta. 2025. The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation. https://ai.meta.com/blog/llama-4-multimodal-intelligence/. (2025).

[17] MLC team. 2023. *MLC-LLM.* https://github.com/mlc-ai/mlc-llm

[18] NVIDIA. 2019. Getting Started with CUDA Graphs. https://developer.nvidia.com/blog/cuda-graphs/.

[19] NVIDIA. 2020. NVIDIA Collective Communications Library. https://developer.nvidia.com/nccl

[20] NVIDIA. 2022. Improving network performance of HPC systems using NVIDIA Magnum IO NVSHMEM and GPUDirect Async. https://developer.nvidia.com/blog/improving-network-performance-of-hpc-systems-using-nvidia-magnum-io-nvshmem-and-gpudirect-async.

[21] NVIDIA. 2024. *TensorRT-LLM*. https://github.com/NVIDIA/TensorRT-LLM

[22] NVIDIA. 2025. GDRCopy. https://github.com/NVIDIA/gdrcopy.

[23] NVIDIA. 2025. NVSHMEM. https://developer.nvidia.com/nvshmem.

[24] OpenAI. 2021. Triton. https://github.com/triton-lang/triton.

[25] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2025. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079 [cs.DC] https://arxiv.org/abs/2407.00079

[26] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. arXiv:1910.02054 [cs.LG] https://arxiv.org/abs/1910.02054

[27] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] https://arxiv.org/abs/1909.08053

[28] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534* (2025).

[29] ShareGPT Team. 2023. ShareGPT. https://sharegpt.com/.

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

[31] StepFun Team. 2025. Step-3 is Large yet Affordable: Model-system Co-design for Cost-effective Decoding. arXiv:2507.19427 [cs.LG] https://arxiv.org/abs/2507.19427

[32] Bin Wang, Bojun Wang, Changyi Wan, Guanzhe Huang, Hanpeng Hu, Haonan Jia, Hao Nie, Mingliang Li, Nuo Chen, Siyu Chen, et al. 2025. Step-3 is Large yet Affordable: Model-system Co-design for Cost-effective Decoding. *arXiv preprint arXiv:2507.19427* (2025).

[33] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[34] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] https://arxiv.org/abs/2304.11277

[35] 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. 2024. SGLang: Efficient Execution of Structured Language Model Programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=VqkAKQibpq

[36] Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. 2025. MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism. arXiv:2504.02263 [cs.DC] https://arxiv.org/abs/2504.02263