RMAI: Rethinking Memory for AI (Inference)

In-Kernel Remote Shared Memory as a Software Alternative to CXL

Amir Noohi
University of Edinburgh
Edinburgh, UK
amir.noohi@ed.ac.uk

Mostafa Derispour
Isfahan University of Technology
Isfahan, Iran
m.deris@ec.iut.ac.ir

Antonio Barbalace
University of Edinburgh
Edinburgh, UK
antonio.barbalace@ed.ac.uk

Abstract

As AI models grow exponentially in size, memory has emerged as a critical bottleneck for inference at scale. While hardware solutions like Compute Express Link (CXL) promises to solve the problem of memory capacity and sharing, they require capital investment, and are not widely available. This paper presents RMAI, an in-kernel remote shared memory framework tailored for AI inference workloads, offering a transparent, scalable, and cost-effective software alternative to hardware-based memory expansion and sharing solutions. By leveraging the operating system's capabilities, RMAI introduces dynamic virtual memory regions that reduce page faults, minimize overheads associated with user-kernel transitions, and optimize data locality for inference workloads. In this paper, we particularly focus on Mixture-of-Experts (MoE) models. In this initial evaluation we demonstrate that RMAI achieves performance levels comparable to CXL-like architectures, with up to 10x faster expert switching and reduced memory management overhead across large-scale inference tasks compared to disk-based solutions. This work redefines the role of remote shared memory in AI systems, positioning it as a practical and high-performance solution for memory capacity and sharing in modern data centers.

CCS Concepts

• **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Distributed artificial intelligence**; • **Networks** → **Programming interfaces**.

Keywords

Memory disaggregation, Distributed shared memory, Mixture of Experts, Kernel-level memory management, RDMA (Remote Direct Memory Access), Compute Express Link (CXL), Operating system memory management, Partitioned Global Address Space (PGAS), Large-scale AI models

ACM Reference Format:
Amir Noohi, Mostafa Derispour, and Antonio Barbalace. 2025. RMAI: Rethinking Memory for AI (Inference): In-Kernel Remote Shared Memory as a Software Alternative to CXL. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3721146.3721954

EuroMLSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1538-9/25/03
https://doi.org/10.1145/3721146.3721954

1 Introduction

The unprecedented scale of Large Language Models (LLMs) has fundamentally reshaped artificial intelligence, creating critical challenges in memory management and system architecture. Models like Google's Switch Transformer (1.6 trillion parameters) [18], GLaM (1.2 trillion parameters) [15], and M6-T (>1 trillion parameters) [42] exemplify this growth, leveraging Mixture-of-Experts architectures to strategically partition parameters across multiple experts while activating only a subset for each input [28]. This design achieves computational efficiency but introduces significant memory demands during inference, requiring rapid access to massive parameter sets, far exceeding the capacity of a single node's RAM. Parameters not immediately needed are offloaded to slower memory tiers, such as local storage devices, resulting in latency overheads that hinder performance [43].

Traditional datacenters with monolithic servers exacerbate memory constraints by rigidly coupling CPU and RAM resources, often leading to low memory utilization (under 40% in Google and 60% in Alibaba) [12, 35, 37] and poor efficiency in HPC [31]. To address this, datacenters are moving toward *disaggregated* architectures that decouple compute from memory [16]. Such designs promise to meet the 2–3× higher memory requirements of large-scale inference [19, 43] more flexibly than traditional servers, often relying on RDMA or CXL for high-speed data access [26, 39].

While GPUs dominate AI *training*, modern CPUs have become increasingly viable for AI *inference* thanks to accelerators like Intel's AMX [2] and ARM's SME [1], yielding for example 5–10× gains in quantized inference [5, 32]. CPUs not only offer ample memory capacity and ease programmability

but also avoid offloading overheads [34]. As they incorporate these specialized matrix units and expand bandwidth [7], CPUs can effectively support large-model inference workloads alongside GPUs and accelerators [2, 32, 34].

Memory Expansion. Traditional disk-based memory solutions, even high-performance NVMe SSDs with 7GB/s+ sequential throughput and 100-200µs latency, while ubiquitous, suffer from severe latency and limited bandwidth, making them ill-suited for inference workloads where performance depends on rapid data access. When RAM is full, and inference requests require loading parameters from disk, the resulting bottlenecks drastically degrade performance, rendering the computational power of the system ineffective.

Compute Express Link (CXL) offers the potential for shared memory pools across nodes, enabling elastic scaling. However, current deployments face significant challenges. While direct CXL connections are limited by physical distance (typically 2 meters), multi-rack deployments are possible through CXL switch-based topologies. The challenge is that each additional switch hop introduces significant latency overhead, limiting practical scalability in larger deployments. Efforts to extend direct connection range using optical signaling remain experimental and introduce signal loss, increased power consumption, and deployment costs [39]. Efforts to extend this range using optical signaling remain experimental and introduce signal loss, increased power consumption, and deployment costs [39]. Additionally, CXL adoption requires costly hardware upgrades, with power consumption concerns (2–3 W/GB/s) and limited hardware availability further restricting scalability [39]. While ASIC-based CXL solutions are now available from several manufacturers, CXL switches remain extremely limited in the market, with Xconn's model being one of the few commercially available options and difficult to procure. This limited switch availability further constrains the deployment of large-scale CXL-based memory pooling solutions [39].

Remote memory accessed via RDMA, on the other hand, presents a practical and already-deployed alternative that solves several key issues of memory disaggregation [6, 14, 20]. RDMA offers low latency (1–2 µs), high bandwidth (25–800 Gbps), and cost-efficient scalability, enabling nodes to efficiently utilize underused memory on other machines [39]. This approach not only bypasses the limitations of disk-based solutions but also avoids the infrastructure barriers of CXL. Despite its established use in memory disaggregation, RDMA's potential to address the dynamic memory requirements of AI-specific workloads, such as Mixture-of-Experts (MoE) models, has not been fully explored. By integrating remote memory with MoE architectures, we propose a novel solution that leverages the scalability of RDMA to meet the inference demands of modern AI, while overcoming bottlenecks inherent to local memory and disk.

Contributions. This paper presents the first analysis of using remote memory as a faster memory tier than local disk to cache—and share—AI/ML workload parameters, with particular emphasis on Mixture-of-Experts architectures. While remote memory—i.e., swapping in/out memory from/to the memory of another machine over RDMA—has been studied before [14, 20, 25, 30] in the context of cloud workloads, the unique memory access patterns and performance requirements of modern AI workloads necessitate a fundamental rethinking to also enable *data sharing* among multiple machines (albeit read-only sharing). Our design is *inspired by the Partitioned Global Address Space (PGAS) model*, which naturally accommodates transparent remote data placement and direct addressing, allowing multiple machines to reference shared remote memory without per-object lookups or complicated directory tables.

Our approach demonstrates several critical advantages over existing solutions. High Performance: 40% reduction in total inference time compared to disk-based solution and 10% compare to CXL solution. Low Latency: up to 10× improvement in expert loading speed versus disk and up to 2x faster load/unload compare to CXL. High Scalability: Near-linear performance scaling across distributed nodes. Improved Compatibility: Seamless integration with existing AI frameworks.

This work makes several contributions to the field: (1) a comprehensive analysis of MoE memory access patterns and associated optimization opportunities; (2) novel remote shared memory PGAS-like architecture specifically engineered for efficient expert loading; (3) RDMA-based communication protocol optimized for large-scale inference; (4) extensive evaluation demonstrating performance parity with hardware solutions while enabling flexible deployment. This paper presents a first step toward fully realizing transparent, kernel-level remote memory for MoE. Nonetheless, our initial results confirm the viability and efficiency of this approach, laying the groundwork for future advances in large-model inference.

2 Background

2.1 Modern Datacenter Architecture

Modern datacenters are designed to handle diverse workload requirements through a mix of compute-optimized and memory-optimized nodes. Compute-optimized nodes typically include AI-enabled processors (e.g., Intel AMX, Arm SVE) or accelerators like GPUs and TPUs, prioritizing high

Table 1: Comparison of Memory and Interconnect Technologies

Technology	Cost	Availability in Data Centers	Latency	Bandwidth
DDR4 RAM	$6.20–$12.40 per GB	Widespread	80–100 ns	208 Gb/s
CXL 1.1 Memory	$1,860 per 128 GB	Limited	245–255 ns	136–208 Gb/s
RDMA	$62–$1,240 per 25–800 Gb/s	High	1–2 µs	25–800 Gbps
NVMe Storage	$496 per 1 TB	Widespread	92,000–537,000 ns	36 Gb/s

computational throughput but constrained by limited memory capacities of 32–64GB due to high costs and power limitations of high-bandwidth memory. Conversely, memory-optimized nodes feature large memory capacities (384 GB–2 TB) with basic CPUs and are intended for data-heavy tasks [6, 39].

However, these resources are often underutilized due to the fixed CPU-to-memory ratio in monolithic architectures [35, 37]. The growing memory demands of AI workloads, such as inference for large models, exacerbate this inefficiency, where memory requirements can exceed available RAM. Network-based memory disaggregation has emerged as a practical solution. With modern datacenter networks offering 25–800 Gbps bandwidth, and RDMA capable of sub-2 µs latencies, remote memory can now be accessed at speeds far superior to traditional disk-based approaches, which suffer from millisecond-scale latencies and bandwidth limitations (see Table 1) [14, 20]. This enables memory-optimized nodes to serve as high-performance extensions for compute nodes, reducing costs and improving resource utilization.

2.2 Mixture-of-Experts Architecture and Memory Demands

Mixture-of-Experts (MoE) models use conditional computation so that only relevant portions of the network are activated per input. As shown in Figure 1, gating selects which experts process each token (sparse activation), routing dispatches tokens across devices via an All-to-All pattern, and experts run in parallel while typically needing 1–4 GB each. Because total expert memory can exceed a single device's capacity, results are combined according to the gating decisions to produce the final outputs, demanding advanced memory management. These frequent expert loads and unloads make MoE an ideal use case for remote shared memory solutions, such as RDMA-based disaggregation, which can scale beyond local memory limits.

3 Motivation

Current approaches to Mixture of Experts offloading primarily rely on disk-based storage or advanced hardware solutions such as CXL. However, these methods exhibit critical limitations: disk-based solutions suffer from high latency and low bandwidth, while CXL remains expensive,

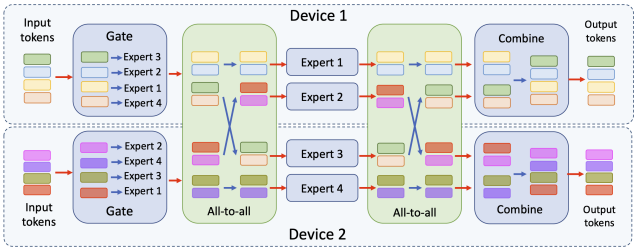

Figure 1: MoE architecture[22]

limited in availability, and confined to within-rack deployments [11, 27]. Despite these constraints, many datacenters possess underutilized memory resources on nodes not engaged in high-performance computation. These **memory nodes** represent an untapped opportunity for MoE workloads, yet no transparent and optimized method currently exists to harness this resource effectively.

Temporal Locality in MoE Workloads. Memory access patterns in MoE workloads exhibit significant temporal locality, as observed in various studies [11]: (1) **70-80% of expert activations occur within 100ms windows**, indicating bursty memory demand driven by dynamic input routing. (2) **Expert reuse follows a power-law distribution ($\alpha \approx 1.5$)**, with a small subset of experts being accessed disproportionately often. (3) **The average expert residency time is 200-300ms**, reflecting rapid memory turnover before eviction or replacement. These patterns suggest that caching mechanisms could mitigate disk-related overheads, yet disk I/O latencies and limited bandwidth render such solutions unsuitable for inference tasks requiring real-time responses [17].

Access Frequency and System Overhead. The dynamic expert selection in Mixture-of-Experts models results in frequent expert migration, which introduces substantial overhead to system performance. Each expert load or unload operation typically occurs within 50-100ms intervals, as dictated by dynamic gating decisions that allocate specific experts for processing based on input characteristics [19, 27]. These frequent transitions generate high memory management overhead, including significant page fault activity. For instance, a 4GB expert mapped into memory using standard 4KB pages results in over a million page faults if not mitigated by memory management features like Transparent Huge Pages (THP) [17]. This page fault overhead can severely delay inference and increase system latency.

Additionally, MoE workloads require substantial memory bandwidth during expert loading and unloading, with transfer rates often reaching 16-32 GB/s, which can strain performance in systems with limited memory bandwidth [11, 17]. While modern memory systems offer theoretical bandwidths of up to 100 GB/s, high-frequency, bursty access patterns—exacerbated by temporal locality—often cause contention and reduce effective throughput [27]. This memory contention compounds the latency introduced by disk-based systems or other lower-bandwidth tiers, further impairing performance.

Finally, scheduling delays, typically 50-100µs per expert switch, also add to the system overhead [19]. These delays arise from the time required to allocate and re-map experts into memory, manage interconnect contention, and update metadata structures like TLBs [17, 27]. Together, these factors make frequent expert migrations a critical bottleneck in MoE workloads, significantly reducing the computational efficiency of inference.

Although prior works have proposed techniques like expert pinning, weight caching, tensor parallelism, and memory offloading to mitigate these challenges [19, 23], they fail to fully address the core issue: the dynamic and frequent migration of expert parameters across the memory hierarchy. A more robust approach, capable of transparently leveraging underutilized resources such as remote shared memory, is promising.

3.1 Operating System Memory Management

Traditional OS memory management mechanisms struggle to efficiently handle MoE workloads due to frequent expert loading and unloading. These operations trigger substantial overhead in terms of page faults, memory fragmentation, and Translation Lookaside Buffer (TLB) pressure, all of which degrade performance.

Page Fault Overhead. Each time an MoE expert is loaded, the OS must allocate and map large memory regions, leading to thousands of page faults per expert if using 4KB pages. Without Transparent Huge Pages (THP), loading a 4GB expert requires approximately 1,048,575 page faults, resulting in a worst-case delay of 6–12 seconds (at 6–12 µs per page fault) However, using THP is limited to a set of sizes, such as 2MB and 1GB. Additionally, since Transparent Huge Pages (THP) are not available on all systems and require contiguous memory – which require expensive page migration, their use can result in extra overheads and inefficient memory utilization.

Impact of Expert Size on System Performance. Our empirical analysis of the MoE expert inference scenario, in which experts are loaded from disk—a common pattern in practice[27] —reveals significant performance implications across different expert sizes. This underscores the critical relationship between expert scale and system overhead. As illustrated in Figure 2, the total processing time exhibits a super-linear growth pattern as expert sizes increase from 128MB to 8GB, with the most dramatic impact observed in load/unload operations and computation time.

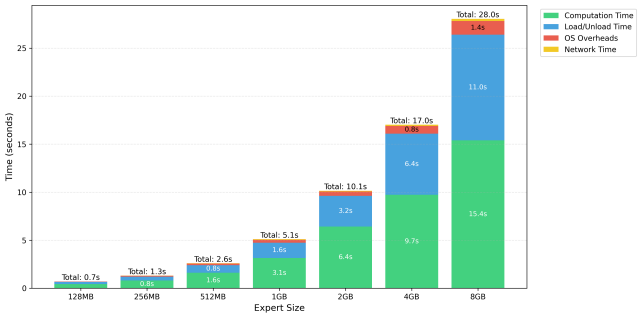

Figure 2: Time Components in Traditional Disk-Based Expert Loading

For smaller experts (128MB-512MB), the system maintains relatively balanced performance characteristics, with total processing times ranging from 0.7s to 2.6s. However, as expert sizes scale beyond 1GB, we observe a marked deterioration in system efficiency. The load/unload time, in particular, demonstrates concerning growth—from 0.22s at 128MB to 11.01s at 8GB—representing a significant portion of the total processing overhead. This growth pattern aligns with our earlier observations regarding memory management challenges and page fault activities.

The computational component, while scaling more gracefully, still shows substantial increase from 0.43s at 128MB to 15.39s at 8GB. Operating system overheads follow a similar trend, rising from 0.04s to 1.41s, reflecting the increasing strain on system resources. Notably, network transfer times remain relatively modest across all configurations (0.014s to 0.22s), suggesting that network bandwidth is not the primary bottleneck in current implementations.

This analysis reinforces our motivation for developing more efficient expert management strategies, particularly for larger expert sizes where traditional disk-based approaches show clear limitations. The disproportionate growth in load and unload times and OS overheads emphasizes the need for innovative solutions that can better handle the dynamic nature of MoE workloads while maintaining acceptable latency profiles.

4 Design

To develop a software alternative to CXL for commodity hardware we focused on two key aspects: (a) *transparency*, to enable seamless integration with userspace code; (b) *performance*, to ensure the proposed solution remains viable in environments without CXL support. To the best of our knowledge, there is no transparent and performance-optimized software solution for MoE CPU inference. Existing approaches

introduce either offloading to disk or non-commodity hardware such as CXL when memory is insufficient to store the MoE model.

Our goal is to provide the userspace application with the illusion of having the complete model – all experts, at all times, without requiring to manage or be aware of how the model data is distributed and handled. To achieve such goal we envision a PGAS-like approach where experts are placed in remote memory (that can be locally cached), but at the same time globally addressable by multiple compute nodes.

Symmetric Unified Virtual Address. To eliminate unnecessary copying and (de)serialization, a symmetric virtual address is essential. In this approach, all nodes share a consistent view of data, enabling direct data transfers between them.

To further enhance efficient data sharing within the cluster particularly for parameter exchange we extended this concept to create a unified virtual address space. This allows all nodes to directly access each other's data while maintaining ownership of their own portions.

Automatic Data Migration and Deallocation. In addition, these applications require access to large memory spaces that exceed the capacity of a single node's memory, necessitating memory distribution or sharding across multiple nodes. This approach requires efficient mechanisms for data migration. Furthermore, as individual nodes approach their capacity limits due to extensive memory access requirements, it becomes essential to implement policies for data deallocation. Two possible strategies exist: providing user-space APIs for these operations or offloading them to the kernel. The former increases the complexity of user-space applications, requiring developers to rewrite their code based on the API. Even after doing so, it introduces additional overhead due to user-kernel context switches. Therefore, we opted for the latter approach, automating data migration and deallocation at the kernel level to ensure transparency and efficiency.

Overall Architecture. As shown in Figure 3 our architecture comprises of two distinct node types: memory nodes and compute nodes. Each node holds a dedicated portion of experts, shared through RMAI. This distributed system creates the illusion that compute nodes possess the complete model, while RMAI automatically manages underlying operations such as data migration and deallocation. Existing PGAS solutions are not transparent and lack these functionalities and are therefore unsuitable for this scenario. Our design ensures high compatibility with existing user-space code, requiring only minimal modifications. Implementation involves simply replacing the model memory with RMAI-managed memory before execution.

Page Deallocation Policy. As noted in Section 3, MoE workloads exhibit temporal locality, requiring careful page deallocation. To address this, we implemented two key strategies. First, we maintain multiple copies of hot pages across both compute and memory nodes, ensuring pages belonging to frequently accessed experts remain available in compute nodes and can be retrieved from multiple memory nodes when necessary. Second, we employ an LRU page deallocation mechanism to free up less demanding pages associated with infrequently used experts.

5 Prototype Implementation

We implemented an initial prototype of our design in the Linux kernel with about 2000 LoC. While we could developed our prototype starting from an already existing remote memory project, we decided to started from scratch because previous projects (a) do not support sharing remote memory among different machines; (b) do only support page size granularity (indeed, FastSWAP supports object size, but it is not transparent).

Coarse-Grained Virtual Memory Regions. The default page size granularity is often 4096 bytes, which can lead to frequent page faults when working with large models, significantly impacting performance. To address this, we define Virtual Memory Regions (VMRs) that exceed 4096 bytes and are typically multiples of the default page size. Adjusting granularity through VMRs reduces the number of page faults and improves network utilization in terms of throughput by enabling batched message transmission. Additionally, it minimizes the number of control messages. However, excessively large VMRs can introduce challenges in AI workloads, which are highly multi-threaded. In such cases, multiple threads may become blocked on the same VMR, leading to underutilized CPU resources.

This limitation justifies why we do not use Transparent Huge Pages (THP), as their sizes are not configurable, offering only a few predefined options. Moreover, THP requires contiguous memory allocation, which can result in inefficient memory utilization. To optimize performance, we determine the ideal VMR size through evaluation, selecting the configuration that achieves the best results for our workloads. Larger VMRs reduce page faults and improve network utilization by enabling batched transfers. However, they can also increase CPU idle time, as the processor must wait for the entire VMR to be transferred before resuming execution.

During our experiments, we found that coarse-grained VMRs can reduce the number of page faults by up to 80%, decrease TLB operations by up to 4x, and enable TLB shootdown batching. Additionally, it improves network utilization by up to 3×. These optimizations reduce operating system overhead and enhance network efficiency.

In-Kernel Implementation. We chose to implement our solution in the kernel, offloading memory management to kernel

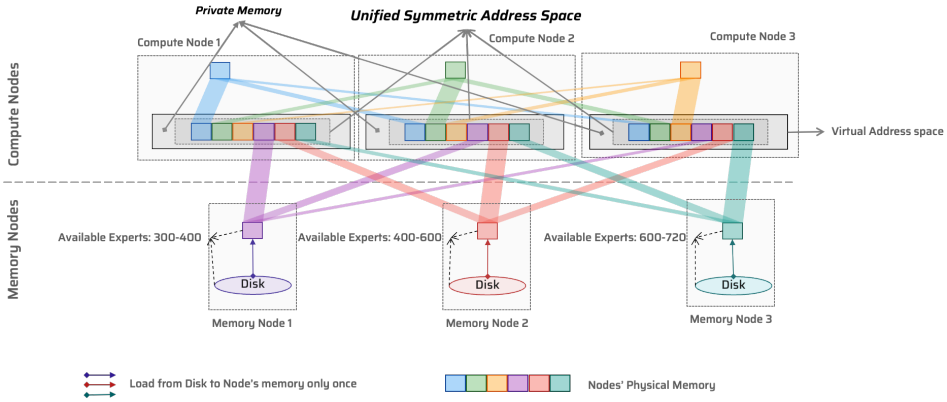

Figure 3: RMAI memory overview

while allowing user-space to handle computation logic. Implementing dynamic VMRs requires TLB modifications and PTE access, which are only available at the kernel level.

6 Evaluation

In this section, we provide a detailed evaluation of our RMAI system, comparing it against traditional disk-based and CXL-attached memory systems. We analyze the computation time, memory access overheads, and scalability across varying expert sizes, explaining each observation thoroughly.

6.1 Experimental Setup

Hardware. Experiments were conducted on two identical machines located in our Lab. Table 2 summarizes the hardware configuration used in our evaluation.

Component	Specification
CPU	Intel Xeon Gold 5418Y (2.0GHz, 24 cores)
Memory	512GB DDR5 @ 4800 MT/s
Storage	465.8GB NVMe SSD (WDS500G1X0E)
Network	100GbE Mellanox ConnectX-6
CXL Device	Samsung CXL 1.1 DRAM Memory Expander (128GB)

Table 2: Hardware configuration used for evaluation.

Software. We run our experiment on Linux kernel 5.15 and Pytorch version 2.6. The evaluation uses a custom implementation of a Mixture-of-Experts (MoE) model in PyTorch. The model incorporates: (1) LRU-based caching to manage expert access dynamically, (2) RDMA for fetching experts from remote nodes with minimal latency, (3) PGAS abstraction to present a transparent unified global memory view, (4) optimized memory management to handle disk, CXL, and RDMA memory tiers seamlessly, and (5) in our PyTorch MoE implementation, we did not modify the model's forward or gating logic to explicitly fetch experts. Instead, we mapped each expert's parameters into RMAI-managed virtual memory via `mmap()`. Thus, when the model accesses an expert's weights, a page fault occurs, and RMAI transparently retrieves the data from a remote memory node. This approach avoids manual syscall invocations in the model code – the remote memory access is completely transparent thanks to the PGAS abstraction. No explicit data-fetching function is called from PyTorch; the kernel automatically handles page faults and fetches required pages remotely. This ensures minimal modifications to existing AI frameworks.

6.2 Methodology

Three scenarios were evaluated: (1) *Disk-Based Memory (Baseline)*, where experts are loaded from NVMe storage into main memory. This incurs high overhead due to storage stack operations and page faults; (2) *CXL-Based Memory*, where experts are loaded from CXL-attached memory. This reduces I/O latency but retains overheads from data copying and re-registration; and (3) *RMAI (Our System)*, where RDMA fetches experts directly from remote memory. RMAI eliminates re-registration overhead and leverages the PGAS abstraction for simplified memory management.

6.3 Results and Analysis

Figure 4 illustrates execution times for varying expert sizes (128MB to 8GB), broken into four components: computation, load and unload time, OS overhead, and network time. We discuss those below.

6.3.1 Computation Time. The computation time remains consistent across all scenarios, measured at approximately 15.39 seconds for 8GB experts. This is expected because all approaches (disk-based, CXL-based, and RMAI) perform the forward computation entirely in main memory. Since computation depends only on the data already loaded, it remains unaffected by the memory architecture.

6.3.2 Load/Unload Time. The load/unload time shows significant differences. For disk-based memory, it is 11.01 seconds for 8GB experts due to high disk access latency and storage stack overheads, compounded by handling page faults for each 4KB memory page. In the CXL-based approach, this

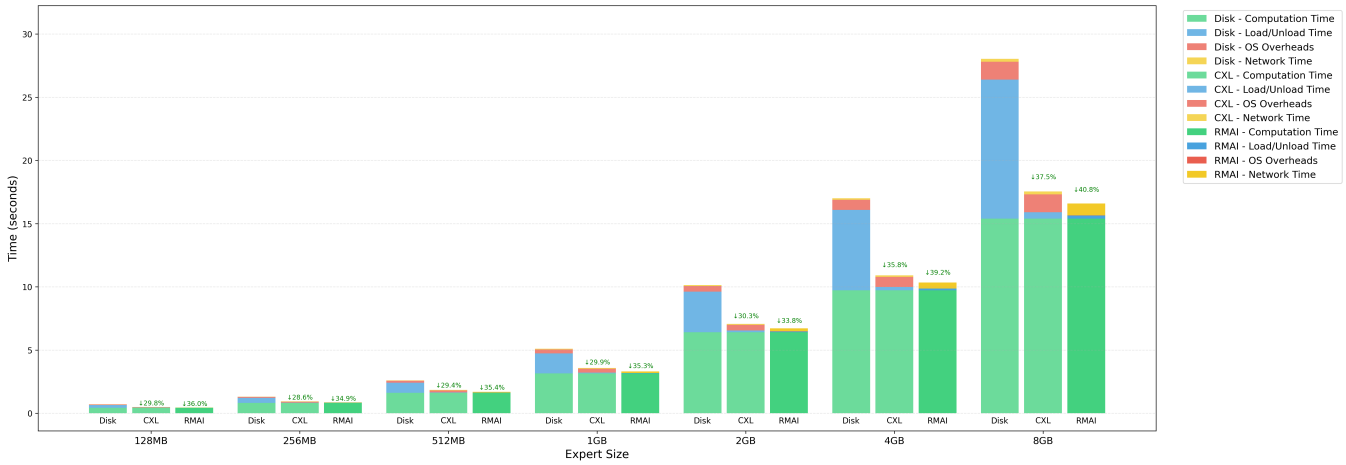

Figure 4: Comparison of execution time components across scenarios for varying expert sizes.

time is reduced to 0.51 seconds (a 95.4% improvement) because CXL memory bypasses the storage stack. However, the need for data re-registration and copying into main memory introduces residual overheads. RMAI achieves the lowest load/unload time at 0.25 seconds (a 97.7% reduction from disk-based memory), as RDMA enables direct access to remote memory, eliminating intermediate data copies and re-registration entirely.

6.3.3 Operating System Overhead. For both disk-based and CXL-based systems, the OS overhead for an 8GB expert is 1.41 seconds, by page fault handling and data mapping. Disk-based systems further suffer from storage stack management, which involves high-level filesystem operations. In contrast, RMAI reduces the OS overhead to just 0.013 seconds (a 99% reduction). This is possible because the PGAS abstraction reduces page faults significantly by treating memory as a fixed global space.

6.3.4 Network Time. Disk-based and CXL-based systems exhibit minimal network time (0.22 seconds for 8GB experts) since they rely on local storage/memory for loading experts and only use the network to transfer layer outputs in a pipelined fashion. For RMAI, the network time is higher at 0.94 seconds due to RDMA operations for remote memory access. However, this cost is predictable and scales linearly with expert size. Importantly, the reduction in load/unload time and OS overheads outweighs the additional network time, making RMAI the overall superior approach.

6.3.5 Scalability Analysis. The benefits of RMAI become increasingly apparent as expert sizes grow: For small experts (128MB–1GB), RMAI achieves a 35.9% reduction in total execution time compared to disk-based memory. This is primarily due to the significantly reduction of page fault handling, although the network overhead partially offsets the gains for smaller experts. For medium experts (1GB–4GB), RMAI delivers a 39.2% reduction in execution time, with major improvements stemming from a 93.2% reduction in load/unload time and a 98.8% reduction in OS overhead. For large experts (4GB–8GB), RMAI achieves a 40.7% reduction in execution time, enabled by the PGAS abstraction's fixed-location mapping, which eliminates memory management overheads, and the linear scaling of RDMA network costs.

7 Discussion

While our primary focus has been on CPU inference and commodity hardware, we believe RMAI could serve as a high-speed memory tier for GPU inference, particularly in scenarios where disk speed is a bottleneck[38], CXL memory is unavailable or of similar or lower performance [40, 41]. By leveraging RDMA links, RMAI enables faster expert loading, enhancing GPU performance in such environments. Additionally, certain training schemes, such as PyTorch FSDP [44], require synchronizing the model and experts on demand during each iteration. Our approach proves particularly useful in these scenarios, as it enables efficient offloading and transfer of experts while maximizing the overlap between communication and computation using VMRs, as discussed in the design section. Although our implementation was originally designed for CPUs, we can still leverage them in scenarios like fine-tuning, which falls under training and where CPUs are applicable. Therefore, we see our work as applicable to fine-tuning MoE models using paradigms like FSDP this suggests our work can also generalize beyond mixture of experts models and can be applied to LLMs in general.

In our prototype solution, we determined the optimal VMR size by conducting an experiment and running it with the calculated best size. However, in a full production-ready implementation, this process needs to be automated. One approach is to analyze access patterns to dynamically determine the optimal size, or alternatively, we could provide

System	PGAS/DSM	Symmetric	Unified	Kernel-Level	Transparent	AI/Inference
Infiniswap [20]	✗	✗	✗	✗	✓	✗
Leap [30]	✗	✗	✗	✗	✓	✗
CFM [6]	✗	✗	✗	✗	✓(sch.)	✗
GMEM [45]	✗	✗	✗	✓	✓(dev.)	✗
Hydra [25]	✗	✗	✗	✗	✓(part.)	✗
LegoOS [37]	✗	✗	✗	✓(disagg.)	✓(part.)	✗
PopcornOS [8]	✓(DSM)	✓	✓	✓	✓	✗
AIFM [14]	✗	✗	✗	✗	✗	✗
SAPS (Actor-PGAS) [33]	✓	✗	✗	✗	✗	✗
DRust [29]	✓	✗	✓	✗	✓(lang.)	✗
RMAI (Ours)	✓	✓	✓	✓	✓	✓

Table 3: Comparison of existing remote-memory and DSM/PGAS solutions. (✓= fully, ✗= not, and parenthetical notes for partial or specialized aspects.)

an API for user space to supply detailed information to our in-kernel memory manager.

Another research direction for future work is fault tolerance. In our proposed solution, each memory node holds a specific portion of experts. However, to ensure reliability in a fault-tolerant setting, we need to maintain multiple copies of experts across different memory nodes or implement a multi-owner approach. This way, if a node fails, compute nodes can still access the necessary data, ensuring continued operation without disruption.

While our experiments focused on a single compute node, RMAI's design inherently supports multiple compute nodes accessing shared memory. In principle, each compute node could process separate portions of the model (e.g Pipelining paradigm) or handle different prompts concurrently. This parallelism could further reduce inference latency by distributing expert computations across multiple machines.

8 Related Works

Infiniswap [20] and **Leap** [30] both exploit RDMA to extend a node's local memory by paging out to remote machines, reducing disk overhead. However, these systems focus on block-level paging rather than providing a global shared address space, and thus lack built-in support for high-frequency read-sharing typical in AI inference.

CFM [6] introduces a far memory-aware cluster scheduler to improve job throughput via remote memory, but it neither offers a unified address space nor addresses frequent on-demand expert switching seen in mixture-of-experts (MoE) workloads. **GMEM** [45] proposes generalized OS-managed memory for peripheral devices (e.g., GPUs), yet it does not provide a partitioned global address space (PGAS) for general-purpose CPU inference across multiple nodes.

Hydra [25] focuses on resiliency and high availability of remote memory, mainly via erasure coding and replication, rather than supporting a symmetric or unified memory abstraction for AI-specific workloads. **LegoOS** [37] similarly offers a kernel-based disaggregated architecture, but it lacks a fully transparent, symmetric PGAS that is critical for dynamic parameter sharing in MoE models. Similarly, recent multiple-kernel OSes, like Popcorn Linux [8–10, 24, 36], and distributed hypervisors [3, 4, 13, 21] do provide a symmetric and unified memory abstraction that is prone to overheads as not optimized to AI workloads.

AIFM [14] presents a high-performance user-space design for far memory; however, it does not integrate with the OS's virtual memory subsystem to create a transparent, system-wide address space. **SAPS** [33] employs an actor-based approach to PGAS, but it remains entirely in user space and requires explicit message-passing constructs, making it unsuitable for unmodified AI frameworks.

Finally, **DRust** [29] addresses fine-grained distributed shared memory through language-level ownership in Rust. While it eliminates explicit coherence checks, it requires rewriting applications in Rust and does not introduce a kernel-level, symmetric address space accessible to arbitrary processes.

By contrast, our solution—**RMAI**—combines a kernel-level PGAS with a *symmetric*, unified address space, providing seamless transparency to unmodified applications while addressing the dynamic memory demands of MoE inference workloads.

9 Conclusion

While CXL promises to solve at least the memory capacity and communication overheads problems of today distributed LLM, this paper demonstrates that at the cost of a memory optimized server and high-speed networking a MoE model can run faster than on CXL (up to 10%), and indeed faster than the baseline (up to 45%) where the experts are on SSD. This is achievable just by software innovation. That is moving part of the application software into the kernel, exploiting remote

memory, transparently, but aware of the size of each expert. While these are initial results and run without the case of sharing – we don't have any CXL switch to compare with, we hope these results will generate further research in debloating AI frameworks and moving some of their functionalities down in the software stack, like in this case.

References

[1] [n. d.]. Arm Scalable Matrix Extension (SME). https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-scalable-matrix-extension-introduction

[2] [n. d.]. Intel® Advanced Matrix Extensions (Intel® AMX). https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/advanced-matrix-extensions/overview.html

[3] [n. d.]. ScaleMP. https://wiki.metacentrum.cz/wiki/ScaleMP_(en).

[4] [n. d.]. Tidescale. https://cloud.ibm.com/catalog/content/node-red-operator-certified::2-7650c9f1-28ba-404e-a2bb-f898466ee6e1-global.

[5] Erdem Aktas and Zhimin Yao. 2024. *We tested Intel's AMX CPU accelerator for AI. Here's what we learned.* Google Cloud Blog. https://cloud.google.com/blog/products/identity-security/we-tested-intels-amx-cpu-accelerator-for-ai-heres-what-we-learned Staff Software Engineers at Google Cloud.

[6] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can far memory improve job throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 14, 16 pages. doi:10.1145/3342195.3387522

[7] Arm Limited. 2024. *A Comprehensive Guide to Understanding AI Inference on the CPU.* Technical Report. Arm Limited. arm.com Examines architectural evolution and optimizations for AI inference workloads across Arm's CPU platform ecosystem, from cloud scale deployments to edge computing applications..

[8] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. ACM, New York, NY, USA, 645–659. doi:10.1145/3037697.3037738

[9] Antonio Barbalace, Binoy Ravindran, and David Katz. 2014. Popcorn: a replicated-kernel OS based on Linux. In *Proceedings of the Linux Symposium, Ottawa, Canada.*

[10] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. 2015. Popcorn: Bridging the programmability gap in heterogeneous-isa platforms. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–16.

[11] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A Survey on Mixture of Experts. arXiv:2407.06204 [cs.LG] https://arxiv.org/abs/2407.06204

[12] Yue Cheng, Ali Anwar, and Xuejing Duan. 2018. Analyzing Alibaba's Co-located Datacenter Workloads. In *2018 IEEE International Conference on Big Data (Big Data)*. 292–297. doi:10.1109/BigData.2018.8622518

[13] Ho-Ren Chuang, Karim Manaouil, Tong Xing, Antonio Barbalace, Pierre Olivier, Balvansh Heerekar, and Binoy Ravindran. 2023. Aggregate VM: Why Reduce or Evict VM's Resources When You Can Borrow Them From Other Nodes?. In *Proceedings of the Eighteenth European Conference on Computer Systems* (Rome, Italy) *(EuroSys '23)*. Association for Computing Machinery, New York, NY, USA, 469–487. doi:10.1145/3552326.3587452

[14] Aleksandar Dragojevic, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2014)*. USENIX – Advanced Computing Systems Association. https://www.microsoft.com/en-us/research/publication/farm-fast-remote-memory/

[15] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten Bosma, Zongwei Zhou, Tao Wang, Yu Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. GLaM: Efficient Scaling of Language Models with Mixture-of-Experts. arXiv:2112.06905 [cs.CL] https://arxiv.org/abs/2112.06905

[16] Mohammad Ewais and Paul Chow. 2023. Disaggregated Memory in the Datacenter: A Survey. *IEEE Access* 11 (2023), 20688–20712. doi:10.1109/ACCESS.2023.3250407

[17] Jiarui Fang, Zilin Zhu, Shenggui Li, Hui Su, Yang Yu, Jie Zhou, and Yang You. 2023. Parallel Training of Pre-Trained Models via Chunk-Based Dynamic Memory Management. *IEEE Transactions on Parallel and Distributed Systems* 34, 1 (Jan. 2023), 304–315. doi:10.1109/tpds.2022.3219819

[18] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. arXiv:2101.03961 [cs.LG] https://arxiv.org/abs/2101.03961

[19] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 135–153. https://www.usenix.org/conference/osdi24/presentation/fu

[20] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 649–667. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[21] Xingguo Jia, Jin Zhang, Boshi Yu, Xingyue Qian, Zhengwei Qi, and Haibing Guan. 2022. GiantVM: A Novel Distributed Hypervisor for Resource Aggregation with DSM-aware Optimizations. *ACM Trans. Archit. Code Optim.* 19, 2, Article 20 (March 2022), 27 pages. doi:10.1145/3505251

[22] Chenyu Jiang, Ye Tian, Zhen Jia, Shuai Zheng, Chuan Wu, and Yida Wang. 2024. Lancet: Accelerating Mixture-of-Experts Training via Whole Graph Computation-Communication Overlapping. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 74–86. https://proceedings.mlsys.org/paper_files/paper/2024/file/339caf45a6fa281cae8adc6465343464-Paper-Conference.pdf

[23] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 463–479. https://www.usenix.org/conference/osdi20/presentation/jiang

[24] David Katz, Antonio Barbalace, Saif Ansary, Akshay Ravichandran, and Binoy Ravindran. 2015. Thread migration in a replicated-kernel os. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. IEEE, 278–287.

[25] Youngmoon Lee, Hasan Al Maruf, Mosharaf Chowdhury, Asaf Cidon, and Kang G. Shin. 2022. Hydra : Resilient and Highly Available Remote Memory. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. USENIX Association, Santa Clara, CA, 181–198.

https://www.usenix.org/conference/fast22/presentation/lee

[26] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Vancouver, BC, Canada) *(ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 574–587. doi:10.1145/3575693.3578835

[27] Jiacheng Liu, Peng Tang, Wenfeng Wang, Yuhang Ren, Xiaofeng Hou, Pheng-Ann Heng, Minyi Guo, and Chao Li. 2025. A Survey on Inference Optimization Techniques for Mixture of Experts Models. arXiv:2412.14219 [cs.LG] https://arxiv.org/abs/2412.14219

[28] Ka Man Lo, Zeyu Huang, Zihan Qiu, Zili Wang, and Jie Fu. 2024. A Closer Look into Mixture-of-Experts in Large Language Models. arXiv:2406.18219 [cs.CL] https://arxiv.org/abs/2406.18219

[29] Haoran Ma, Yifan Qiao, Shi Liu, Shan Yu, Yuanjiang Ni, Qingda Lu, Jiesheng Wu, Yiying Zhang, Miryung Kim, and Harry Xu. 2024. DRust: Language-Guided Distributed Shared Memory with Fine Granularity, Full Transparency, and Ultra Efficiency. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 97–115. https://www.usenix.org/conference/osdi24/presentation/ma-haoran

[30] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 843–857. https://www.usenix.org/conference/atc20/presentation/al-maruf

[31] George Michelogiannakis, Benjamin Klenk, Brandon Cook, Min Yee Teh, Madeleine Glick, Larry Dennison, Keren Bergman, and John Shalf. 2022. A Case For Intra-rack Resource Disaggregation in HPC. *ACM Trans. Archit. Code Optim.* 19, 2, Article 29 (March 2022), 26 pages. doi:10.1145/3514245

[32] Seonjin Na, Geonhwa Jeong, Byung Hoon Ahn, Jeffrey Young, Tushar Krishna, and Hyesoon Kim. 2024. Understanding Performance Implications of LLM Inference on CPUs . In *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE Computer Society, Los Alamitos, CA, USA, 169–180. doi:10.1109/IISWC63097.2024.00024

[33] Sri Raj Paul, Akihiro Hayashi, Kun Chen, and Vivek Sarkar. 2022. A Scalable Actor-based Programming System for PGAS Runtimes. arXiv:2107.05516 [cs.DC] https://arxiv.org/abs/2107.05516

[34] Timothy Prickett Morgan. 2023. Why AI Inference Will Remain Largely On The CPU. *The Next Platform* (5 4 2023). https://www.nextplatform.com/2023/04/05/why-ai-inference-will-remain-largely-on-the-cpu/ Sponsored analysis of Intel's AMX matrix acceleration technology for AI inference workloads.

[35] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. 2012. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing* (San Jose, California) *(SoCC '12)*. Association for Computing Machinery, New York, NY, USA, Article 7, 13 pages. doi:10.1145/2391229.2391236

[36] Marina Sadini, Antonio Barbalace, Binoy Ravindran, and Francesco Quaglia. [n. d.]. A Page Coherency Protocol for Popcorn Replicated-kernel Operating System. ([n. d.]).

[37] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 69–87. https://www.usenix.org/conference/osdi18/presentation/shan

[38] Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. 2024. HOBBIT: A Mixed Precision Expert Offloading System for Fast MoE Inference. doi:10.48550/arXiv.2411.01433

[39] Zhonghua Wang, Yixing Guo, Kai Lu, Jiguang Wan, Daohui Wang, Ting Yao, and Huatao Wu. 2024. Rcmp: Reconstructing RDMA-Based Memory Disaggregation via CXL. *ACM Trans. Archit. Code Optim.* 21, 1, Article 15 (Jan. 2024), 26 pages. doi:10.1145/3634916

[40] Tong Xing and Antonio Barbalace. 2025. Rethinking Applications' Address Space with CXL Shared Memory Pools. (2025).

[41] Tong Xing, Cong Xiong, Tianrui Wei, April Sanchez, Binoy Ravindran, Jonathan Balkind, and Antonio Barbalace. 2025. Stramash: A Fused-kernel Operating System For Cache-Coherent, Heterogeneous-ISA Platforms. (2025).

[42] An Yang, Junyang Lin, Rui Men, Chang Zhou, Le Jiang, Xianyan Jia, Ang Wang, Jie Zhang, Jiamang Wang, Yong Li, Di Zhang, Wei Lin, Lin Qu, Jingren Zhou, and Hongxia Yang. 2021. M6-T: Exploring Sparse Expert Models and Beyond. arXiv:2105.15082 [cs.LG] https://arxiv.org/abs/2105.15082

[43] Dianhai Yu, Liang Shen, Hongxiang Hao, Weibao Gong, Huachao Wu, Jiang Bian, Lirong Dai, and Haoyi Xiong. 2024. MoESys: A Distributed and Efficient Mixture-of-Experts Training and Inference System for Internet Services. *IEEE Transactions on Services Computing* 17, 5 (2024), 2626–2639. doi:10.1109/TSC.2024.3399654

[44] 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. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3848–3860. doi:10.14778/3611540.3611569

[45] Weixi Zhu, Alan L. Cox, and Scott Rixner. 2023. GMEM: Generalized Memory Management for Peripheral Devices. arXiv:2310.12554 [cs.OS] https://arxiv.org/abs/2310.12554