TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/38 pp1411–1439
DOI: 10.26599/TST.2025.9010169
Volume 31, Number 3, June 2026

Review

A Survey on Accelerated Technologies for Mixture-of-Experts Model Training Systems

Qi Zhang, Jidong Zhai*, and Weimin Zheng

Abstract: Mixture-of-Experts (MoE) models have emerged as a transformative paradigm for scaling Large Language Models (LLMs), enabling unprecedented model capacity while maintaining computational efficiency through sparse activation mechanisms. However, the unique architectural characteristics of MoE models introduce significant system-level challenges that fundamentally differ from traditional dense models. These challenges necessitate specialized system optimizations tailored to MoE's distinctive properties. This survey systematically analyzes accelerated technologies for MoE training systems, discussing recent advances across four critical optimization dimensions: hybrid parallel computing, comprehensive memory management, fine-grained communication scheduling, and adaptive load balancing. Our analysis reveals a paradigm shift from computation-centric to workload-centric optimization strategies. What's more, we identify emerging research directions including machine learning-guided load balancing, cross-layer optimization frameworks, and hardware-software co-design for MoE training workloads. This work aims to provide researchers and system engineers with a comprehensive technical reference to support the design of more efficient and scalable next-generation MoE training systems.

Key words: Mixture-of-Experts (MoE); Large Language Models (LLMs); distributed training; system optimization; parallel computing; memory management; communication scheduling; load balancing

1 Introduction

The exponential growth in Large Language Model (LLM) capabilities has been largely driven by scaling model parameters, following the scaling laws that demonstrate consistent improvements with increased model size[1]. However, this scaling approach faces fundamental limitations due to the linear relationship between model capacity and computational complexity, making training increasingly expensive and resource-intensive. Mixture-of-Experts (MoE) models[2–5] have emerged as a transformative paradigm that decouples this linear relationship, enabling unprecedented model scaling while maintaining computational efficiency through conditional computation mechanisms.

The past few years have witnessed remarkable progress in MoE model development, evolving from research prototypes such as GShard[6] and Switch Transformer[4] to production-scale deployments including Mixtral 8x7B[7], DeepSeek-V3[8], Snowflake Arctic[9], and PanGu-Ultra MoE[10]. The fundamental innovation of MoE architectures lies in two key mechanisms that dramatically reduce training costs

• Qi Zhang and Weimin Zheng are with Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: qi_zhang@tsinghua.edu.cn; zwm-dcs@tsinghua.edu.cn.
• Jidong Zhai is with Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with School of Computer Science and Engineering, Qinghai University, Xining 810000, China. E-mail: zhaijidong@tsinghua.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2025-07-27; revised: 2025-09-07; accepted: 2025-10-13

while maintaining model expressiveness. First, sparse activation ensures that only a subset of expert networks is dynamically selected to participate in computation for each input token, significantly reducing the actual computational load per training iteration. Second, expert partitioning distributes different experts across multiple computing devices, effectively overcoming single-device memory limitations and enabling models with trillions of parameters to be trained on distributed systems.

Despite these advantages, MoE architectures introduce unique system-level challenges that fundamentally differ from traditional dense models. These challenges make existing optimization techniques for dense LLMs training systems inadequate. First, cross-device communication from expert partitioning becomes a critical training bottleneck. In particular, the All-to-All communication operations required in expert parallelism implementations could consume 35%−50% of total training time[11, 12]. Second, load imbalance severely impacts computational resource efficiency, as certain experts may become heavily loaded while others remain underutilized, creating system-wide performance bottlenecks[4, 13]. Third, the sparse characteristics of MoE models make memory management substantially more complex than dense counterpart, requiring sophisticated strategies for expert parameter allocation, activation checkpointing, and dynamic memory scheduling[14, 15]. These challenges are interdependent and exhibit non-linear growth with increasing model scale and expert count, rendering simple hardware scaling solutions inadequate. Consequently, system-level optimization specifically tailored to MoE architectural characteristics has become critical for realizing the full potential of these models in large-scale training scenarios.

Given the significant interest from both academia and industry in MoE training system optimization technologies, this survey provides a systematic investigation of recent advances in this rapidly evolving field. While several surveys have addressed related topics in LLM training system optimization[16−18] and MoE model architectures and system optimization[19−22], our work distinguishes itself by providing the first comprehensive analysis specifically focused on system-level optimization techniques for MoE training scenario. Through systematic discussion of state-of-the-art works, we reveal key insights into the evolution from computation-centric to workload-centric optimization paradigms while identifying critical trade-offs between different strategies. This foundation enables us to outline promising future research directions including machine learning-guided optimization, cross-layer co-design, and hardware-software integration for MoE workloads.

Specifically, the primary contributions of this survey are threefold:

• We provide a systematic taxonomy and analysis of MoE training system optimization techniques across multiple dimensions;

• We identify key technical insights and design principles that guide effective MoE system optimization;

• We outline future research directions that will shape the next generation of scalable MoE training infrastructure.

The structure of this survey is organized as follows. We begin by providing an overview of MoE architectures in Section 2. We then discuss performance evaluation references and system-level performance optimization challenges in Section 3. Subsequently, we examine four fundamental optimization dimensions for addressing these challenges: Section 4 investigates multi-dimensional parallelism strategies, Section 5 analyzes memory optimization technologies, Section 6 studies communication optimization approaches, and Section 7 discusses load balancing optimization methods. Section 8 synthesizes design principles and best practices derived from these optimization techniques, providing practical guidance for building efficient MoE training systems. Finally, Section 9 concludes this survey.

2 MoE Architecture

The architectural evolution from dense to sparse models represents a fundamental paradigm shift in large language model design. The layer architecture of conventional dense Transformer LLMs is depicted in Fig. 1a. In dense models, each layer follows a fixed sequential processing pipeline. Input representations are first processed through the attention mechanism to capture contextual dependencies. This is followed by residual connections and layer normalization (add & norm) operations that facilitate information integration

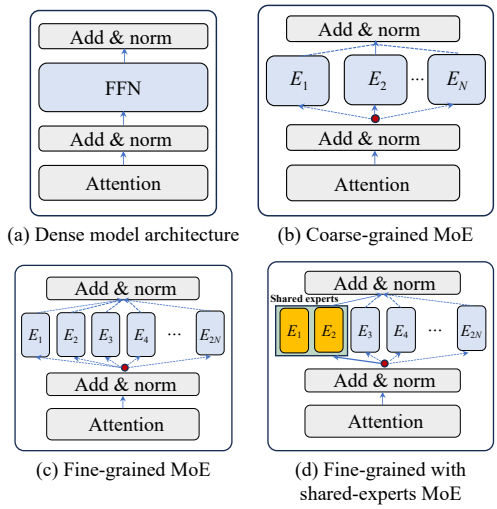

Fig. 1 Evolution of MoE architectures.

and training stabilization. Subsequently, the representations undergo nonlinear transformation via the Feed-Forward Network (FFN), which typically consists of two linear projections with an activation function. Finally, another round of residual connections and layer normalization completes the layer's computational cycle, ensuring gradient flow and representational stability throughout the network depth.

In contrast to the traditional dense architecture of LLM, sparse activation-based MoE models represent a fundamentally different computational paradigm that enables model scaling while maintaining computational efficiency. In MoE architectures, the single FFN in each Transformer layer is replaced by multiple parallel FFNs (termed as "experts"), which can be denoted as $\mathcal{E} = \{E_1, E_2, \ldots, E_N\}$. A gating network dynamically selects and activates K experts from the total N experts for each input token. Formally, the output of an MoE layer for an input token x can be expressed as

$$F(x) = \sum_{i \in \text{TopK}(G(x))} G_i(x) \cdot E_i(x) \tag{1}$$

where $G_i(x)$ represents the gating weight for expert $E_i(x)$, and TopK $(\cdot)$ denotes the indices of the K experts selected by the gating function. The evolution of MoE architectures can be categorized into three types (as illustrated in Figs. 1b–1d), each addressing different trade-offs between computational efficiency, model capacity, and specialization granularity.

2.1 Coarse-grained MoE architecture

As illustrated in Fig. 1b, coarse-grained MoE architecture represents a paradigmatic shift from dense to sparse computation. The key innovation lies in decomposing the monolithic FFN layer in traditional Transformers LLM into multiple parallel expert networks. This design transcends the constraint of "full parameter activation" and transitions toward a computational mode of "selective activation". Coarse-grained MoE models typically employ 8 to 16 experts per layer, with each expert containing millions to billions of parameters. This architecture strikes a balance between computational efficiency and implementation complexity, making it suitable for production deployments. Representative works include Mixtral 8×7B[7] and DBRX[23]. These models have demonstrated remarkable performance across various benchmarks while significantly reducing computational costs compared to equivalent dense LLMs.

2.2 Fine-grained MoE architecture

As depicted in Fig. 1c, fine-grained MoE architecture employs smaller, more numerous experts, enabling more refined knowledge partitioning and enhanced specialization compared to coarse-grained designs. This approach typically employs 64 to 256 experts per layer, with each expert being substantially smaller than those in coarse-grained designs. The increased granularity allows for more precise routing decisions and enhanced model expressiveness through specialized expert functions. Fine-grained MoE architecture has been shown to exhibit accelerated convergence rates compared to their coarse-grained counterparts, attributed to the reduced interference between different types of knowledge and more efficient gradient updates[24]. Pioneering implementations include Switch Transformer[4], GShard[6], and GLaM[5].

2.3 Fine-grained with shared-experts MoE architecture

As presented in Fig. 1d, the fine-grained with shared-experts MoE architecture combines the benefits of both always-activated shared experts and dynamically routed ones, creating a more robust and stable training paradigm. The architecture employs two distinct types of expert networks: Shared experts (highlighted in orange) that are always activated and responsible for capturing cross-domain universal patterns and fundamental linguistic representations; On the other hand, routed experts (depicted in blue) that are selectively activated to focus on domain-specific and

specialized knowledge acquisition. This design ensures that essential common knowledge is consistently accessible while still enabling fine-grained processing through dynamic routing algorithms.

Formally, the output $F(x)$ of the routing function can be expressed as

$$F(x) = F_{\text{shared}}(x) + \sum_{i \in \text{TopK}(G(x))} G_i(x) \cdot E_i(x) \tag{2}$$

where $F_{\text{shared}}(x)$ represents the contribution from shared experts that are always activated, ensuring consistent access to universal knowledge across all inputs, while the second term captures the specialized processing from dynamically selected experts. This architecture has been successfully implemented in several state-of-the-art models, including DeepSeek-V3[8], Hunyuan-Large[25], Qwen2-MoE[26], and PanGu-Ultra MoE[10]. These models have demonstrated superior performance in both general language understanding and specialized domain tasks, validating the effectiveness of the hybrid design.

3 Performance Evaluation and Challenge

While MoE architectures offer compelling advantages in scaling LLMs, they introduce a unique set of system-level challenges that fundamentally differ from traditional dense model training scenario. These challenges arise from the inherent characteristics of sparse activation, dynamic routing, and expert specialization, creating complex interdependencies that cannot be addressed through simple hardware scaling or conventional optimization techniques. Figure 2 illustrates the key system bottlenecks in distributed MoE training systems. To facilitate reader comprehension, this section first establishes the evaluation metrics and baseline for assessing the performance of MoE training systems, and then systematically analyzes the key system challenges and their implications for system design.

3.1 Evaluation metrics and benchmark

Rigorous evaluation of MoE training systems requires comprehensive methodologies. These must capture both standard distributed training metrics and unique performance dimensions introduced by sparse expert architectures. However, the absence of standardized benchmarks complicates cross-system comparisons, as different works employ varying model configurations, hardware platforms, and measurement methods. This section synthesizes the key metrics and benchmarks (reference implementation) used in MoE system evaluation.

Performance metrics: Similar to dense LLM training systems, MoE counterparts are primarily evaluated on training throughput (indicated by tokens or samples per second) and computational performance, quantified as flop operations per second (namely FLOPS), which directly determine training time and cost. However, the sparse activation and dynamic routing characteristics of MoE introduce additional evaluation dimensions that are critical in understanding system efficiency: a system achieving high raw throughput may still under-perform if communication overhead consumes significant part of

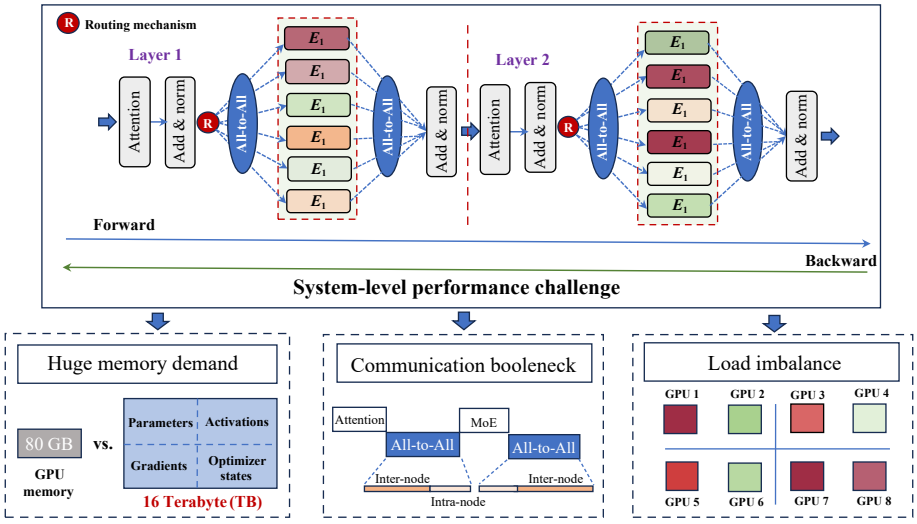

Fig. 2 System challenges in the distributed MoE training context, illustrating three key bottlenecks including huge memory demand, All-to-All communication overhead, and load imbalance across experts, along in a typical distributed training setup.

iteration time, or if load imbalance leaves Graphics Processing Units (GPUs) low utilization. Therefore, hardware efficiency metrics are essential for quantifying accelerator utilization under achieved throughput. Model FLOPS Utilization (MFU) measures the ratio of achieved FLOPS to hardware theoretical peak, while Hardware FLOPS Utilization (HFU) represents hardware efficiency when activation recomputation is applied. These two indices together reveal whether bottlenecks stem from accelerator hardware underutilization.

Beyond computational efficiency metrics, two bandwidth metrics critically impact MoE training efficiency. Network bandwidth utilization measures achieved communication throughput as a percentage of theoretical peak, while memory bandwidth utilization typically quantifies how effectively the system saturates High-Bandwidth-Memory (HBM) bandwidth of accelerators. Together, these metrics reveal system bottlenecks and guide optimization priorities: low network utilization indicates the need for communication optimizations, whereas low memory bandwidth utilization suggests opportunities for memory access pattern improvements.

Benchmarks and reference implementation: The MoE research community currently lacks fully standardized benchmarks comparable to those for dense models. For instance, MLPerf-Training[27] provides industry-standard evaluation protocols, but its primary LLM benchmarks target dense architectures. In the absence of standardized benchmarks, several systems have emerged as de facto evaluation baselines. Megatron-LM[28] establishes references for production-scale hybrid parallelism with MoE extensions. DeepSpeed-MoE[15] provides a baseline system that achieves memory optimization. FasterMoE[11] and Tutel[12] serve as primary references for communication optimization techniques in MoE training systems. Comparative evaluations typically measure performance improvements relative to these baselines under controlled conditions—identical model configurations, expert counts, hardware platforms, and datasets—to isolate the impact of specific optimization techniques. Despite the utility of these established baselines, comprehensive standardized evaluation protocols remain necessary to enable fair assessment of both system performance and across the diverse MoE training landscape.

3.2 System performance challenges

3.2.1 Huge memory demand

MoE models exhibit substantially larger memory requirements compared to dense models of equivalent computational complexity. While a dense transformer model with similar flop operations might require tens of gigabytes of memory, MoE models with hundreds of experts can demand hundreds of gigabytes to several terabytes of memory for parameter storage alone. For instance, a 1-trillion-parameter MoE model trained under Brain Floating point 16-bit (BF16) mixed-precision requires approximately 16 terabytes (TB) of memory to accommodate the model parameters, gradients, activations, and optimizer states. This massive scale pushes beyond the memory capacity of even the most advanced GPU clusters, necessitating sophisticated memory management strategies that span multiple memory hierarchies.

Beyond sheer capacity requirements, the MoE training workload introduces fundamental management complexity problems. First, memory fragmentation occurs due to the sparse distribution of expert parameters across different devices and memory segments, leading to inefficient memory utilization and increased allocation overhead. Second, additional routing metadata and gating network states must be maintained for each layer, including expert selection histories, load balancing statistics, and routing probabilities. Third, dynamic activation patterns complicate memory locality optimization, as the set of active experts varies unpredictably across different inputs and training iterations.

The technical complexity stems from the inherent dynamism in MoE training systems. Since expert token allocation varies continuously during training, traditional static memory allocation strategies prove inadequate. On the other hand, dynamic memory allocation schemes can improve memory utilization by 15%−20% compared to static approaches but incur additional runtime overhead due to frequent allocation and deallocation operations[28, 29]. Furthermore, the memory hierarchy considerations become more complex due to the need to efficiently manage expert parameters distributed across different memory tiers depending on activation patterns and memory constraints[14].

3.2.2 Communication bottleneck

Expert parallelism across multiple accelerators

introduces severe communication bottlenecks centered around All-to-All communication patterns, which constitute the most critical system constraint. Unlike the predictable communication patterns in dense model training, MoE systems exhibit dynamic, data-dependent communication requirements that create significant optimization challenges. Quantitative analysis reveals that in distributed MoE training systems, All-to-All communication operations consume 35%−50% of per-iteration time in typical multi-node configurations[7, 11]. Moreover, All-to-All communication exhibits extremely high latency variability due to the unpredictable nature of token routing decisions, severely compromising training system predictability and convergence stability.

The fundamental cause of communication bottlenecks lies in MoE's unique data-dependent communication patterns. Since token routing decisions are determined dynamically at runtime, the resulting communication exhibits several characteristics: First, communication patterns are difficult to predict and optimize in advance, preventing effective pre-computation and scheduling. Second, the dynamic nature of routing leads to highly imbalanced data exchange between devices, creating hotspots and underutilized links. Third, the need for global synchronization across all participating devices increases synchronization barriers and reduces pipeline efficiency. What's more, the severity of communication bottlenecks is highly dependent on hardware topology and interconnect characteristics. Within nodes connected by high-bandwidth, low-latency NVLink connections, All-to-All communication efficiency can reach 70%−80% of theoretical peak bandwidth. However, with cross-node communications over InfiniBand or Ethernet connections, this efficiency drops dramatically to 20%−35%[12, 30]. This efficiency disparity necessitates topology-aware optimization strategies that adapt communication patterns to specific network architectures and bandwidth hierarchies.

3.2.3 Load imbalance

Load imbalance represents the most challenging system performance issue in the MoE training context, stemming directly from the fundamental architectural design principles of expert specialization and dynamic token routing. This challenge is inherently difficult to resolve because perfect load balancing fundamentally conflicts with the core premise of MoE architectures—enabling experts to specialize on different types of input patterns and linguistic phenomena.

Load imbalance manifests in two critical system-level problems that significantly impact training efficiency: First, resource underutilization occurs when certain experts or accelerators remain in idle or low-utilization states for extended periods, leading to poor hardware efficiency and wasted computational resources. Second, performance bottlenecks emerge when heavily loaded experts create system-wide throughput limitations, as the overall training speed is constrained by the slowest processing unit.

The complexity of load balancing in MoE systems is further compounded by its dynamic nature. The degree of load imbalance evolves continuously throughout the training process as experts progressively specialize on different types of inputs, making their utilization patterns increasingly heterogeneous. This dynamic evolution renders static load balancing strategies inadequate and necessitates adaptive mechanisms capable of real-time adjustment based on current routing patterns, expert utilization statistics, and training phase characteristics[13, 31].

4 Parallelism Optimization

The exponential scale expansion of model size with MoE architectures renders traditional a single-accelerator or single-node training system fundamentally insufficient to accommodate the LLM. Large-scale distributed MoE training necessitates efficient parallelism strategies encompassing multiple dimensions, including data parallelism for batch distribution across devices, tensor parallelism for intra-layer weight partitioning, and expert parallelism for expert distribution across accelerators[15, 28, 29]. Furthermore, to fully exploit the computational potential of multi-node, multi-accelerator systems, hybrid parallelism strategies have become essential in contemporary MoE training frameworks.

This section examines multi-dimensional parallelism strategies for training MoE models at scale. We begin by introducing fundamental parallelism paradigms and their unique characteristics in MoE contexts. We then analyze three evolutionary stages of hybrid parallelism: homogeneous approaches that apply uniform strategies across all components; heterogeneous methods that exploit computational diversity between attention and expert layers; and dynamic techniques that adapt

parallelism strategies at runtime based on workload characteristics. Through this analysis, we reveal the paradigm shift from treating parallelization as a static configuration problem to viewing it as a dynamic optimization challenge that requires workload-aware adaptation.

4.1 Fundamental parallelism

Data Parallelism (DP): As the most fundamental and widely adopted parallelism strategy in deep learning training frameworks, DP uniformly distributes training data across multiple computing devices along the batch dimension[32]. Each DP group maintains a complete model replica, independently processes its assigned data samples, and synchronizes gradients through All-Reduce collective communication to update model parameters after completing forward and backward propagation for each batch samples[33, 34]. By processing multiple batches in parallel across devices, DP effectively improves aggregate training throughput, though at the cost of increased memory consumption due to parameter replication. However, DP suffers from memory redundancy as devices in each DP group store the complete model parameters, making it less suitable for accelerators with limited memory capacity.

Tensor Parallelism (TP): By partitioning large matrix operations at the operator level, TP addresses the fundamental challenge of single-device memory limitations when accommodating large model layers[28, 29]. Typical TP implementations split tensors along either row or column dimension, employing collective communications such as All-Reduce to aggregate computational results during both forward and backward propagation[35]. While TP effectively alleviates memory constraints for training large MoE models, it incurs significant communication overhead due to frequent synchronization requirements in each layer. Consequently, TP is typically constrained to intra-node deployment to avoid prohibitive inter-node communication costs.

Pipeline Parallelism (PP): PP divides deep neural networks into multiple consecutive stages by layers, with different stages allocated to distinct devices for execution, thereby forming a computational pipeline[36–38]. Inter-stage data communication utilizes Point-to-Point (P2P) transfers, which typically incur lower overhead compared to collective communication primitives. This characteristic makes PP particularly valuable for resource-constrained platforms or scenarios with limited inter-device bandwidth. However, PP introduces pipeline bubbles during forward and backward training processes, leading to device underutilization and reduced computational efficiency, especially at pipeline boundaries[39].

Sequence Parallelism (SP)/Context Parallelism (CP): Both approaches are specifically designed to address computational and memory inefficiencies when training large models on long-sequence data. SP partitions input samples along the sequence dimension, primarily reducing memory consumption of long-sequence activations in non-attention layers such as LayerNorm, Dropout, and residual connections[29, 40]. In contrast, CP partitions key-value pairs in the attention mechanism along the sequence dimension, thereby optimizing memory usage of attention matrices[40]. Furthermore, CP enables overlap of communication and computation operations, improving overall training efficiency[41].

Expert Parallelism (EP): Since expert matrices in FFN layers constitute the primary component of model size and consume substantial memory footprint, EP is specifically designed for MoE architectures to partition different subsets of experts in each layer across multiple accelerators[4, 13]. Due to the dynamic routing mechanism that maps training tokens to experts in each layer, EP is inherently coupled with the All-to-All communication to facilitate token redistribution[12, 15]. Through this approach, each EP group maintains parameters for only a subset of experts, significantly reducing per-device memory overhead. In mainstream MoE training systems, EP is usually combined with DP by executing attention layers along the DP dimension while processing FFN layers along the EP dimension.

Expert Tensor Parallelism (ETP): ETP extends beyond traditional EP by further partitioning individual expert matrices, addressing scenarios where even single experts exceed device memory capacity[35, 42]. While conventional TP partitions attention layers and expert layers in a uniform manner, ETP differs from TP by providing specific tensor partition capability only on expert layers. This approach is particularly beneficial for managing “hot experts” that receive disproportionately high token assignments and require substantial memory to accommodate their computational workload[43]. However, ETP introduces additional communication overhead due to increased synchronization requirements for expert-level tensor

operations, necessitating careful trade-offs between memory efficiency and communication costs.

4.2 Hybrid parallelism

While each of the parallelism strategies discussed above offers distinct advantages, naively combining them into a hybrid approach for training MoE models often leads to significant inefficiencies. The fundamental challenge stems from the inherent incompatibilities between these strategies and the unique architectural and workload characteristics of MoE models. For instance, traditional pipeline parallelism, which was designed for uniform computation patterns, suffers from excessive bubble time when confronted with MoE's irregular and unpredictable workloads. Additionally, under Fully Sharded Data Parallel (FSDP), the distributed sharding of model parameters significantly amplifies the All-Gather communication overhead required for collecting expert parameters, making it substantially more expensive than in dense model training.

MoE models exhibit three distinctive characteristics that demand hybrid solutions: (1) computational heterogeneity between attention and expert layers, (2) dynamic workload variation across training iterations due to routing evolution, and (3) parameter scaling that requires distributing massive models across devices while maintaining sparse activation patterns[6, 15]. Overcoming these challenges and designing efficient multi-dimensional hybrid parallelism strategies has enabled MoE models with hundreds of billions to trillions of parameters, such as Google's 600-billion-parameter GShard[6] and Microsoft's 1.7-trillion-parameter models[15] to be successfully trained and deployed in production environments.

The evolution of hybrid parallelism for MoE models can be classified into three categories, each addressing different aspects of the these challenges: (1) homogeneous strategies that optimize parallelism approaches by considering both workload and topology characteristics; (2) heterogeneous approaches that exploit fine-grained intra-layer characteristics; and (3) dynamic parallelism techniques that adapt to runtime conditions. We outline related works in Fig. 3.

4.2.1 Homogeneous hybrid parallelism

Early-stage MoE parallel training systems applied uniform combinations of parallelism strategies across all model layers. This design treats the entire MoE model as a single computational unit, applying the same parallelism configuration throughout the network. FastMoE[44] exemplifies this approach, providing a unified framework where all layers use the same DP+EP configuration. DeepSpeed-MoE[15] extends the homogeneous paradigm by introducing TP alongside DP and EP, creating a three-dimensional parallelism space. This system maintains uniformity across layers while providing additional flexibility in resource allocation.

Recognizing that large-scale distributed systems exhibit significant differences in communication bandwidth between intra-node and inter-node connections, researchers have developed **topology-aware parallelism strategies**. For example, based on communication topology characteristics, FasterMoE[11] introduces a distributed roofline modeling approach that considers both computation and communication

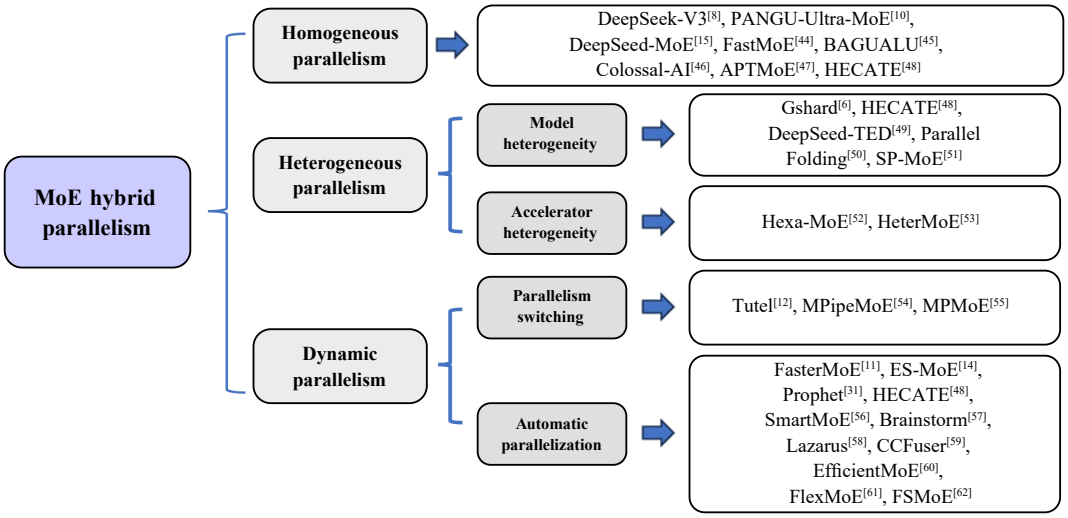

Fig. 3 MoE hybrid parallelism techniques taxonomy in MoE training systems.

characteristics to optimize hybrid parallel configurations. Colossal-AI[63] proposes regulating more experts within nodes to encourage intra-node communication, thereby reducing the overhead of inter-node communication. BaGuaLu[45] designs architecture-specific optimizations for the Sunway Supercomputer, successfully training a massive MoE model with 174 trillion parameters through tailored data and expert parallel strategies. APTMoE[47] proposes efficient MoE training schemes on resource-constrained commercial accelerators by combining pipeline parallelism with affinity-aware offloading.

These efforts demonstrate that substantial improvements in training efficiency can be achieved by optimizing parallelism combinations and considering hardware characteristics. However, the fundamental limitation of homogeneous approaches lies in their inability to exploit the distinct computational characteristics of different components in each layer. Attention module, with their dense and regular computation patterns, have different optimal parallelism strategies compared to MoE component with sparse, irregular patterns. This mismatch leads to suboptimal resource utilization, where either the attention part is over-parallelized (leading to communication overhead) or the expert module are under-parallelized (leading to memory pressure).

4.2.2 Heterogeneous hybrid parallelism

Recognizing that attention and expert components have fundamentally different computational characteristics, heterogeneous parallelism strategies have been developed to improve MoE training efficiency. GShard[6] pioneers this approach by applying data parallelism to attention parts while introducing expert parallelism for MoE ones. This division allows each component to use its most efficient parallelism strategy while maintaining overall model consistency.

DeepSpeed-TED[49] introduces a hierarchical view of parallelism. Instead of viewing different parallelism dimensions as competing alternatives, this approach treats them as complementary techniques that can be composed hierarchically. Recognizing that MoE and non-MoE parameters have different memory access patterns and computational characteristics, Colossal-AI proposed EZ-MoE[63], which introduces a different perspective on heterogeneous parallelism by combining Zero Redundancy Optimizer (ZeRO) powered data parallelism with expert parallelism. For MoE modules, the system creates multiple copies within devices to reduce cross-device communication, while non-MoE modules use global ZeRO parallelism for parameter efficiency.

In Megatron-Core[64, 65], MoE parallel folding[50] has been introduced to separate the parallelism strategies between attention and MoE layers. Attention modules can configure parallelism combinations from TP, CP, DP, and PP dimensions, while MoE modules adopt TP, DP, EP, and PP combinations. Furthermore, the EP×TP group in MoE sub-layers becomes a subgroup of the DP×CP×TP group in attention layers. This hierarchical relationship reduces the minimum GPU requirements from CP×EP to max (CP, EP), enabling efficient resource utilization while maintaining high communication locality. Recent work on FSSDP[48] achieves expert parallelism on top of FSDP by fully sharding MoE parameters and optimizer states across devices. Moreover, FSSDP materializes individual experts on demand through two sparse collectives (SparseAllGather and SparseReduceScatter), eliminating the need to store complete model parameters. This sparse materialization strategy achieves up to 3.54× speedup over state-of-the-art MoE training systems while maintaining memory efficiency.

Heterogeneous accelerator optimization: Modern clusters often consist of heterogeneous accelerators with different computational capabilities and memory capacities to reduce infrastructure investment costs. Several approaches have emerged to effectively leverage this heterogeneity in MoE systems. HeterMoE[53] exploits hardware heterogeneity by recognizing that attention and expert computations have different hardware sensitivities. The system assigns attention computation to newer, more capable GPUs while placing expert computation on older hardware. This hardware-aware approach extends beyond simple workload assignment to include sophisticated memory management and scheduling strategies that account for the different capabilities of heterogeneous hardware.

HEXA-MoE[66] takes a complementary approach by proposing a heterogeneous-aware expert allocation framework that redesigns MoE computation through expert-specific operators. Rather than relying on traditional general matrix multiplication (namely GEMM) that require token padding and lead to computation redundancy, HEXA-MoE introduces three specialized operators: Expert-Specific Matrix Multiplication (ESMM), Expert-Specific Summation

(ESS), and Expert-Specific Transposed Matrix Multiplication (ESTMM). What's more, it enables workload distribution across heterogeneous devices by dynamically adjusting local batch sizes in data-centric configurations or sub-dimensions of FFN intermediate sizes in model-centric configurations based on each device's computational capacity.

Heterogeneous hybrid parallelism can also be applied to different training phases, as the forward and backward passes in MoE training have fundamentally different computational and memory characteristics, motivating phase-specific parallelism strategies[62]. Therefore, optimal parallelism strategies should be designed not only in an architecture-aware manner but also in a phase-aware way, considering both the heterogeneous nature of modern clusters and the diverse computational requirements across different training phases.

4.2.3 Dynamic hybrid parallelism

The inherent dynamism of MoE training suggests that the most efficient parallelism strategy at training onset may become suboptimal as the model learns and expert utilization patterns stabilize. Dynamic hybrid parallelization techniques overcome these issues by enabling runtime adaptation of parallelism strategies, allowing systems to continuously optimize resource allocation based on evolving workload characteristics.

A critical enabler of dynamic parallelism is the design of data structures that can efficiently support multiple parallelism strategies without incurring prohibitive transition costs. For instance, Tutel[12] proposes a unified data layouts that maintain compatibility across different parallelism dimensions. The system's data layout abstraction decouples the physical data organization from the logical parallelism strategy, enabling transparent switching between DP, EP, and hybrid configurations without requiring expensive data redistribution operations.

MPipeMoE[54] recognizes that traditional static pipeline parallelism fails to accommodate the dynamic memory requirements of MoE models, where expert activation patterns create varying memory footprints across pipeline stages. Based on this observation, MPipeMoE proposed an adaptive pipeline depth adjustment mechanism, which dynamically modifies the number of pipeline stages based on real-time memory utilization and expert load distribution. MPMoE[55] extends the concept of adaptive pipeline parallelism by introducing proactive memory management techniques that anticipate future memory requirements based on expert activation trends. The system employs predictive algorithms to forecast expert utilization patterns and preemptively adjusts parallelism strategies to prevent memory bottlenecks before they occur.

Automatic parallelization: The complexity of selecting optimal parallelism strategies for efficient training of large language models has motivated researchers to develop automated systems that can optimize parallelism without human intervention. Alpa[67], Aceso[68], nnScaler[69], and Mist[70] demonstrate how to automatically generate optimal parallelism strategies for dense LLMs by modeling the problem as hierarchical optimization problems.

In MoE systems, expert placement is regarded as a new dimension for automatic parallelization[56], which has motivated numerous adaptive expert placement approaches, including FasterMoE[11], Prophet[31], SmartMoE[56], Lazarus[58], and CCFuser[59]. **Note that** while these techniques fundamentally constitute dynamic parallelism—as they adapt parallelism strategies based on runtime workload characteristics—we discuss this body of work in more detail in the load balancing optimization section (Section 7), here, we outline the characteristic comparison of these works in Table 1.

4.3 Technical insights and research directions

The evolution of MoE parallelism strategies—from homogeneous through heterogeneous to dynamic approaches—reveals a paradigm shift in distributed MoE training system design: from computation-centric to workload-centric optimization. This transition exposes several critical insights that will shape the future of large-scale MoE training systems.

The principle of computational locality suggests that optimal parallelism must be co-designed with the intrinsic computational characteristics of each system component[71]. The success of heterogeneous approaches like MoE parallel folding demonstrates that treating different layer types uniformly is fundamentally suboptimal. However, current implementations still operate at coarse granularities—layer types, training phases, or expert groups. The next frontier lies in fine-grained adaptive parallelism that can dynamically adjust strategies at the granularity of individual operations or even tensor slices, based on real-time computational signatures[72].

Table 1 Comparative analysis of dynamic expert scheduling technologies in MoE training systems (√: Supported, ×: Not supported).

Work	Parallel optization	Fine-grained schedule	Data locality	Fault-tolerant	Predictive method	Topology aware	Core technical innovation
FasterMoE[11]	√	×	×	×	√	√	Introduces distributed roofline modeling that jointly optimizes expert shadowing decisions with hardware topology characteristics, enabling predictive load balancing.
ES-MoE[14]	×	√	×	×	√	×	Enables training beyond GPU memory limits through expert-granular offloading combined with sequential expert computation to hide transfer latency.
Prophet[31]	√	√	×	×	√	√	Pioneers selective expert replication strategy across worker subsets rather than globally, reducing All-to-All volume while maintaining load balance.
FSSDP[48]	√	×	×	×	×	√	Extends FSDP with sparse materialization, where experts are materialized on-demand from fully sharded state using novel sparse collectives (SparseAllGather/ReduceScatter).
SmartMoE[56]	√	√	√	×	√	√	Decomposes optimization into offline pool construction and online selection phases, pre-computing Pareto-optimal parallelism strategies to enable rapid runtime adaptation.
Brainstorm[57]	×	√	×	×	√	×	Develops dynamic execution scheduling that accommodates irregular computation patterns characteristic of neural architecture search and dynamic LLMs.
Lazarus[58]	√	√	×	√	×	√	Provides provably optimal expert replica placement under node failure constraints through formal optimization framework.
CCFuser[59]	×	√	√	×	√	×	Exploits inter-GPU shared memory to eliminate data movement during expert rebalancing, fusing communication and computation through seamless zero-copy transfers.
EfficientMoE[60]	√	√	√	×	√	×	Implements adaptive load balancing with real-time workload monitoring and dynamic capacity adjustment, optimizing expert assignment based on observed routing patterns.
FlexMoE[61]	√	√	√	×	√	√	Introduces dynamic expert replication that tracks popularity evolution across training phases, scaling replica count adaptively as expert specialization patterns emerge.
FSMoE[62]	√	√	×	×	√	√	Recognizes forward/backward phase heterogeneity, applying distinct parallelism strategies per training phase to match computational characteristics.

The tension between adaptability and predictability reveals a deeper challenge in system design. While dynamic systems like Tutel[12] and SmartMoE[56] demonstrate significant performance improvements through runtime adaptation, they introduce non-deterministic behavior that complicates debugging, performance analysis, and reproducibility. The field needs to develop principled dynamic optimization approaches that maintain system predictability while enabling adaptive behavior.

On the other hand, the increasing complexity of hybrid parallelism strategies highlights the critical need for holistic system-hardware co-design. Current approaches often treat hardware topology as a constraint rather than a design parameter. The most promising future direction involves parallelism-aware hardware architectures that can efficiently support the complex communication patterns induced by heterogeneous and dynamic parallelism strategies. This includes developing specialized interconnect

topologies, adaptive bandwidth allocation mechanisms, and hardware-accelerated collective operations that can seamlessly handle the irregular communication patterns characteristic of MoE workloads.

5 Memory Optimization

Large-scale MoE model training presents unprecedented memory challenges that fundamentally differ from those in dense models. The massive memory consumption not only constrains batch sizes but also reduces accelerator utilization, creating a fundamental barrier to achieving optimal training performance[14]. While parallelism strategies effectively address some computational distribution challenges, they do not directly optimize memory usage during training and often introduce additional memory overhead through parameter replication and communication buffers.

This section provides a comprehensive analysis of memory optimization techniques specifically designed for MoE training systems. We organize these techniques into four major categories that address different aspects of the memory performance challenge: redundancy optimization techniques, activation recomputation strategies, expert parameter offloading approaches, and low-precision training methods. Figure 4 illustrates the taxonomy of these memory optimization approaches and their relationships.

5.1 Redundancy optimization

Significant parameter redundancy in distributed MoE training is introduced by parallelism strategies. While parallelization is essential for scaling MoE models across multiple devices, it often comes with the cost of increased memory demand due to parameter redundancy across model replicas, gradients, and optimizer states. Additionally, during the training process, continuous memory allocation for temporary data and layer activations contributes to substantial memory overhead. Redundancy optimization techniques address these challenges by detecting and reducing parameter replicas while eliminating unnecessary temporary memory allocations.

5.1.1 Parameter redundancy elimination

Data parallelism, while serving as one of the most effective strategies for improving training throughput, introduces significant parameter redundancy across data parallelism groups. Each device in a data parallel group maintains specific copies of model parameters, gradients, and optimizer states, leading to substantial memory footprint. For MoE models, this memory-throughput trade-off exacerbates memory demands due to their massive parameter counts, making efficient redundancy elimination crucial for scalability.

ZeRO[34], developed within the DeepSpeed training framework[82], represents a foundational technique for eliminating parameter redundancy in distributed training. The core innovation of ZeRO lies in its graduated three-stage approach: ZeRO-1 partitions optimizer states across devices; ZeRO-2 extends this to gradient partitioning; and ZeRO-3 further partitions model parameters themselves.

This hierarchical partitioning ensures that devices in the same data parallel group store only parameter shards rather than complete parameters, reducing memory requirements to approximately $1/M$ of the original footprint, where M represents the number of accelerators in the group.

The impact of optimizer state redundancy is

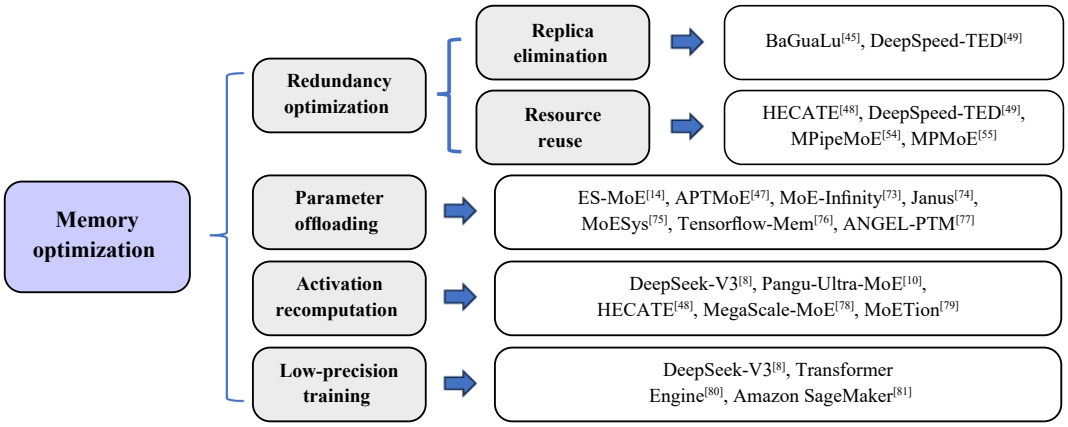

Fig. 4 Memory optimization techniques taxonomy in MoE training systems.

particularly significant in MoE training, as optimizers like Adam[83] maintain momentum and variance estimates that constitute a substantial portion of total memory consumption. Therefore, ZeRO-1 has been widely adopted in MoE training systems, including DeepSpeed-TED[49] and Llama-3 training framework[41], to alleviate memory pressure. Similarly, PyTorch's FSDP[84] implements a redundancy elimination scheme that allows for finer-grained parameter slicing, enabling users to apply parameter sharding selectively to different model components. For high-performance computing environments, BaGuaLu[45] introduces a topology-aware distributed optimizer specifically designed for the Sunway supercomputer's 96 000-node hierarchical heterogeneous network. This system substantially reduces optimizer state overhead to less than 5% of total memory footprint, enabling the training of 174-trillion-parameter MoE models through intelligent memory distribution that accounts for network topology constraints.

5.1.2 Temporal memory reuse

Since tensors in different layers exhibit non-overlap lifetimes during forward and backward propagation, temporal memory sharing presents significant opportunities for reducing memory footprint. This temporal locality can be exploited through sophisticated memory reuse strategies that allow multiple operations to share the same memory buffers when their execution windows do not overlap.

DeepSpeed-TED[49] implements a tile-based method that partitions optimizer states into tiles, enabling the reuse of temporary buffers across different computational phases. This approach reduces memory fragmentation by approximately 40% while maintaining computational efficiency. HECATE[48] extends this concept by designing expert-level memory reuse mechanisms that shard MoE layers at expert granularity, enabling cross-layer memory reuse for materializing expert parameters on demand.

MPipeMoE[54] addresses temporal memory fluctuations in pipeline parallelism through a comprehensive three-pronged memory reuse strategy: (1) Reusing activation memory across pipeline stages based on temporal dependencies; (2) Sharing gradient computation buffers across non-overlapping pipeline stages; and (3) Implementing intelligent cycling strategies for expert parameters based on pipeline scheduling patterns. Building upon MPipeMoE, MPMoE[55] proposes performance model-directed memory configurations that optimize pipeline parallelism and memory reuse strategies jointly.

Above redundancy elimination techniques substantially reduce memory fragmentation and slash peak memory requirements. Although these methods introduce additional communication overhead and scheduling complexity, they enable training of large-scale MoE models by fundamentally improving the trade-off between memory constraints and computational scalability.

5.2 Expert parameter offloading optimization

For MoE models with massive expert parameters, temporarily relocating unused model parameters from GPU memory to slower storage media—such as Dynamic Random Access Memory (DRAM) or Non-Volatile Memory express (NVMe) Solid State Drive (SSDs), as shown in Fig. 5, can effectively overcome GPU memory limitations. When these parameters are required during forward and backward passes, they are swapped into GPU Memory such as HBM. This on-demand parameter scheduling approach has been actively studied for dense LLMs[85–90], with ZeRO-Infinity[88] achieving over 6× model scaling through sophisticated prefetching and overlapping techniques that fully utilize NVMe bandwidth.

However, MoE architectures present unique challenges that render traditional offloading approaches inadequate. The parameter and optimizer state requirements in MoE models are orders of magnitude larger than dense models, while the available bandwidth between CPU and GPU remains constrained at 32 GB/s for Peripheral Component Interconnect express (PCIe) 4.0. Consequently, traditional model-level or layer-level offloading strategies become prohibitively expensive in MoE training systems, creating communication bottlenecks that negate the computational efficiency gains that MoE architectures are designed to provide.

To address these limitations, expert-granular offloading strategies represent a paradigm shift toward fine-grained memory management approaches that align with MoE's natural computational boundaries. ES-MoE[14] pioneers this approach by implementing parameter and optimizer state offloading at the expert granularity, combined with sequential expert computation rather than traditional batched processing. This design enables the system to accommodate 67×

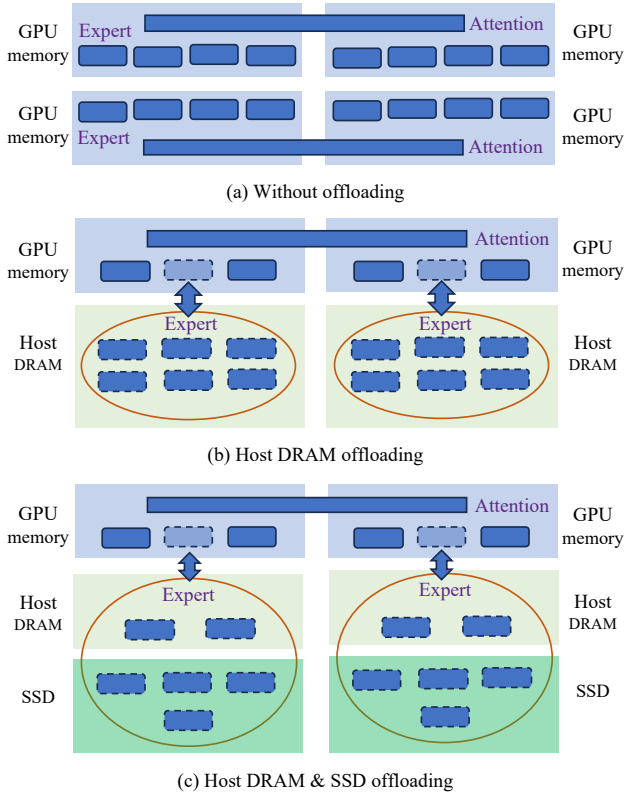

Fig. 5 Expert parameter offloading strategies. (a) Traditional GPU-only storage, (b) host DRAM offloading for inactive experts, and (c) hierarchical offloading utilizing both host DRAM and SSD storage.

more experts while achieving a 17.5× improvement in training throughput. The effectiveness of expert-level offloading stems from its alignment of memory management granularity with MoE's computational structure, allowing fine-grained control over which parameters reside in fast memory based on immediate computational requirements.

Activation-aware offloading: The breakthrough in efficient MoE offloading emerged from addressing a fundamental limitation in existing memory management approaches: the inability to predict expert activation patterns before they occur. Traditional offloading strategies treat expert selection as essentially random, leading to reactive loading schemes that introduce substantial latency penalties whenever experts must be transferred from slower memory tiers. This reactive approach becomes particularly problematic in MoE architectures where the majority of parameters reside in experts, yet dynamic routing decisions make it impossible to determine which experts will be needed until computation begins.

MoE-Infinity[73] addresses this challenge by exploiting temporal locality in expert activation patterns. Despite dynamic routing, the system leverages recurring activation patterns for intelligent memory management. The system's core innovation lies in its Expert Activation Matrix Collection (EAMC) framework, which systematically analyzes expert activation sequences across diverse input distributions to identify recurring patterns that enable predictive prefetching. By recognizing that certain experts consistently activate together for specific linguistic or semantic constructs, the system can anticipate future expert requirements based on recent activation history, effectively transforming reactive memory management into proactive optimization. Janus[74] proposes a data-centric MoE training framework where experts are transferred between different workers on-demand, implementing sophisticated individual expert swapping that coordinates CPU-GPU transfers with computation schedules to minimize idle time and maximize utilization efficiency.

Hierarchical memory management: Recognizing that modern training systems possess heterogeneous communication capabilities with different bandwidth characteristics, advanced offloading strategies simultaneously exploit multiple communication pathways to minimize offloading overhead. MoESys[75] exemplifies this approach through a hybrid storage strategy. It applies ZeRO-3-like distributed storage to dense model components, leveraging high-bandwidth NVLink connections for parameter prefetching. Meanwhile, sparse expert parameters are offloaded to DRAM and SSD storage. This hierarchical approach demonstrates that effective MoE memory management requires not only innovative algorithms but also careful consideration of underlying hardware topology and communication infrastructure to achieve optimal performance.

5.3 Activation recomputation strategy

In large-scale model training, the storage of intermediate activations often dominate memory usage. Activation recomputation, also known as gradient checkpointing, addresses this challenge through a computation-for-storage trade-off: intermediate activations are discarded during forward propagation and recomputed when needed during backpropagation[29, 91]. This approach can significantly reduce peak memory usage, enabling training of deeper or wider models under memory constraints. For MoE

training specifically, activation recomputation has been applied to mitigate out-of-memory issues caused by severe load imbalance during early training stages[41].

Traditional activation recomputation strategies, while effective at reducing memory pressure, suffer from execution within the critical training path, creating scenarios where memory savings come at the expense of training throughput. This limitation often results in negative performance impacts that discourage adoption in production environments. Recent advances in MoE-specific activation recomputation have addressed these limitations through intelligent selectivity and precision-aware strategies. MegaScale-MoE[78] implements selective recomputation that retains activations costly to recompute while selectively recomputing those generated by memory-intensive operations, eliminating traditional performance penalties associated with activation checkpointing. DeepSeek-V3[8] integrates FP8 mixed-precision storage with selective recomputation strategies, demonstrating how activation recomputation can be enhanced through careful consideration of numerical precision requirements to enable effective memory reduction without compromising training stability.

PanGu-Ultra-MoE[10] implements fine-grained recomputation operating at individual operator level rather than traditional layer-wise strategy. It introduces Multi-head Latent Attention (MLA) Key-Value (KV)-only recomputation, where only key-value vectors in attention modules are recomputed while other activations are selectively stored or recomputed based on computational cost and memory footprint. HECATE[48] reconceptualizes activation management through sparse materialization strategies where only activations corresponding to selected experts are computed and stored, with full activation patterns reconstructed from sparse representations during backpropagation.

The sparse activation characteristics of MoE models present unique opportunities and challenges for activation recomputation. Since each token passes through only selected experts, inactive components do not require recomputation, enabling more efficient memory-computation trade-offs. MoETion[79] addresses this through a "sparse checkpoint" mechanism that selectively saves and recomputes activations based on MoE's routing sparsity. By analyzing which expert outputs are most critical for memory usage and gradient computation, the system checkpoints only a small portion of "low-reuse" activations, significantly reducing overall recomputation overhead. This demonstrates that for dynamic computation graphs like MoE, customizing activation checkpoint strategies with routing information can achieve superior memory-computation trade-offs.

5.4 Low-precision training

Low-precision training has emerged as a powerful optimization strategy that enables training of massive MoE models within constrained memory budgets while maintaining competitive performance. Modern accelerators such as NVIDIA H100 GPUs support 8-bit Float Point (FP8) calculations, making FP8 low-precision training particularly attractive since FP8 GEMMs in forward propagation, weight gradients, and activation gradients effectively halve memory requirements compared to BF16 precision.

Several training frameworks and acceleration engines support FP8 training with flexible precision switching capabilities, including Microsoft's FP8-LM[92], NVIDIA's Transformer Engine[80], and Amazon SageMaker[81]. DeepSeek V3[8] demonstrates the practical effectiveness of FP8 mixed-precision training for a 671-billion-parameter MoE model, achieving competitive performance while significantly reducing memory requirements. However, large-scale MoE training with extremely low precision faces unique challenges, as aggressive quantization can more easily result in expert collapse and training instability compared to dense models. The dynamic routing mechanisms in MoE models are particularly sensitive to precision reduction, as small quantization errors can significantly impact expert selection decisions. Despite these challenges, the substantial memory savings achievable through low-precision training make this an important area for continued research and development.

5.5 Technical insights and research directions

Current MoE memory optimization techniques reveal several critical insights about expert activation patterns and memory hierarchy utilization that will guide future research directions. Expert activation exhibits strong temporal and spatial locality, with certain experts being co-activated frequently within short time windows and across similar input sequences[4, 6]. This locality can be exploited through intelligent prefetching and expert

clustering strategies that group frequently co-activated experts in the same memory regions.

The heterogeneous nature of expert utilization presents opportunities for adaptive precision strategies. Frequently activated experts that handle common patterns can benefit from higher precision to maintain model quality, while specialized experts that activate infrequently can tolerate lower precision with minimal impact on overall performance. This insight suggests dynamic precision scaling strategies where expert precision is adjusted based on utilization, activation importance scores, and gradient magnitudes during the training period.

On the other hand, the optimal balance between memory usage and computational overhead is highly dependent on hardware characteristics and workload patterns, but importantly, this balance is dynamic rather than static. The trade-off changes based on several factors: (1) batch size effects, where larger batch sizes amortize expert loading costs but increase memory pressure; (2) sequence length dependencies, where longer sequences benefit more from expert caching due to increased activation probability; and (3) hardware topology characteristics, where memory bandwidth, cache hierarchy, and interconnect properties fundamentally alter optimal operating points.

6 Communication Optimization

The sparse activation and dynamic routing characteristics of MoE models introduce significant communication challenges that fundamentally differ from traditional dense models. Traditional distributed training systems, designed for dense models with regular data distributions and predictable communication patterns, provide limited effectiveness for MoE optimization where data-dependent communication pattern should be considered during runtime.

This section discusses MoE-oriented communication optimization technologies, which are organized into three key categories: All-to-All collective optimization, computation-communication overlap techniques, and communication compression methods. We outline related work in Fig. 6. Through systematic analysis, we extract key insights about the importance of topology-aware scheduling, the resource complementarity enabling effective overlap, and emerging opportunities in compression-aware communication where operations can be performed directly on compressed representations without full decompression.

6.1 Collectives redesign and scheduling

The naive All-to-All collective (also called Linear All-to-All) requires all participants to communicate with each other simultaneously. This approach is agnostic to bandwidth heterogeneity between intra-node and inter-node connections. Consequently, it results in low utilization of intra-node high-bandwidth links during data exchange and synchronization[55]. Furthermore, when training MoE models with hybrid parallelization methods, the interleaving between different collective communications, especially All-to-All and All-Reduce operations in the backward process, creates bandwidth competition[74]. Therefore, resource-aware All-to-All communication is essential for improving training efficiency.

Hierarchical All-to-All: To adapt to heterogeneous networking bandwidth, hierarchical All-to-All

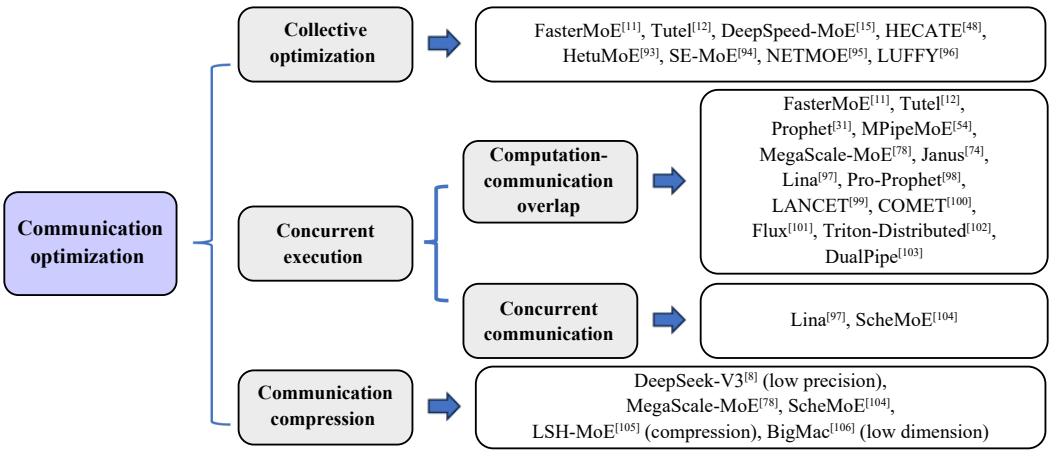

Fig. 6 Communication optimization techniques taxonomy in MoE model training systems.

approaches, where data exchange across workers is enforced with awareness of cluster topology, have been studied and implemented by several MoE training systems. For example, HetuMoE[93] implements a topology-aware One-Dimensional (1D) All-to-All approach where data destined for GPUs within the same node are first aggregated and sent together, then scattered to individual GPUs within the node. This design fully utilizes intra-node bandwidth while reducing the number of inter-node communications.

Building upon the 1D approach, both DeepSpeed-MoE[15] and SE-MoE[94] implement Two-Dimensional (2D) All-to-All communication, which decomposes the operation into separate intra-node and inter-node phases. This approach leverages intra-node high-bandwidth connections to minimize local data exchange latency and prepares large data chunks for efficient inter-node transfer. FasterMoE[11] extends this concept with group-based All-to-All, enabling intra-node data exchange to complete rapidly before initiating expert computation. Tutel[12] further optimizes the 2D All-to-All approach by improving data layout and leveraging compiler optimizations to reduce synchronization overhead between different All-to-All stages. More recently, HECATE[48] introduces advanced scheduling algorithms that dynamically adapt to network conditions and load patterns.

Locality aware scheduling: Beyond algorithmic optimizations, exploiting locality principles have shown significant promise. NETMOE[95] introduces network-aware expert placement strategies that minimize cross-node communication by co-locating frequently accessed experts. LUFFY[96] extends this concept by implementing dynamic expert migration based on workload patterns, achieving substantial communication overhead reduction in heterogeneous cluster environments.

6.2 Concurrent execution

The naive workflow of MoE training systems typically implements communication and computation, as well as different types of collective communication, serially. For example, when training MoE models with hybrid parallelism strategies like expert parallelism and expert-sharding parallelism, All-Gather or Reduce-Scatter operations for data collection can be executed concurrently with All-to-All[62]. Additionally, researchers have observed that All-to-All operations can be delayed by All-Reduce during the backward process. Efficiently interleaving different tasks through computation-communication overlap or interleaved communication represents an effective approach to improve training efficiency. This section discusses computation-communication optimization and interleaved communication optimization.

6.2.1 Computation-communication overlap

Unlike the strict dependency relationships of traditional dense models, expert computation in MoE exhibits natural parallelism: forward computation of different experts can proceed simultaneously, and computation within experts can overlap with communication of other experts. Thus, optimizing MoE training system performance through computation-communication overlap methodologies has attracted significant interest from both academia and industry.

Task-level overlap: The most straightforward approach to overlap communication and computation is to treat them as complete task units and enable overlapping at the task level through scheduling mechanisms, without modifying operator implementations. FasterMoE[11] and Tutel[12] separate these two different types of tasks into distinct Compute Unified Devices Architecture (CUDA) streams and enforce out-of-order asynchronous communication/computation sub-task execution whenever their data dependencies are satisfied. Tutel further introduces unified data layout abstractions that maintain compatibility across different parallelism strategies, enabling transparent switching without expensive data reorganization. Inspired by the micro-batching technique used in GPipe[36], MPipeMoE[55] enforces overlap by partitioning mini-batch data into several micro-batches and interleaving communication and computation operations for adjacent micro-batches.

Beyond basic asynchronous scheduling, predictive approaches leverage temporal patterns in expert activation to proactively allocate resources and schedule tasks. Prophet[31] exploits temporal locality in token routing patterns to predict expert loads and proactively schedule parameter transfers across iterations, while restructuring gradient aggregation to overlap expert All-Reduce with non-expert layer communications. Building upon this idea, Pro-Prophet[98] systematically decomposes load balancing into offline planner (profiling inter-iteration distributions) and online scheduler (maximizing overlap within iterations), enabling data-dependent

optimization. Janus[74] implements a data-centric framework where experts migrate on-demand between workers, coordinating asynchronous CPU-GPU transfers with computation scheduling to minimize idle time.

LANCET[99] extends task-level overlap optimization from individual MoE layers to the entire computation graph. Through Intermediate Representation (IR) analysis, LANCET discovers global overlap opportunities that are invisible to local optimizers, enabling whole-graph coordination of communication and computation. DeepSeek-V3[8] introduces DualPipe[103], an innovative bidirectional pipeline parallelism algorithm that achieves near-perfect overlap through careful task-level scheduling. DualPipe employs bidirectional pipeline with fine-grained chunk decomposition (attention, All-to-All dispatch, MLP, and All-to-All combine) and manually adjusts the ratio of GPU Streaming Multiprocessors (SMs) dedicated to communication versus computation within each chunk, maximizing hardware utilization through resource allocation at the task level.

Operator-level overlap: While task-level approaches schedule coarse-grained operations, operator-level techniques dive into individual operators, decomposing them into finer-grained sub-operations to enable deeper integration of communication and computation. However, concurrently running computation and communication kernels on the same accelerator can incur performance degradation[12]. Therefore, the key to effectively overlapping computation and communication at operator level lies in kernel fusion and fine-grained decomposition that allows seamless integration within unified kernels.

FLUX[101] introduces a novel fine-grained decomposition approach that over-decomposes communication and computation operations into much finer-grained tiles than existing methods, then fuses tiled computation and communication into a single larger kernel. This aggressive decomposition enables communication and computation to execute concurrently within the same kernel, eliminating synchronization overhead between separate operations. COMET[100] observes that granularity mismatch between communication and computation makes task-level overlap difficult to fully utilize accelerators, resulting in suboptimal efficiency. It addresses this by decomposing coarse-grained operation dependencies into fine-grained data dependencies, allowing communication and computation to be fused into a single kernel at tensor element granularity rather than operation level.

Triton-distributed[102] takes a compiler-based approach to operator-level overlap. By integrating communication primitives compliant with the OpenSHMEM standard[107] directly into the Triton compiler[108], Triton-distributed allows programmers to achieve complex joint optimization of computation, memory access, and communication through a unified high-level Python programming model. The compiler automatically generates fused kernels that seamlessly integrate communication operations with computation at the operator level.

Hybrid approaches: Several advanced systems combine both task-level and operator-level techniques to achieve superior overlap efficiency. Lina[97] operates at both levels: at the task level, it conducts deep analysis of expert activation patterns to discover temporal locality, enabling predictive prefetching and priority-based collectives scheduling; at the operator level, it decomposes communication operations into uniform micro-ops for fine-grained scheduling, allowing more precise control over communication-computation interleaving. MegaScale-MoE[78] implements a comprehensive hierarchical overlap scheme that explicitly optimizes at both levels. At the inter-operator (task) level, MegaScale-MoE achieves overlap through macro module execution that coordinates multiple operators. At the intra-operator level, it employs tile-based GEMM decomposition, breaking down individual operators into tiles that enable concurrent execution of communication and computation kernels within the same operator. This two-level strategy allows MegaScale-MoE to maximize overlap opportunities across different granularities.

The detailed comparison of these techniques is demonstrated in Table 2. The evolution from task-level to operator-level and finally to hybrid approaches reveals an important trend: as systems mature, they increasingly adopt multi-level optimization strategies that coordinate scheduling decisions across different granularities to achieve near-optimal overlap efficiency.

6.2.2 Concurrent collective communication

As discussed earlier, advancements in hybrid parallelism offer more optimization space for MoE training but lead to more sophisticated communication

Table 2 Comparative analysis of computation-communication overlap technologies in MoE training systems (√: Supported, ×: Not supported).

Work	Task-level overlap	Operator-level overlap	Compiler optimization	Expert prediction	Resource allocation	Core technical innovation
FasterMoE[11]	√	×	×	×	√	Pioneers asynchronous sub-task execution model with dedicated CUDA streams for communication and computation, enabling out-of-order execution when data dependencies permit.
Tutel[12]	√	×	√	×	√	Introduces unified data layout abstraction that maintains compatibility across DP/EP/hybrid configurations, enabling transparent parallelism switching without expensive data reorganization.
MPipeMoE[54]	√	×	×	×	√	Employs online algorithm to adaptively configure pipeline granularity for optimal communication-computation overlap in MoE sub-stages, coupled with performance-model-guided runtime selection among memory reusing strategies.
Prophet[31]	√	×	×	√	×	Exploits temporal locality in token routing patterns to predict expert loads and proactively schedule parameter transfers across iterations, while restructuring gradient aggregation to overlap expert All-Reduce with non-expert layer communications.
Pro-Prophet[98]	√	×	×	√	√	Systematically decomposes load balancing into offline planner (profiling inter-iteration distributions) and online scheduler (maximizing overlap within iterations), enabling data-dependent optimization.
Lina[97]	√	√	×	√	√	Discovers temporal locality in expert activations through deep pattern analysis, enabling predictive prefetching and priority-based collectives scheduling that anticipates future expert requirements.
Janus[74]	√	×	×	√	√	Implements data-centric framework where experts migrate on-demand between workers, coordinating asynchronous CPU-GPU transfers with computation scheduling to minimize idle time.
LANCET[99]	√	×	√	√	×	Extends overlap optimization from individual MoE layers to entire computation graph through compiler IR analysis, discovering global overlap opportunities invisible to local optimizers.
FLUX[101]	×	√	√	×	√	Introduces aggressive over-decomposition paradigm, partitioning operations into much finer tiles than conventional approaches and fusing tiled communication with computation into unified kernels.
MegaScale-MoE[78]	√	√	×	×	√	Implements hierarchical overlap at inter-operator level through macro module execution, and at intra-operator level through tile-based GEMM decomposition enabling concurrent execution of communication and computation kernels.
DualPipe[103]	√	×	√	×	√	Achieves near-perfect overlap through bidirectional pipeline with fine-grained chunk decomposition (attention/A2A-dispatch/MLP/A2A-combine) and manual SM allocation between communication and computation.
COMET[100]	×	√	√	×	×	Decomposes coarse computation and communication operation dependencies into fine-grained data dependencies, enabling kernel fusion at tensor element granularity rather than operation level.
Triton-distributed[102]	×	√	√	×	√	Integrates OpenSHMEM primitives directly into Triton compiler, enabling programmers to express complex joint optimizations of computation, memory access, and communication through unified high-level Python interface.

patterns. One limitation of traditional All-to-All collectives is that different tasks (or stages) are implemented serially. Since there are no data dependencies between intra-node and inter-node operations, All-to-All can be decoupled into two independent tasks and executed concurrently. For example, ScheMoE[104] addresses the limitation of serial task execution in traditional All-to-All implementations by decoupling intra-node and inter-node communications into independent tasks executed concurrently using separate CUDA streams. This approach effectively hides intra-node communication latency while maintaining computation efficiency. Beyond decoupling communication tasks, another critical challenge emerges from bandwidth contention between different collective operations during concurrent execution. Lina[97] systematically analyzes this bottleneck, revealing that All-to-All and All-Reduce operations often contend for network bandwidth when they overlap in the backward pass. To address this issue, Lina introduces tensor partitioning that decomposes communication operations into uniform micro-ops, enabling fine-grained priority scheduling where All-to-All operations receive dedicated bandwidth while All-Reduce operations utilize remaining resources opportunistically.

6.3 Communication compression

In the MoE training context, typical communication volume in each All-to-All operation can reach millions of bytes[104]. While previous sections discussed communication optimization through collective scheduling and overlapping, from a data perspective, an efficient approach to reduce All-to-All communication latency is to compress data volume or reduce data size. However, communication compression techniques face unique challenges in MoE training scenarios: compression algorithms must reduce transmission volume without destroying token semantics and routing information. This constraint renders many compression techniques effective in traditional distributed training ineffective in MoE scenarios.

Volume compression: One straightforward approach to enforce communication compression is to compress and decompress data before and after All-to-All communication. For instance, ScheMoE[104] designs abstractions for data compression and decompression in MoE layers, where low-bit data representation and compression algorithms can be utilized to reduce communication volume. From an information-theoretic perspective, the compression potential of MoE communication primarily derives from redundancy in token embeddings and predictability of routing decisions. By observing that data after routing decisions exhibit similarity, LSH-MoE[105] utilizes Principal Component Analysis (PCA) to cluster data and transmits only centroids rather than original data to reduce communication volume.

Lower dimension: Another method to reduce communication overhead is through low-dimensional communication approaches. BigMac[106] introduces Dimension-Compressed Communication Adaptation (DCCA), where tensor dimensions are scaled down before the first All-to-All operation and scaled up after the second All-to-All communication. By adding descending and ascending projections at expert entrance and exit points, BigMac achieves comparable model quality to fine-grained MoEs while reducing end-to-end latency by up to 3.09× for training.

Lower precision: Expressing data with lower bit precision is a common method to reduce space or bandwidth requirements. Quantization techniques have demonstrated enormous potential in MoE compression. QMoE[109] achieves breakthrough compression results, compressing the 1.6-trillion-parameter Switch Transformer[4] from 3.2 TB to 160 GB, achieving extreme compression of less than 1 bit per parameter. MegaScale-MoE[78] implements a multi-tier precision strategy: reducing inter-node parameter synchronization precision from 32-bit Floating Point (FP32) to BF16, effectively halving communication overhead while maintaining convergence stability. DeepSeek-V3[8] adopts FP8 quantization for expert parallelism, reducing communication volume by half compared to BF16.

Compression-aware communication: Existing compression approaches in MoE systems follow a compress-communicate-decompress paradigm, where activations are compressed before All-to-All communication and fully decompressed upon arrival before computation. While this reduces transmission overhead, the decompression step remains on the critical path, creating performance bottlenecks.

A promising alternative, inspired by database systems like TADOC[110] and CompressDB[111] that perform operations directly on compressed data, is to perform computation directly on compressed

activations without full decompression. This approach fundamentally rethinks the communication-computation interface by treating compressed representations as computational substrates rather than mere storage formats. While systems like DeepSeek-V3[8] integrate low-precision communication with selective full-precision computation for numerically sensitive operations, current implementations still treat compressed formats purely as communication encodings. Developing truly compression-aware communication—where operations execute directly on compressed data—represents a significant opportunity for future research.

6.4 Technical insights and research directions

Effective MoE communication optimization requires balancing three competing objectives: communication volume reduction, computational overhead minimization, and numerical precision preservation. Exploiting the inherent characteristics of MoE workloads, particularly the heterogeneous nature of expert utilization creates unique optimization opportunities through predictive scheduling and adaptive resource allocation. What's more, innovative communication optimization also can create new parallelism opportunities impossible to achieve through computation optimization alone. Also, next-generation MoE systems would design tight integration between hardware design and software optimization, including specialized communication accelerators, optimized network topologies for dynamic routing, and memory architectures exploiting expert computation locality.

7 Load Balancing Optimization

Load imbalance in MoE systems manifests in multiple dimensions. During training, some experts may become overloaded while others remain underutilized, leading to inefficiencies and routing collapse where the model repeatedly selects a few experts. This phenomenon occurs because in a normal MoE training, the gating network converges to mostly activate the same few experts, which self-reinforces as favored experts are trained quicker and hence selected more. The consequences are multifaceted: computational resources are wasted on idle experts, communication bottlenecks emerge around popular experts, and model capacity is underutilized.

Traditional approaches to mitigate this imbalance have relied on auxiliary loss functions[3]. An auxiliary loss is added to encourage giving all experts equal importance, ensuring that all experts receive a roughly equal number of training examples. However, these approaches introduce their own complications. The auxiliary loss introduces interference gradients during training, which conflict with the primary objective of the model—language modeling, making it difficult to balance load and achieve high levels of accuracy. In this section we discuss the load balancing technologies from expert capability management and adaptive expert placement two system perspectives.

7.1 Expert capability management

The fundamental challenge in MoE systems lies in managing computational resources when expert popularity varies dramatically and unpredictably during training. Early MoE implementations often suffered from either catastrophic memory overflow when popular experts received too many tokens, or significant computational waste due to conservative over-provisioning. Traditional capacity management approaches relied on static allocation strategies that failed to adapt to the dynamic nature of expert selection patterns, leading to either token dropping that degraded model quality or excessive memory consumption that limited scalability.

Expert capacity management represents one of the most critical aspects of MoE system design, directly impacting both training stability and computational efficiency. The concept was first systematically introduced in GShard[6] and further refined in Switch Transformer[4], establishing the theoretical and practical foundations for managing token overflow in sparse expert routing.

The Expert Capacity (EC) is mathematically defined as

$$\mathrm{EC} = \frac{B \times \mathrm{CF} \times K}{N},$$

where B stands for batch size, CF represents the capacity factor that determines the balance between computational efficiency and token utilization[4], K indicates the number of selected experts, and N denotes the number of experts. When $\mathrm{CF} < 1$, some tokens will always be dropped, whereas when $\mathrm{CF} > 1$, slack capacity is added to reduce token dropping frequency, albeit at the cost of increased memory consumption and potential computational overhead.

The capacity management extends beyond simple token allocation and encompasses sophisticated

overflow handling mechanisms. Expert capacity introduces a threshold of how many tokens can be processed by an expert, and when the number of tokens routed to an expert exceeds its capacity, the excess tokens are considered overflowed. The Switch Transformer[4] pioneers the approach where overflowed tokens are sent to the next layer via residual connections, effectively bypassing the MoE layer entirely. This mechanism provides a fail-safe against extreme load imbalances while maintaining system stability and ensuring that no computational paths are permanently blocked.

Recent advances have introduced more sophisticated capacity management strategies. Vision-MoE (V-MoE)[112] demonstrates that in computer vision applications, where images typically generate many patches, a small pre-defined expert capacity is essential to reduce hardware constraints, though low capacity tends to lead to patches being dropped, creating a critical trade-off between memory efficiency and information preservation. The mathematical formulation of token overflow has been formalized in recent work[113], where the overflow probability $P(\text{overflow}_i)$ for a given expert i can be expressed as

$$P(\text{overflow}_i) = P\left(\sum_{j=1}^{N_{\text{token}}} [\text{route}\,(j) = i] > \text{EC}_i\right),$$

where $\text{route}\,(\cdot)$ indicates the routing function, EC_i presents the capacity of i-th expert, and N_{token} denotes the number of tokens. This formulation has led to the development of adaptive capacity mechanisms that dynamically adjust expert capacity based on real-time load patterns, representing a significant advancement over static capacity allocation strategies employed in earlier systems.

7.2 Adaptive expert placement

While capacity management aligns token allocations, adaptive expert placement techniques tackle runtime imbalance through expert replication and replacement. As shown in Fig. 7, the core idea of expert replacement is to identify cold and hot experts during runtime, then takes scheduling methods to adjust the workload in different accelerators.

Traditional load balancing approaches operated reactively, responding to imbalances after they occurred rather than predicting and preventing them. To improve this situation, FasterMoE[11] builds a predictive performance model and roofline-like

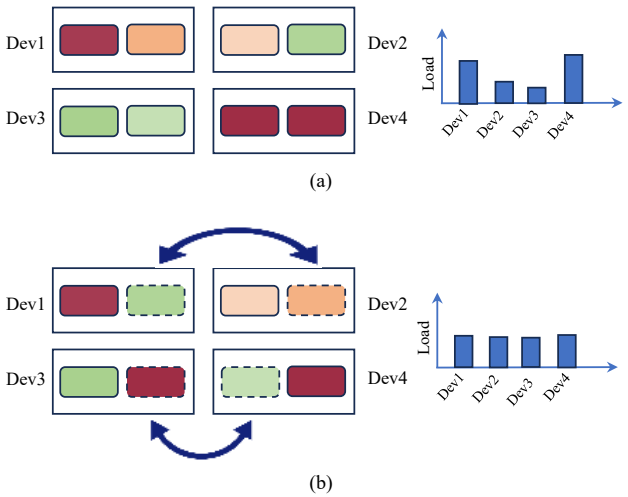

Fig. 7 Adaptive expert placement. (a) Load imbalance across devices and (b) load balancing with expert replacement.

analysis to guide dynamic "shadowing" of hot expert subsets and enforces topology-aware expert selection that considers network latency. SmartMoE[56] introduces an efficient searching algorithm within an enlarged hybrid parallelism space, achieving higher training throughput over FasterMoE by decomposing optimization opportunities into static pools offline and dynamic online selection.

As the expert replication strategies in FasterMoE follows a binary approach, either replicate experts across all workers or not at all, leading to excessive communication overhead and memory consumption. To improve this situation, Prophet[31] proposes replicating hot experts among partial subsets of workers rather than all workers, reducing communication overhead while maintaining load balance benefits. Building upon this idea, Pro-Prophet[98] decomposes load balancing into a planner that profiles inter-iteration token distributions and a scheduler that maximizes communication-computation overlap, achieving substantial load-balancing enhancement. FlexMoE[61] addresses the large online adjustment space in expert replacement through selective expert replication across GPUs according to popularity distribution, providing dynamic device placement that adapts to changing expert popularity patterns. HECATE[48] introduces hierarchical expert tree structures that enable multi-level load balancing and efficient expert discovery.

Data locality based optimization: A critical oversight in early MoE systems was the failure to

consider data locality when making expert placement decisions. Simply distributing experts to balance computational load often result in increased data movement overhead that negated the performance benefits of load balancing. The lack of integrated approaches that jointly optimize computation distribution and data placement leads to suboptimal overall system performance, particularly in distributed training scenarios where communication costs dominate. CCFuser[59] observes that distributing hot experts across accelerators destroys data locality. To address this, it places hot experts on accelerators where data resides while distributing only non-hot experts across devices.

Fault-tolerant and advanced scheduling: Existing MoE systems were designed primarily for stable, fault-free environments and lacked robust mechanisms to handle node failures or network partitions. The absence of fault-tolerant expert scheduling meant that system failures could lead to significant performance degradation or complete training interruption. Moreover, existing approaches do not consider how to maintain load balancing properties when the available computational resources change dynamically due to failures or resource allocation changes. Lazarus[58] applies provably optimal replica placement for hot experts under node-failure constraints, adaptively reallocating expert replicas to maintain balanced loads and full node utilization after failures.

7.3 Technical insights and research directions

The evolution of MoE load balancing techniques reveals several fundamental insights that shape the design of efficient sparse model training systems. Effective load balancing requires a holistic approach that considers the interplay between algorithm design, system architecture, and hardware characteristics. Moreover, the temporal dynamics of expert selection patterns necessitate adaptive rather than static optimization strategies. Expert popularity distributions evolve significantly during training, requiring systems that can continuously monitor and respond to changing load patterns. On the other hand, the trade-off between load balancing and data locality represents a fundamental challenge that requires joint optimization rather than sequential approaches. Hence, systems that prioritize perfect load balancing without considering data movement costs often achieve poor overall performance, highlighting the need for integrated optimization frameworks.

Machine learning guided load balancing represents a significant opportunity, where systems learn optimal expert placement and routing strategies from historical training data and real-time performance metrics. This approach could enable more sophisticated prediction and optimization capabilities than current heuristic-based methods. The integration of MoE load balancing with emerging hardware architectures, particularly specialized AI accelerators and memory-centric computing systems, offers opportunities for co-designed optimization strategies. Hardware-aware load balancing algorithms could exploit unique architectural features to achieve better performance than general-purpose approaches.

8 Design Principle

The discussion and analysis reveals in previous sections demonstrate that achieving efficient MoE training requires fundamentally different approaches from dense model training systems. The evolution of techniques exposes a consistent paradigm shift—from static, computation-centric optimizations designed for regular workloads to adaptive, workload-centric strategies that respond to MoE's dynamic behavior. By synthesizing insights from state-of-the-art systems, we identify five core design principles that could guide building efficient MoE training infrastructure:

Heterogeneity-aware optimization treats attention and expert layers as distinct computational paradigms requiring different optimization strategies. Systems that apply uniform parallelism, memory management, or communication scheduling across all layers systematically underperform those that exploit architectural heterogeneity. For instance, attention layers benefit from high-degree tensor parallelism and activation recomputation, while expert layers require expert-granular offloading and hierarchical All-to-All communication. This principle extends to hardware heterogeneity—effective systems must adapt strategies to bandwidth differences between intra-node and inter-node networks, and capacity differences between GPU HBM, Host DRAM, and NVMe storage.

Predictive resource management leverages temporal patterns in expert activation to proactively allocate resources rather than reactively responding to bottlenecks. We observe that predictive approaches—prefetching experts before activation, replicating hot experts before overload, and

compressing communication based on predicted routing patterns—usually outperform reactive strategies. The key implementation challenge lies in balancing prediction accuracy against overhead: lightweight online profiling that tracks recent activation patterns typically provides sufficient predictive power without introducing significant runtime costs.

Holistic optimization recognizes that optimizations in different dimensions interact in complex, often non-linear ways, requiring holistic rather than isolated approaches. For example, expert replication for load balancing increases memory consumption and communication volume, while aggressive memory offloading serializes expert execution and reduces opportunities for communication-computation overlap. Practically, this requires performance models that capture cross-dimensional effects and search algorithms that explore the combined optimization space rather than optimizing each dimension independently[114].

Adaptive precision and granularity matches optimization granularity and numerical precision to workload characteristics and resource constraints. Fine-grained optimizations (expert-level offloading, operator-level communication overlap, and selective activation recomputation) consistently outperform coarse-grained approaches (layer-level offloading, task-level overlap, and uniform recomputation) by exploiting MoE's sparse structure. Similarly, adaptive precision strategies—FP8 for communication, dynamic capacity factors for load balancing, variable recomputation policies across training phases—achieve better efficiency-quality trade-offs than static policies. The unifying insight is that one-size-fits-all strategies fail for MoE workloads; efficient systems must continuously adapt optimization decisions to the runtime conditions.

Locality-preserving transformations prioritize optimizations that preserve or enhance data locality over those that improve isolated metrics at the cost of increased data movement. This principle manifests differently across dimensions: in parallelization, it favors hierarchical strategies that maximize intra-node communication; in memory management, it suggests expert clustering based on co-activation patterns; in communication, it motivates expert placement strategies that co-locate frequently accessed experts; in load balancing, it implies that expert replication should account for data locality costs. The fundamental insight is that in modern GPU clusters, communication and memory access latency dominate computation time for MoE workloads, making locality the primary optimization objective.

9 Conclusion

This comprehensive survey has systematically discussed the accelerated technologies for MoE large language model training systems across four fundamental optimization dimensions: parallelism strategies, memory management, communication optimization, and load balancing. Our analysis reveals that MoE training system optimization represents a paradigmatic shift from traditional dense ones, requiring adaptive, holistic approaches that address the unique challenges of sparse activation, dynamic routing, and expert specialization.

Key design principles emerging from previous analysis include computational locality, adaptive precision management, and predictive resource allocation, which collectively enable efficient training of trillion-parameter MoE models. Looking forward, the integration of machine learning-guided optimization, hardware-software co-design, and novel MoE architectures presents promising directions for further advancing the efficiency and accessibility of large-scale MoE training. The optimization techniques surveyed here not only address current scalability challenges but also lay the foundation for democratizing access to large-scale Artificial Intelligence (AI) model training, ultimately accelerating innovation across diverse application domains.

Acknowledgment

This work was supported by the National Key R&D Program of China (No. 2023YFB3002002), the Beijing Natural Science Foundation (No. L242017), and the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDB0500103).

References

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, Scaling laws for neural language models, arXiv preprint arXiv: 2001.08361, 2020.

[2] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, Adaptive mixtures of local experts, *Neural*

Comput., vol. 3, no. 1, pp. 79–87, 1991.

[3] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, arXiv preprint arXiv: 1701.06538, 2017.

[4] W. Fedus, B. Zoph, and N. Shazeer, Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, *J. Mach. Learn. Res.*, vol. 23, no. 1, p. 120, 2022.

[5] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, et al., GLaM: Efficient scaling of language models with mixture-of-experts, in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, 2022, pp. 5547–5569.

[6] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, GShard: Scaling giant models with conditional computation and automatic sharding, arXiv preprint arXiv: 2006.16668, 2020.

[7] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, et al., Mixtral of experts, arXiv preprint arXiv: 2401.04088, 2024.

[8] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, et al., DeepSeek-V3 technical report, arXiv preprint arXiv: 2412.19437, 2024.

[9] Snowflake AI Research Team, Snowflake arctic: The best LLM for enterprise AI—efficiently intelligent, truly open, https://www.snowflake.com/blog/arctic-open-efficient-foundation-language-models-snowflake/, 2024.

[10] Y. Tang, Y. Yin, Y. Wang, H. Zhou, Y. Pan, W. Guo, Z. Zhang, M. Rang, F. Liu, N. Zhang, et al., Pangu ultra MoE: How to train your big MoE on ascend NPUs, arXiv preprint arXiv: 2505.04519, 2025.

[11] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, FasterMoE: Modeling and optimizing training of large-scale dynamic pre-trained models, in *Proc. 27th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, Seoul, Republic of Korea, 2022, pp. 120–134.

[12] C. Hwang, W. Cui, Y. Xiong, Z. Yang, Z. Liu, H. Hu, Z. Wang, R. Salas, J. Jose, P. Ram, et al., Tutel: Adaptive mixture-of-experts at scale, in *Proc. 6th Conf. Machine Learning and Systems*, Miami, FL, USA, 2023, pp. 269–287.

[13] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Y. Zhao, A. Dai, Z. Chen, Q. Le, and J. Laudon, Mixture-of-experts with expert choice routing, in *Proc. 36th Int. Conf. Neural Information Processing Systems*, New Orleans, LA, USA, 2022, p. 515.

[14] Y. Kim, H. Lim, and D. Han, Scaling beyond the GPU memory limit for large mixture-of-experts model training, in *Proc. 41st Int. Conf. Machine Learning*, Vienna, Austria, 2024, p. 975.

[15] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale, in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, 2022, pp. 18332–18346.

[16] B. Zhuang, J. Liu, Z. Pan, H. He, Y. Weng, and C. Shen, A survey on efficient training of transformers, in *Proc. 32nd Int. Joint Conf. Artificial Intelligence*, Macao, China, 2023, p.764.

[17] J. Duan, S. Zhang, Z. Wang, L. Jiang, W. Qu, Q. Hu, G. Wang, Q. Weng, H. Yan, X. Zhang, et al., Efficient training of large language models on distributed infrastructures: A survey, arXiv preprint arXiv: 2407.20018, 2024.

[18] M. Xu, D. Cai, W. Yin, S. Wang, X. Jin, and X. Liu, Resource-efficient algorithms and systems of foundation models: A survey, *ACM Comput. Surveys*, vol. 57, no. 5, p. 110, 2025.

[19] T. Fan, Q. Chen, J. Wang, Y. Liu, and S. Zhang, A comprehensive survey on mixture of experts: Algorithms, systems, and applications, arXiv preprint arXiv: 2407.06204, 2024.

[20] J. Liu, P. Tang, W. Wang, Y. Ren, X. Hou, P. A. Heng, M. Guo, and C. Li, A survey on inference optimization techniques for mixture of experts models, arXiv preprint arXiv: 2412.14219, 2024.

[21] D. Zhang, J. Song, Z. Bi, Y. Yuan, T. Wang, J. Yeong, and J. Hao, Mixture of experts in large language models, arXiv preprint arXiv: 2507.11181, 2025.

[22] W. Cai, J. Jiang, F. Wang, J. Tang, S. Kim, and J. Huang, A survey on mixture of experts in large language models, *IEEE Trans. Knowledge Data Eng.*, vol. 37, no. 7, pp. 3896–3915, 2025.

[23] The Mosaic Research Team, Introducing DBRX: A new state-of-the-art open LLM, https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm, 2024.

[24] J. Krajewski, M. Chochowski, and D. Korzekwa, Scaling fine-grained MoE beyond 50B parameters: Empirical evaluation and practical insights, arXiv preprint arXiv: 2506.02890, 2025.

[25] X. Sun, Y. Chen, Y. Huang, R. Xie, J. Zhu, K. Zhang, S. Li, Z. Yang, J. Han, X. Shu, et al., Hunyuan-Large: An open-source MoE model with 52 billion activated parameters by Tencent, arXiv preprint arXiv: 2411.02265, 2024.

[26] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, et al., Qwen2 technical report, arXiv preprint arXiv: 2407.10671, 2024.

[27] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G. Y. Wei, P. Bailis, V. Bittorf, et al., MLPerf training benchmark, in *Proc. 3rd Conf. Machine Learning and Systems*, Austin, TX, USA, 2020, pp. 336–349.

[28] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, Megatron-LM: Training multi-billion parameter language models using model parallelism, arXiv preprint arXiv: 1909.08053, 2019.

[29] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, Reducing activation recomputation in large transformer models, in *Proc. 6th Conf. Machine Learning and Systems*, Miami, FL, USA, 2023, pp. 341–353.

[30] S. Shi, X. Pan, X. Chu, and B. Li, PipeMoE: Accelerating

mixture-of-experts through adaptive pipelining, in *Proc. IEEE INFOCOM 2023-IEEE Conf. Computer Communications*, New York City, NY, USA, 2023, pp. 1–10.

[31] W. Wang, Z. Lai, S. Li, W. Liu, K. Ge, Y. Liu, A. Shen, and D. Li, Prophet: Fine-grained load balancing for parallel training of large-scale MoE models, in *Proc. 2023 IEEE Int. Conf. Cluster Computing* (*CLUSTER*), Santa Fe, NM, USA, 2023, pp. 82–94.

[32] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, et al., PyTorch distributed: Experiences on accelerating data parallel training, *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3005–3018, 2020.

[33] A. Sergeev and M. Del Balso, Horovod: Fast and easy distributed deep learning in TensorFlow, arXiv preprint arXiv: 1802.05799, 2018.

[34] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, ZeRo: Memory optimizations toward training trillion parameter models, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Atlanta, GA, USA, 2020, pp. 1–16.

[35] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, et al., Efficient large-scale language model training on GPU clusters using megatron-LM, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, St. Louis MO, USA, p. 58, 2021.

[36] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al., GPipe: Efficient training of giant neural networks using pipeline parallelism, in *Proc. 33rd Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019, p. 10.

[37] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, PipeDream: Generalized pipeline parallelism for DNN training, in *Proc. 27th ACM Symp. Operating Systems Principles*, Huntsville, Canada, 2019, pp. 1–15.

[38] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia, et al., DAPPLE: A pipelined data parallel approach for training large models, in *Proc. 26th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, Virtual Event, 2021, pp. 431– 445.

[39] P. Qi, X. Wan, G. Huang, and M. Lin, Zero bubble pipeline parallelism, arXiv preprint arXiv: 2401.10241, 2023.

[40] H. Liu, M. Zaharia, and P. Abbeel, Ring attention with blockwise transformers for near-infinite context, arXiv preprint arXiv: 2310.01889, 2023.

[41] A. Vavre, E. He, D. Liu, Z. Yan, J. Yang, N. Tajbakhsh, and A. Aithal, Llama 3 meets MoE: Efficient upcycling, arXiv preprint arXiv: 2412.09952, 2024.

[42] S. Roller, S. Sukhbaatar, A. Szlam, and J. Weston, Hash layers for large sparse models, in *Proc. 35th Int. Conf. Neural Information Processing Systems*, Virtual Event, 2021, p. 1343.

[43] A. Clark, D. de las Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud, et al., Unified scaling laws for routed language models, in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, 2022, pp. 4057–4086.

[44] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, FastMoE: A fast mixture-of-expert training system, arXiv preprint arXiv: 2103.13262, 2021.

[45] Z. Ma, J. He, J. Qiu, H. Cao, Y. Wang, Z. Sun, L. Zheng, H. Wang, S. Tang, T. Zheng, et al., BaGuaLu: Targeting brain scale pretrained models with over 37 million cores, in *Proc. 27th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, Seoul, Republic of Korea, 2022, pp. 192–204.

[46] HPC-AI Tech, Colossal-AI: Making large AI models cheaper, faster, and more accessible, https://github.com/hpcaitech/ColossalAI, 2024.

[47] Y. Wei, J. Du, J. Jiang, X. Shi, X. Zhang, D. Huang, N. Xiao, and Y. Lu, APTMoE: Affinity-aware pipeline tuning for MoE models on bandwidth-constrained GPU nodes, in *Proc. Int. Conf. High Performance Computing, Networking, Storage, and Analysis*, Atlanta, GA, USA, 2024, p. 90.

[48] Y. Qing, G. Zhu, F. Li, L. Lei, Z. Sun, X. Guan, S. Zhao, X. Chen, D. Huang, S. Wang, et al., Hecate: Unlocking efficient sparse model training via fully sharded sparse data parallelism, arXiv preprint arXiv: 2502.02581, 2025.

[49] S. Singh, O. Ruwase, A. A. Awan, S. Rajbhandari, Y. He, and A. Bhatele, A hybrid tensor-expert-data parallelism approach to optimize mixture-of-experts training, in *Proc. 37th Int. Conf. Supercomputing*, Orlando, FL, USA, 2023, pp. 203–214.

[50] D. Liu, Z. Yan, X. Yao, T. Liu, V. Korthikanti, E. Wu, S. Fan, G. Deng, H. Bai, J. Chang, et al., MoE parallel folding: Heterogeneous parallelism mappings for efficient large-scale MoE model training with megatron core, arXiv preprint arXiv: 2504.14960, 2025.

[51] N. Wang, W. Lin, L. Zhang, S. Shi, R. Zhou, and B. Li, SP-MoE: Expediting mixture-of-experts training with optimized pipelining planning, in *Proc. IEEE INFOCOM 2025-IEEE Conf. Computer Communications*, London, UK, 2025, pp. 1–10.

[52] S. Luo, J. Peng, P. Li, H. Wang, and T. Chen, Hexa-MoE: Efficient and heterogeneous-aware MoE acceleration with zero computation redundancy, arXiv preprint arXiv: 2411.01288, 2024.

[53] Y. Wu, X. Liu, S. Jin, C. Xu, F. Qian, Z. M. Mao, M. Lentz, D. Zhuo, and I. Stoica, HeterMoE: Efficient training of mixture-of-experts models on heterogeneous GPUs, arXiv preprint arXiv: 2504.03871, 2025.

[54] Z. Zhang, D. Yang, Y. Xia, L. Ding, D. Tao, X. Zhou, and D. Cheng, MPipeMoE: Memory efficient MoE for pre-trained models with adaptive pipeline parallelism, in *Proc. 2023 IEEE Int. Parallel and Distributed Processing Symp.* (*IPDPS*), St. Petersburg, FL, USA, 2023, pp. 167–177.

[55] Z. Zhang, Y. Xia, H. Wang, D. Yang, C. Hu, X. Zhou, and D. Cheng, MPMoE: Memory efficient MoE for pre-

trained models with adaptive pipeline parallelism, *IEEE Trans. Parall. Distr. Syst*., vol. 35, no. 6, pp.998–1011, 2024.

[56] M. Zhai, J. He, Z. Ma, Z. Zong, R. Zhang, and J. Zhai, SMARTMoE: Efficiently training Sparsely-Activated models through combining offline and online parallelization, in *Proc. 2023 USENIX Ann. Technical Conf*., Boston, MA, USA, 2023, pp. 961–975.

[57] W. Cui, Z. Han, L. Ouyang, Y. Wang, N. Zheng, L. Ma, Y. Yang, F. Yang, J. Xue, L. Qiu, et al., Optimizing dynamic neural networks with brainstorm, in *Proc. 17th USENIX Symp. Operating Systems Design and Implementation*, Boston, MA, USA, 2023, pp. 797–815.

[58] Y. Wu, W. Qu, T. Tao, Z. Wang, W. Bai, Z. Li, Y. Tian, J. Zhang, M. Lentz, and D. Zhuo, Lazarus: Resilient and elastic training of mixture-of-experts models with adaptive expert placement, arXiv preprint arXiv: 2407.04656, 2024.

[59] H. Wang, Y. Xia, D. Yang, X. Zhou, and D. Cheng, Harnessing inter-GPU shared memory for seamless MoE communication-computation fusion, in *Proc. 30th ACM SIGPLAN Ann. Symp. Principles and Practice of Parallel Programming*, Las Vegas, NV, USA, 2025, pp. 170–182.

[60] Y. Zeng, C. Huang, Y. Mei, L. Zhang, T. Su, W. Ye, W. Shi, and S. Wang, EfficientMoE: Optimizing mixture-of-experts model training with adaptive load balance, *IEEE Trans. Parall. Distr. Syst.*, vol. 36, no. 4, pp. 677–688, 2025.

[61] X. Nie, X. Miao, Z. Wang, Z. Yang, J. Xue, L. Ma, G. Cao, and B. Cui, FlexMoE: Scaling large-scale sparse pre-trained model training via dynamic device placement, *Proc. ACM Manag. Data*, vol. 1, no. 1, p. 110, 2023.

[62] X. Pan, W. Lin, L. Zhang, S. Shi, Z. Tang, R. Wang, B. Li, and X. Chu, FSMoE: A flexible and scalable training system for sparse mixture-of-experts models, in *Proc. 30th ACM Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Rotterdam, the Netherlands, 2025, pp. 524–539.

[63] S. Li, H. Liu, Z. Bian, J. Fang, H. Huang, Y. Liu, B. Wang, and Y. You, Colossal-AI: A unified deep learning system for large-scale parallel training, in *Proc. 52nd Int. Conf. Parallel Processing*, Salt Lake City, UT, USA, 2023, pp. 766–775.

[64] NVIDIA Corporation, NVIDIA Megatron Core, https://developer.nvidia.com/megatron-core, 2025.

[65] NVIDIA Corporation, Megatron-LM: Ongoing research training transformer models at scale, https://gitee.com/zhang-pinge/Megatron-LM, 2025.

[66] S. Luo, J. Peng, P. Li, and T. Chen, HEXA-MoE: Efficient and heterogeneousa-ware MoE acceleration with zero computation redundancy, arXiv preprint arXiv: 2411.01288, 2024.

[67] L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing, et al., Alpa: Automating inter-and intra-operator parallelism for distributed deep learning, in *Proc. 16th USENIX Symp. Operating Systems Design and Implementation*, Carlsbad, CA, USA, 2022, pp. 559–578.

[68] G. Liu, Y. Miao, Z. Lin, X. Shi, S. Maleki, F. Yang, Y. Bao, and S. Wang, Aceso: Efficient parallel DNN training through iterative bottleneck alleviation, in *Proc. 19th European Conf. Computer Systems*, Athens, Greece, 2024, pp. 163–181.

[69] Z. Lin, Y. Miao, Q. Zhang, F. Yang, Y. Zhu, C. Li, S. Maleki, X. Cao, N. Shang, Y. Yang, et al., nnScaler: Constraint-guided parallelization plan generation for deep learning training, in *Proc. 18th USENIX Symp. Operating Systems Design and Implementation*, Santa Clara, CA, USA, 2024, pp. 347–363.

[70] Z. Zhu, C. Giannoula, M. Andoorveedu, Q. Su, K. Mangalam, B. Zheng, and G. Pekhimenko, Mist: Efficient distributed training of large language models via memory-parallelism co-optimization, in *Proc. 20th European Conf. Computer Systems*, Rotterdam, the Netherlands, 2025, pp. 1298–1316.

[71] K. Yang, L. Liu, H. Liu, and T. Deng, A novel parallel processing element architecture for accelerating ode and AI, *Tsinghua Science and Technology*, vol. 30, no. 5, pp. 1954–1964, 2025.

[72] Z. Zong, Y. Chen, Q. Zhang, D. Zhao, J. Li, Y. Jing, and J. Zhai, Accelerating distributed training of large concurrent-branch models through bidirectional pipeline coordination, *Tsinghua Science and Technology*, vol. 30, no. 6, pp. 2638–2652, 2025.

[73] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, MoE-infinity: Activation-aware expert offloading for efficient MoE serving, https://api.semanticscholar.org/CorpusID: 267211688, 2024.

[74] J. Liu, J. H. Wang, and Y. Jiang, Janus: A unified distributed training framework for sparse mixture-of-experts models, in *Proc. ACM SIGCOMM 2023 Conf*., New York, NY, USA, 2023, pp. 486–498.

[75] D. Yu, L. Shen, H. Hao, W. Gong, H. Wu, J. Bian, L. Dai, and H. Xiong, MoESys: A distributed and efficient mixture-of-experts training and inference system for internet services, *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2626–2639, 2024.

[76] J. Wei, A. Qiao, A. Jayarajan, G. Gibson, V. Vasudevan, and E. Xing, Training larger models on TensorFlow without additional GPU, https://api.semanticscholar.org/CorpusID:211223143, 2019.

[77] X. Nie, Y. Liu, F. Fu, J. Xue, D. Jiao, X. Miao, Y. Tao, and B. Cui, Angel-PTM: A scalable and economical large-scale pre-training system in tencent, *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3781–3794, 2023.

[78] C. Jin, Z. Jiang, Z. Bai, Z. Zhong, J. Liu, X. Li, N. Zheng, X. Wang, C. Xie, Q. Huang, et al., MegaScale-MoE: Large-scale communication-efficient training of mixture-of-experts models in production, arXiv preprint arXiv: 2505.11432, 2025.

[79] S. Gandhi and C. Kozyrakis, MoEtion: Efficient and reliable sparse checkpointing for mixture-of-experts models at scale, arXiv preprint arXiv: 2412.15411, 2024.

[80] NVIDIA Corporation, TransformerEngine: A library for accelerating transformer models on NVIDIA GPUs, https://github.com/NVIDIA/TransformerEngine, 2023.

[81] AWS Machine Learning Blog Team, Accelerate mixtral 8x7B pre-training with expert parallelism on amazon SageMaker, https://aws.amazon.com/blogs/machine-learning/accelerate-mixtral-8x7b-pre-training-with-expert-parallelism-on-amazon-sagemaker/, 2024.

[82] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters, in *Proc. 26th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, Virtual Event, 2020, pp. 3505–3506.

[83] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv: 1412.6980, 2014.

[84] Y. Zhao, A. Gu, R. Varma, L. Luo, C. C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, et al., PyTorch FSDP: Experiences on scaling fully sharded data parallel, *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 3848–3860, 2023.

[85] C. C. Huang, G. Jin, and J. Li, SwapAdvisor: Pushing deep learning beyond the GPU memory limit via smart swapping, in *Proc. 25th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Lausanne, Switzerland, 2020, pp. 1341–1355.

[86] X. Peng, X. Shi, H. Dai, H. Jin, W. Ma, Q. Xiong, F. Yang, and X. Qian, Capuchin: Tensor-based GPU memory management for deep learning, in *Proc. 25th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, Lausanne, Switzerland, 2020, pp. 891– 905.

[87] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, ZeRO-Offload: Democratizing billion-scale model training, in *Proc. 2021 USENIX Ann. Technical Conf.*, pp. 551–564. https://www.usenix.org/system/files/atc21-ren-jie.pdf, 2021.

[88] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, ZeRO-infinity: Breaking the GPU memory wall for extreme scale deep learning, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, St. Louis, MO, USA, 2021, p. 59.

[89] X. Sun, W. Wang, S. Qiu, R. Yang, S. Huang, J. Xu, and Z. Wang, STRONGHOLD: Fast and affordable billion-scale deep learning model training, in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, Dallas, TX, USA, 2022, pp. 1–17.

[90] K. Wu, J. B. Park, X. Zhang, M. Hidayetoğlu, V. S. Mailthody, S. Huang, S. Lumetta, and W. M. Hwu, SSDTrain: An activation offloading framework to SSDs for faster large language model training, in *Proc. 2025 62nd ACM/IEEE Design Automation Conf.*, San Francisco, CA, USA, 2025, pp. 1–7.

[91] W. Liu, M. Li, G. Tan, and W. Jia, Mario: Near zero-cost activation checkpointing in pipeline parallelism, in *Proc. 30th ACM SIGPLAN Ann. Symp. Principles and Practice of Parallel Programming*, Las Vegas, NV, USA, 2025, pp. 197–211.

[92] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, et al., FP8-LM: Training FP8 large language models, arXiv preprint arXiv: 2310.18313, 2023.

[93] X. Nie, P. Zhao, X. Miao, T. Zhao, and B. Cui, HetuMoE: An efficient trillion-scale mixture-of-expert distributed training system, arXiv preprint arXiv: 2203.14685, 2022.

[94] L. Shen, Z. Wu, W. Gong, H. Hao, Y. Bai, H. Wu, X. Wu, H. Xiong, D. Yu, and Y. Ma, Se-MoE: A scalable and efficient mixture-of-experts distributed training and inference system, arXiv preprint arXiv: 2205.10034v1, 2022.

[95] X. Liu, Y. Wang, F. Fu, X. Miao, S. Zhu, X. Nie, and B. Cui, NetMoE: Accelerating MoE training through dynamic sample placement, in *Proc. 13th Int. Conf. Learning Representations*, Singapore, 2025, pp. 1−19.

[96] F. Chen, P. Li, Z. Hong, Z. Su, and S. Guo, Communication-efficient sparsely-activated model training via sequence migration and token condensation, *IEEE Trans. Network.*, doi: 10.1109/TON.2025.3585359.

[97] J. Li, Y. Jiang, Y. Zhu, C. Wang, and H. Xu, Accelerating distributed MoE training and inference with Lina, in *Proc. 2023 USENIX Ann. Technical Conf.*, Boston, MA, USA, 2023, pp. 945–959.

[98] W. Wang, Z. Lai, S. Li, W. Liu, K. Ge, A. Shen, H. Su, and D. Li, Pro-Prophet: A systematic load balancing method for efficient parallel training of large-scale MoE models, arXiv preprint arXiv: 2411.10003, 2024.

[99] C. Jiang, Y. Tian, Z. Jia, S. Zheng, C. Wu, and Y. Wang, Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping, arXiv preprint arXiv: 2404.19429, 2024.

[100] S. Zhang, N. Zheng, H. Lin, Z. Jiang, W. Bao, C. Jiang, Q. Hou, W. Cui, S. Zheng, L. W. Chang, et al., Comet: Fine-grained computation-communication overlapping for mixture-of-experts, arXiv preprint arXiv: 2502.19811, 2025.

[101] L. W. Chang, W. Bao, Q. Hou, C. Jiang, N. Zheng, Y. Zhong, X. Zhang, Z. Song, C. Yao, Z. Jiang, et al., FLUX: Fast software-based communication overlap on GPUS through kernel fusion, arXiv preprint arXiv: 2406.06858, 2024.

[102] S. Zheng, W. Bao, Q. Hou, X. Zheng, J. Fang, C. Huang, T. Li, H. Duanmu, R. Chen, R. Xu, et al., Triton-distributed: Programming overlapping kernels on distributed AI systems with the triton compiler, arXiv preprint arXiv: 2504.19442, 2025.

[103] DeepSeek-AI, DualPipe: A bidirectional pipeline parallelism algorithm for computation-communication overlap in DeepSeek V3/R1 training, https://github.com/deepseek-ai/DualPipe, 2025.

[104] S. Shi, X. Pan, Q. Wang, C. Liu, X. Ren, Z. Hu, Y. Yang, B. Li, and X. Chu, ScheMoE: An extensible mixture-of-experts distributed training system with tasks scheduling, in *Proc. 19th European Conf. Computer Systems*, Athens, Greece, 2024, pp. 236–249.

[105] X. Nie, Q. Liu, F. Fu, S. Zhu, X. Miao, X. Li, Y. Zhang, S. Liu, and B. Cui, LSH-MoE: Communication-efficient MoE training via locality-sensitive hashing, in *Proc. 38th Int. Conf. Neural Information Processing Systems*,

Vancouver, Canada, 2024, p. 1716.
[106] Z. Jin, S. Wang, J. Zhu, H. Zhan, Y. Bai, L. Zhang, Z. Ming, and C. Li, BigMac: A communication-efficient mixture-of-experts model structure for fast training and inference, in *Proc. 39th AAAI Conf. Artificial Intelligence*, Philadelphia, PA, USA, 2025, pp. 17689–17698.
[107] Open SHMEM Specification Committee, OpenSHMEM application programming interface version 1.6, http://www.openshmem.org/site/sites/default/site_files/OpenSHMEM-1.6.pdf, 2024.
[108] Triton Contributors, Triton: Development repository for the triton language and compiler, https://github.com/triton-lang/triton, 2025.
[109] E. Frantar and D. Alistarh, QMoE: Practical sub-1-bit compression of trillion-parameter models, arXiv preprint arXiv: 2310.16795, 2023.
[110] F. Zhang, J. Zhai, X. Shen, D. Wang, Z. Chen, O. Mutlu, W. Chen, and X. Du, TADOC: Text analytics directly on compression, *VLDB J.*, vol. 30, no. 2, pp. 163–188, 2021.
[111] F. Zhang, W. Wan, C. Zhang, J. Zhai, Y. Chai, H. Li, and X. Du, CompressDB: Enabling efficient compressed data direct processing for various databases, in *Proc. 2022 Int. Conf. Management of Data*, Philadelphia, PA, USA, 2022, pp. 1655–1669.
[112] C. Riquelme, J. Puigcerver, B. Mustafa, M. Neumann, R. Jenatton, A. S. Pinto, D. Keysers, and N. Houlsby, Scaling vision with sparse mixture of experts, in *Proc. Ann. Conf. Neural Information Processing Systems 2021*, Virtual Event, 2021, pp. 8583–8595.
[113] Knowing enough about MoE to explain dropped tokens in GPT-4, https://152334h.github.io/blog/knowing-enough-about-moe/, 2023.
[114] R. Wang, W. Li, K. Shen, T. Zhang, and X. Liao, Evolutionary multi-tasking optimization for high-efficiency time series data clustering, *Tsinghua Science and Technology*, vol. 29, no. 2, pp. 343–355, 2024.

Qi Zhang received the PhD degree from Beihang University, China in 2023. He is currently a postdoctoral researcher of Department of Computer Science and Technology, Tsinghua University, China. His research interests include high performance computing, evaluation, and optimization for intelligent computing systems.

Jidong Zhai received the BEng degree in computer science from University of Electronic Science and Technology of China, China in 2003, and the PhD degree in computer science from Tsinghua University, China in 2010. He is currently a tenured professor at Department of Computer Science and Technology, Tsinghua University, China. His research interests include performance evaluation for high-performance computers, performance analysis, and modeling of parallel applications.

Weimin Zheng received the MEng degree from Tsinghua University, China in 1982. He is currently a professor at Department of Computer Science and Technology, Tsinghua University, China. His research covers distributed computing, compiler techniques, and network storage. He is an academician of the Chinese Academy of Engineering.