DIRECTOR: Accelerating Distributed MoE Serving via Online Proactive Expert Placement

Qianli Liu[1], Kaibin Guo[2], Zicong Hong[1], Peng Li[3], Fahao Chen[4], Haodong Wang[1], Jian Lin[1], and Song Guo[1]
[1]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong
[2]School of Software Engineering, Sun Yat-Sen University, China
[3]School of Cyber Science and Engineering, Xi'an Jiaotong University, China
[4]School of Artificial Intelligence, Shandong University, China
qianli.liu@connect.ust.hk, guokb@mail2.sysu.edu.cn, ziconghong@gmail.com, pengli@xjtu.edu.cn,
chenfh@ieee.org, hwanghb@connect.ust.hk, jlindc@connect.ust.hk, songguo@cse.ust.hk

***Abstract*—Expert parallelism has become the prevailing paradigm to serve Mixture-of-Experts (MoE) models. Its efficiency depends on the communication and computation latencies of the GPUs, which are linked to the placement of experts in the GPUs. Existing works for optimizing expert placement focus on leveraging past requests' expert activation patterns. However, they demonstrate deficiencies facing diverse and rapidly changing request patterns, calling for an online, proactive approach. Implementing such an approach requires addressing several challenges: the uncertainty associated with incoming requests' expert activation, the cost of expert migration, and the NP-hard complexity in optimization. Therefore, we present DIRECTOR, a new distributed MoE serving system that minimizes end-to-end latency via prediction-driven, online expert placement. DIRECTOR uses either a lightweight cascaded predictor or a low-bit quantized replica for expert activation patterns of incoming requests. An online migration module then enacts the changes with near-zero downtime by executing migrations in compute-bound phases, keeping disruption bounded. At its core, a relaxation-based expert placement optimizer operates under capacity constraints, runs in polynomial time, and achieves a $(1+\epsilon)$ approximation ratio. Finally, we implement a prototype and demonstrate, through extensive experiments, a reduction in end-to-end latency of $11 \sim 55\%$ for popular MoE models (e.g., Mistral, DeepSeek and Qwen) compared to existing work.**

I. INTRODUCTION

Mixture-of-Experts (MoE) models [1]–[7] have become a popular architecture for scaling large language models (LLMs) to trillions of parameters while keeping the computational cost proportional. Unlike dense models, which use all parameters, MoE models dynamically route input tokens to a subset of feed-forward networks (FFNs), known as *experts*. This selective activation allows the required FLOPs to scale sub-linearly as the model size increases [8]–[10].

The increasing parameter count of MoE models requires expert parallelism for distributed MoE serving due to the significant memory footprint [11]. In this paradigm, the experts are distributed across a cluster of GPUs, while non-expert parameters, such as self-attention blocks, are replicated on every GPU. A forward pass through each MoE layer requires a pair of all-to-all communication steps: one to dispatch tokens to their selected experts, and another to aggregate the results. This paradigm effectively utilises the sparse, conditional computation of MoE, establishing it as a widely adopted standard for serving large-scale MoE models.

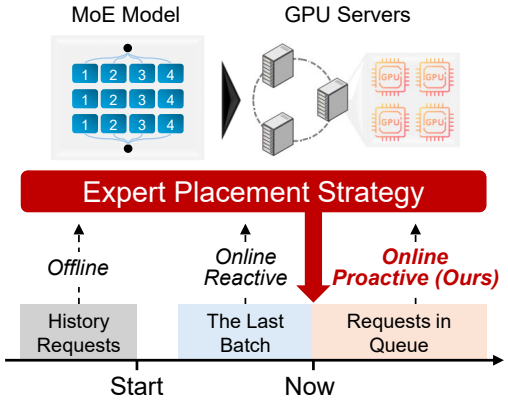


Fig. 1: A comparison of offline expert placement, online reactive expert placement, and online proactive expert placement.

Despite the advantages of expert parallelism, two performance bottlenecks have been identified: high latency during frequent all-to-all communication and an imbalanced computational load across GPUs. The communication bottleneck arises because the hidden state of each token must be dispatched to the selected experts during all-to-all operations for each MoE layer. These experts are often dispersed across the GPUs, and the hidden state is large. Furthermore, the imbalanced load is caused by a shift in distribution between the inference workloads and the model's training data. This non-uniform selection results in stragglers. The GPUs hosting these over-selected experts determine the overall latency, as the aggregation cannot proceed until the last expert has finalised its computation.

To accelerate the distributed MoE serving, as shown in Figure 1, recent works can be classified into two strategies: *offline*

This research was supported by fundings from the Hong Kong RGC General Research Fund (152169/22E, 152228/23E, 162161/24E, 162116/25E), Research Impact Fund (No. R5060-19, No. R5011-23), Collaborative Research Fund (No. C1042-23GF), NSFC/RGC Collaborative Research Scheme (Grant No. 62461160332 & CRS_HKUST602/24), Areas of Excellence Scheme (AoE/E-601/22-R), National Natural Science Foundation of China (No. 62471383), and the InnoHK (HKGAI). Corresponding authors: Zicong Hong, Song Guo.

expert placement and *online reactive expert placement*. The offline works compute a static expert placement before serving begins. By analyzing historical routing patterns and inter-layer expert affinities, they co-locate frequently co-activated experts to minimize communication and duplicate frequent experts to balance the load [11]–[13]. However, these works rely on pre-profiled data, which renders their static placement sub-optimal when the inference workload exhibits a shift in data distribution, resulting in performance degradation. In contrast, the online works adjust to the workload. They respond to dynamic workload by replicating overloaded experts [14]–[16] or choosing whether to move experts or requests [17], [18]. However, they often adopt an online reactive design which optimizes for the most recent batch, so decisions lag behind the workload; under rapidly shifting inputs, this lag yields persistent misalignment and poor performance (§ III).

Therefore, we propose a new paradigm: *online proactive expert placement* for distributed MoE serving. This would analyze the requests in the queue and optimize the placement of experts on GPUs before processing them. This anticipatory placement would co-locate experts likely to be activated together on the same GPU for the requests to be processed, thereby reducing the need for expensive inter-GPU communication. Furthermore, it ensures that the computational workload is distributed evenly across all GPUs, thereby maintaining system-wide load balancing and preventing bottlenecks caused by overloaded GPUs.

However, realizing this paradigm faces three key challenges: **1) Routing Path Uncertainty.** In MoE, the gate at layer l is evaluated on the representation produced by the experts chosen at layer l–1, routing is data-dependent and sequentially coupled across layers. Hence, a token's layerwise routing cannot be determined a priori; it is revealed only during the forward pass and must be estimated by prediction. **2) Cost of Adaptivity.** Migrating experts at run time competes directly with computation: while an expert's parameters are in transit, that expert cannot do computations, and the data transfer itself introduces communication overhead. An effective system must therefore balance migration and computation, scheduling transfers to limit the downtime and ensuring that any reconfiguration delivers a performance gain. **3) Complexity of Expert Placement.** The search space for an optimal expert placement is immense and relying on brute-force search or exact solvers to find the optimal placement is infeasible for real-time decision-making. For example, DeepSeekMoE 16B contains 27 MoE layers, each with 64 experts. When deployed across 4 GPUs in expert parallelism. In each layer, the number of ways to assign 16 experts to the first GPU is given by the combination C_{16}^{64}. Given that the model has 27 layers that can be configured independently, the total size of the placement search space approaches $(C_{16}^{64})^{27}$. Formally, the task of finding an optimal placement is an NP-hard problem [19].

To address these challenges, this paper introduces DIRECTOR, a distributed MoE serving system that performs prediction-driven, online expert placement reconfiguration. DIRECTOR uses either a cascaded predictor or a quantized MoE replica to accurately predict routing, thereby enabling the proactive updating of expert placements. A computation-overlapped migration module then enacts changes with near-zero downtime by executing transfers only in compute-bound phases, keeping disruption bounded. We summarize our contributions as follows.

- We design an online mechanism that couples routing prediction with execution and performs computation-overlapped live migration with near-zero downtime, enabling the system to anticipate routing and adjust placements with bounded impact.
- We develop a polynomial-time scheduling algorithm that finds an expert placement with a provable performance bound. The algorithm efficiently balances communication costs and GPU load to reduce overall latency.
- We implement and evaluate a prototype of DIRECTOR. Through extensive experiments on diverse workloads and hardware configurations, we demonstrate a significant end-to-end latency reduction of $11 \sim 55\%$ compared to existing MoE serving systems.

II. BACKGROUND & RELATED WORK

A. Mixture of Experts

MoE architecture replaces the standard FFN with an MoE layer in the transformer [20]. Each MoE layer consists of a set of FFN sub-networks known as *experts* and a learnable *gating network* that routes input tokens to them. For each incoming token, the gating network computes relevance scores and selects a small subset of experts for processing, commonly top-k routing. After the selected experts have processed the token, the outputs of these experts are aggregated, typically through a weighted summation based on the scores assigned by the gating network. This sparse activation increases the model's total parameter count without proportionally increasing the computation requirement for a single forward pass.

B. Expert Parallelism

In expert parallelism, the experts are distributed to different GPUs, with parameters such as the self-attention block, layer normalization and gating network being replicated on each GPU [8], [11]. During a forward pass, each GPU first applies these replicated components to its local tokens. The gate then selects the experts for each token, after which the tokens are packaged and sent to the GPUs hosting their targeted experts via all-to-all communication. Once the expert computations have finished, a second all-to-all communication process returns the token outputs to their original GPUs. This enables the model to proceed to the next layer.

C. Optimization for Expert Parallelism

Various optimisations have been introduced to improve the efficiency of distributed MoE training and inference.

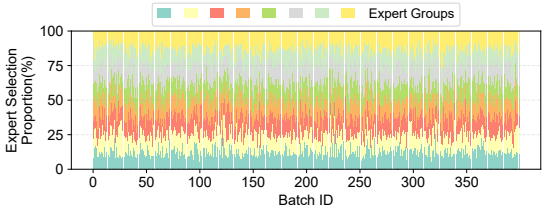


Fig. 2: Expert selection frequency in a single layer (layer 20 as an example) of DeepSeekMoE 16B with a batch size of 32.

a) Model Architecture & Routing Rule: Some works modify the model's structure to increase the likelihood of a token being processed locally. For instance, expert pruning decreases the number of experts, thereby reducing inter-GPU communication [21], [22]. Other works optimize the gate's selection in the forward pass. DeepSeek [6], [7], FasterMoE [14] and TA-MoE [23] enforce a constraint that limits the number of experts that can be selected from other GPUs. As our work focuses on optimizing the placement of experts at a system level, these algorithm-level optimizations are orthogonal.

b) Communication-Computation Pipeline: Some works recognize that standard execution results in a sequential bottleneck, whereby GPU-intensive computations are forced to wait for network-intensive data transfers. To eliminate this dependency, the input tensor is partitioned into smaller chunks. This creates a pipelined workflow that allows the expert computation on one chunk to run concurrently with the all-to-all communication of another [16], [24]–[28]. They optimize the pipeline using a uniform, random expert placement strategy, which is complementary to our work.

c) Expert Placement: The final family optimises the placement of experts on GPUs. These can be categorised as either determining an optimal placement offline or adapting it at runtime. The former leverages offline profiling to precompute a fixed expert layout [13], [29]. By analysing historical routing data or cross-layer expert affinities, they create a globally optimised, yet static, placement. The latter reacts to observed traffic by moving or creating replicas of popular experts on underutilised GPUs [14], [15], [17]. Another work combines the expert-centric approach (moving data) with a data-centric alternative (moving experts) [18]. A more recent paradigm dynamically rearranges the placement of sequences between devices [30], [31]. However, all these online works are reactive, optimizing placements based on past patterns. This inherent lag will lead to suboptimal performance for dynamic workloads, leaving a critical gap for our proactive design that can anticipate and adapt to incoming demands.

III. Motivation

This section analyses the characteristics of expert parallelism and identifies the bottleneck of existing expert placement approaches, revealing an opportunity for optimization.

Finding 1. *The activation patterns of experts within an MoE model exhibit significant dynamism when processing different request batches over time.*

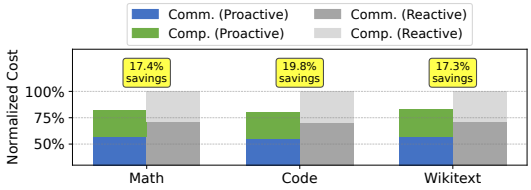


Fig. 3: The maximum reduction in latency achieved by the design of online proactive expert placement.

Both the offline and online reactive approaches operate on the assumption that the patterns of expert activation are relatively stable during the serving process [13]–[15], [17], [29], [32]. However, this may not be valid in realistic, multi-user serving environments. To demonstrate this, we randomly select requests from text generation [33], mathematical reasoning [34], code generation [35] and record their expert activation patterns. As shown in Figure 2, the selection frequency of any given expert varies significantly. This instability calls into question the effectiveness of both offline and online reactive approaches. This calls for a new *online proactive* approach that optimizes expert placement based on the expert activation patterns of incoming requests rather than the completed ones.

Finding 2. *When it comes to incoming requests, it is more effective to optimise expert placement based on their activation patterns than on past requests.*

Although an online proactive approach does not yet exist, we evaluate its potential benefits below. We choose a small-scale MoE model Mixtral 8×7B and compare the optimal latency of reactive and proactive strategies through exhaustive searching. In the reactive strategy, the optimal placement determined from the previous batch of requests is applied to the next one. In the proactive strategy, the optimal placement is determined from the incoming batch itself. We limit the number of copies of each expert to one here, since exhaustive searching would be impractical if expert replication were considered. We adopt the Cluster A configuration in § VIII. As shown in Figure 3, the optimal end-to-end latency can be reduced by up to 20%. Furthermore, we believe that the reduction will be amplified when considering expert replication. This shows the significant potential of online proactive design.

Finding 3. *Existing predictors of expert activation patterns with negligible overhead perform poorly in the state-of-the-art fine-grained MoE models.*

Implementing such an online proactive design needs to identify the expert routing pattern of incoming workload. One possible solution would be to use a small neural network for prediction. However, most existing predictors [29], [36] are only designed for traditional MoE models, e.g. Switch Transformers [9], that select a top-1/2 expert(s) per token. This problem is exacerbated by the architectural trend toward fine-grained MoE: state-of-the-art models are rapidly scaling their

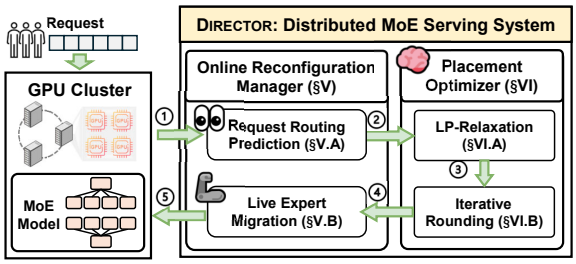


Fig. 4: The workflow of DIRECTOR.

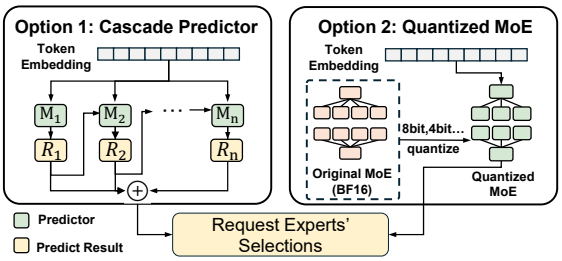


Fig. 5: Two options of predictors in DIRECTOR

TABLE I: Performance of the predictor across different MoE models. The overhead is measured as the ratio of prediction time to the per-token end-to-end generation latency.

Model	Expert Num. (per Layer)	Top-k Routing	Prediction Accuracy	Relative Overhead
Mixtral-8×7B	8	2	61.45%	1.70%
DeepSeekMoE-16B	64	6	56.14%	2.71%
DeepSeek-V2-Lite	64	6	55.17%	2.58%
Qwen3-30B-A3B	128	8	46.69%	1.59%

number of experts and top-k routing values. To evaluate the feasibility, we train a single-layer Transformer with limited overhead budget on the gate network's logits from historical requests—and evaluating it on several state-of-the-art MoE models. As shown in Table I, although the overhead is negligible, the predictor achieves up to 60% accuracy. Thus, we need to design a new predictor for the online proactive design in the state-of-the-art fine-grained MoE models.

IV. System Overview

Motivated by the findings in § III, we propose DIRECTOR, a new distributed MoE serving system with three design goals:

- **Proactive Placement**: Rather than optimising for past expert activation patterns, the system should optimise expert placements based on the incoming workloads.
- **Live Reconfiguration**: The reconfiguration of expert placement should result in minimal downtime and a limited impact on performance.
- **Performance Guarantee**: The placement algorithm should be designed to provide a guaranteed performance solution in a limited amount of time.

The system overview is shown in Figure 4. It employs two key components: an online reconfiguration manager (§ V) and a relaxation-based placement optimizer (§ VI). The former is responsible for perceiving expert routing of incoming workloads and executing expert reconfiguration. The latter is responsible for identifying an optimised expert placement strategy based on predicted expert routing.

The system maintains a queue of pending requests and the online reconfiguration manager uses this to predict the workload. Based on this prediction, the placement optimizer computes an updated placement to minimize the expected latency. Once the updated placement has been obtained, the live migration mechanism schedules the required migrations to minimize downtime and limit the impact on performance.

V. Online Expert Placement Reconfiguration Mechanism

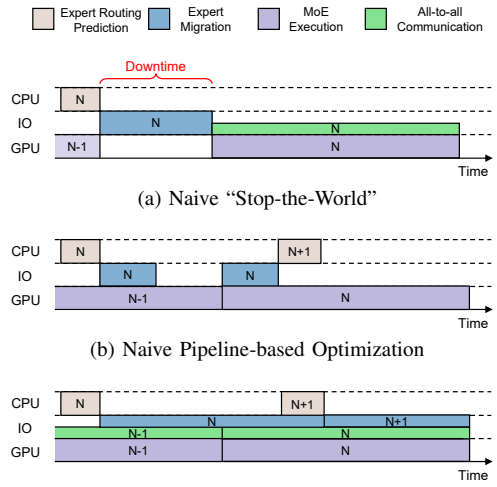
As mentioned in § I and § III, to maintain an effective placement, the first two challenges are: how to predict the expert routing for pending requests, and how to make expert migrations without interrupting the normal serving process. Therefore, we introduce an online expert placement reconfiguration mechanism. As illustrated in Figure 4, the mechanism operates as follows: first, it employs an adaptive predictor to forecast the expert activation patterns for pending requests in the queue at a low cost. Based on this prediction, it uses the algorithm from § VI to get an optimal placement. Finally, it migrates experts during idle periods in GPU communication.

A. Adaptive Routing Predictor

To accommodate various application scenarios, we provide two options for predicting expert activation patterns, as shown in Figure 5: *cascade predictor* and *quantized predictor*.

a) Cascade Predictor.: We observe a limitation in the naive predictor in § III that maps token embeddings to top-k expert choices: its predictions for deeper MoE layers become less accurate. This is because the routing in the deeper layers relies on the routing established previously. A simple predictor that only considers the initial embeddings lacks this conditional context, resulting in the accumulation of prediction errors across layers. Guided by this observation, we model routing as a conditional sequence of decisions. Accordingly, we adopt a cascade design that explicitly conditions later predictions on earlier ones. This design: (1) injects earlier routing predictions as context to curb error propagation in deeper layers, and (2) offers tunable depth to trade accuracy for overhead. As shown in Figure 5, we stack single-layer transformer predictors $\{M_1, M_2, \ldots, M_n\}$. M_1 consumes token embeddings and predicts the expert routing R_1; each subsequent M_i augments these embeddings with the predicted routing R_{i-1} from M_{i-1} to predict routing R_i for its designated block of subsequent layers.

b) Quantized Predictor.: For scenarios demanding maximum forecast fidelity, our second option is a low-bit quantized replica of the original MoE model. With the same model architecture and pre-trained knowledge [37], its gating networks generate expert rankings that are highly consistent with the full-precision model, ensuring reliable predictions. To manage the cost, we quantize both the replicas' weights and activations (e.g., to INT4). Although the model output might



(a) Naive "Stop-the-World"

(b) Naive Pipeline-based Optimization

(c) Our Pipelining

Fig. 6: Execution timelines of serving and reconfiguration.

be different, we validate that its predictions closely track the original model's routing in § VIII.

These two options offer a practical trade-off between accuracy and overhead: the cascade predictor prioritizes low latency for frequent, minor adjustments, while the quantized predictor ensures high fidelity for critical, major ones.

B. Computation-Overlapped Live Migration

To migrate the experts, A naive "stop-the-world" design, illustrated in Figure 6a, is inefficient because it halts all computation during the expert migration. A naive solution is to pipeline migration with computation. For example, the system can migrate the first half layers of experts while the experts in subsequent layers are processing tokens in Figure 6b. However, this approach overlooks a critical resource conflict: both the expert migration and the expert parallelism's all-to-all operations require communication bandwidth. Scheduling them at the same time can lead to congestion.

We thus design a fine-grained scheduling strategy. Its core idea is to prevent expert migration and all-to-all communication from executing concurrently, as shown in Figure 6c. As mentioned in § II, expert parallelism exists computation phase (attention, FFN) and the communication phase (all-to-all); our strategy schedules expert migrations to run only during computation phases when the communication network is idle. By leveraging the expert routing prediction, the module estimates the duration of the computation and communication windows, ensuring the migration completes before the next all-to-all operation begins. This approach effectively hides the migration overhead, ensuring the expert placement reconfiguration does not impact the critical path of inference.

VI. Relaxation-Based Expert Placement Algorithm

A. System Model

a) GPU Cluster: We model a cluster as a set of GPUs $\mathcal{G}$. The maximum number of experts per GPU per layer due to memory limitations is denoted by $C_{\max}$.

b) Expert Placement: The MoE model consists of L MoE layers, each having E experts, indexed from 1 to E. The placement map P assigns each expert to a host GPU. For example, $P(l,i)$ denotes the GPU hosting expert i in layer l.

c) Request Workload: For each request awaiting execution, the predictor in § V predicts the token-to-expert routing pattern. We use a *workload matrix*, W, to indicate the predicted workload. An element $W^l_{j,i}$ of this matrix indicates the number of tokens on GPU j, routed to expert i in layer l.

d) Computation Latency: The workload on a GPU is dictated by the total workload of the experts it hosts. Given the workload matrix W^l and a placement map P, the load on GPU j at layer l, denoted λ^l_j, is the sum of demands for all experts assigned to it:

$$\lambda^l_j(P) = \sum_{i=1}^{E} \mathbb{I}[P(l,i)=j] \left(\sum_{k\in\mathcal{G}} W^l_{k,i} \right), \tag{1}$$

where $\mathbb{I}[\cdot]$ is the indicator function. The layer's computation latency, T^l_{comp}, is determined by the straggler GPU. We use the function $f_{\text{comp}}(\lambda)$, derived from offline profiling, to denote the time required to process λ tokens. The total computation latency is:

$$T_{\text{comp}}(P) = \sum_{l=1}^{L} \left(\max_{j\in\mathcal{G}} f_{\text{comp}}\left(\lambda^l_j(P)\right) \right). \tag{2}$$

e) Communication Latency: Let d be the data size for a single token. For a given expert placement P, the total data volume to be transferred from GPU j to GPU k at layer l is denoted by $D^l_{j,k}(P)$, calculated as:

$$D^l_{j,k}(P) = \sum_{i:P(l,i)=k} d \cdot W^l_{j,i}. \tag{3}$$

The transmission latency between GPUs is modelled by the function $f_{\text{comm}}(D^l_{j,k}(P))$. It is empirically profiled and is sensitive to the network topology (e.g., intra-node vs. inter-node communication). The layer's overall communication latency is dictated by the straggler, calculated as:

$$T_{\text{comm}}(P) = \sum_{l=1}^{L} \left(\max_{j\in\mathcal{G}} \sum_{k\neq j} f_{\text{comm}}\left(D^l_{j,k}(P)\right) \right). \tag{4}$$

B. Problem Formulation

Our goal is to find an expert placement that minimizes the total latency of computation and communication:

$$\begin{aligned} \arg\min_{P} \quad & T_{\text{comp}}(P) + T_{\text{comm}}(P) \\ \text{s.t.} \quad & \sum_{i=1}^{E} \mathbb{I}[P(l,i)=j] \le C_{\max}, \quad \forall j, \forall l. \end{aligned} \tag{5}$$

Solving this problem is challenging, since it involves a trade-off between co-locating experts to save communication and distributing them to balance computation. Moreover, the

problem is NP-hard, reducible to the Generalized Assignment Problem (GAP) [19]—and features an immense search space that thwarts general heuristic algorithms.

C. Algorithm Design

To solve this challenging problem, our proposed algorithm finds a near-optimal placement in two main phases. First, we transform the problem into a feasibility problem and use a binary search over the total latency budget, β, to find the minimum achievable latency. At each search step, we solve a Linear Program (LP) relaxation of the problem. Second, after finding the optimal fractional solution $\bar{x}_{\text{best}}$ corresponding to the tightest budget, we employ an iterative rounding scheme to convert it into a final integer expert placement P. The overall process is outlined in Algorithm 1.

a) LP Relaxation with Linearization: We define a binary variable $x_{l,i,j} \in \{0,1\}$ to indicate if expert i in layer l is placed on GPU j. To create a tractable LP, we first express the computational load and communication data volume using these variables. Let $\lambda_j^l(x) = \sum_{i=1}^{E} x_{l,i,j}(\sum_{k\in\mathcal{G}} W_{k,i}^l)$ be the expected computational load on GPU j at layer l. Similarly, let $D_{j,k}^l(x) = d \cdot \sum_{i=1}^{E} W_{j,i}^l x_{l,i,k}$ be the expected data volume transferred from GPU j to k.

We then linearize the overall non-linear objective function by introducing auxiliary variables for the *max* operators and employing a standard piecewise linear approximation for the latency functions f_{comp} and f_{comm}. For a given latency budget β, the resulting feasibility LP is defined as:

$$\text{find} \quad x \tag{6a}$$

$$\text{s.t.} \quad \sum_{l=1}^{L}(\tau_{\text{comp}}^l + \tau_{\text{comm}}^l) \le \beta \tag{6b}$$

$$\tau_{\text{comp}}^l \ge f_{\text{comp}}\left(\lambda_j^l(x)\right), \qquad \forall l, j \tag{6c}$$

$$\tau_{\text{comm}}^l \ge \sum_{k\neq j} f_{\text{comm}}\left(D_{j,k}^l(x)\right), \qquad \forall l, j \tag{6d}$$

$$\sum_{j\in\mathcal{G}} x_{l,i,j} \ge 1, \qquad \forall l, i \tag{6e}$$

$$\sum_{i=1}^{E} x_{l,i,j} \le C_{\max}, \qquad \forall l, j \tag{6f}$$

$$0 \le x_{l,i,j} \le 1, \qquad \forall l, i, j \tag{6g}$$

where τ_{comp}^l and τ_{comm}^l are auxiliary variables representing the computation and communication latency at layer l.

b) Iterative Rounding: After obtaining the optimal fractional solution $\bar{x}_{\text{best}}$, we adapt an iterative rounding design to derive a final integer placement. The procedure is a randomized algorithm that performs a constrained "random walk" in the solution space. As proved in §VII, this process guarantees a near-optimal integral solution in polynomial time.

The rounding process consists of $O(\log N_{\text{vars}})$ (see § VII) macro-iterations. In each, we further reduce the number of fractional variables. The core mechanisms, which are essential for the subsequent analysis, are detailed below.

Algorithm 1: Relaxation-based Placement Algorithm

Input: Workload matrices $\{W^1, W^2, \ldots, W^L\}$, GPU set $\mathcal{G}$, capacity $C_{\max}$
Output: Optimal expert placement map P

Initialize the budget bound $(\beta_{\text{low}}, \beta_{\text{high}})$;
$\bar{x}_{\text{best}} \leftarrow$ null;
while β_{high} - $\beta_{low} > \epsilon$ **do**
  $\beta \leftarrow (\beta_{\text{low}} + \beta_{\text{high}})/2$;
  solve LP relaxation with given $\beta \rightarrow \bar{x}$;
  update search bounds $(\beta_{\text{high}}, \beta_{\text{low}})$;
get integer placement P $\leftarrow \bar{x}_{\text{best}}$ by iterative rounding;
return P;

1. Initialization and Variable Scaling: At the start of a macro-iteration, we classify each fractional variable x_v (where v is a multi-index for (l,i,j)) based on its value. For each variable, we define a discrete *scale* $s_v = 2^{-k}$ if its value x_v or $1 - x_v$ falls into the range $(2^{-(k+1)}, 2^{-k}]$. This scale is crucial as it dictates the magnitude of perturbation the variable will receive; variables closer to an integer value have a smaller scale and are moved more cautiously.

2. Constrained Random Walk: The random walk is performed within a carefully defined polytope Q that preserves feasibility. This polytope is defined by three sets of constraints: (i) the original LP constraints (Eq. 6), (ii) scale-based variable bounds of the form $x_v \le \alpha \cdot s_v$ (and $1 - x_v \le \alpha \cdot s_v$), where α is a constant, and (iii) an additional set of *scale-preserving constraints*. These take the form:

$$\sum_{v\in U_k} x_v = \sum_{v\in U_k} x_v^{(0)}, \quad \forall k \tag{7}$$

where U_k is the set of variables with the same scale $s_v = 2^{-k}$, and $x^{(0)}$ is the solution at the beginning of the macro-iteration. These constraints ensure that while individual variables are perturbed, their collective sum within a scale group remains constant, thus forcing some variables towards the integer boundaries of 0 or 1.

The walk proceeds for T_{micro} micro-steps, where T_{micro} is a sufficiently large polynomial in N_{vars}. At each step t, the update direction $\mathbf{g}^{(t)}$ is sampled from the null space of the tight constraints of Q. This direction is scaled such that variables with smaller scales are perturbed less. The solution is then updated by a small, fixed step size γ:

$$x^{(t+1)} \leftarrow x^{(t)} + \gamma \cdot \mathbf{g}^{(t)} \tag{8}$$

3. Integrality Guarantee and Termination: If any variable x_v exits the $[0,1]$ range during the walk, it is permanently "clamped" to 0 or 1. This is the primary mechanism that reduces the number of fractional variables. The convergence of this process is theoretically guaranteed by a *potential function* $\Phi(x)$, defined as:

$$\Phi(x) = \sum_{v \text{ is fractional}} s_v^{-2} \cdot \text{dist}(x_v, \{0,1\})^2 \tag{9}$$

where $\text{dist}(x_v, \{0,1\})$ is the distance of x_v to its nearest

integer. The algorithm is designed to ensure that the expected value of $\Phi(x)$ strictly increases with each random step. This provable progress guarantees that a constant fraction of variables become integral in each macro-iteration. After T_{micro} steps, the macro-iteration concludes.

VII. Analysis

In this section, we prove the approximation guarantee of Algorithm 1. To facilitate the following analysis, let $N_{\text{vars}} = L \cdot E \cdot |\mathcal{G}|$ denote the total number of decision variables.

Theorem 1. *A single macro-iteration of the* Iterative Rounding *scheme is a randomized polynomial-time procedure. It transforms a fractional solution x into a new fractional solution x' that simultaneously satisfies:*

1) ***Integrality Progress:*** *The number of fractional variables is reduced by a constant factor with high probability.*
2) ***Structural Preservation:*** *The new solution x' remains within the feasible placement polytope Q.*
3) ***Bounded Latency Violation:*** *The violation of any single linear latency constraint is controllably small and dependent on the step size γ.*

Lemma 1.1. *The rounding macro-iteration maintains feasibility with respect to all constraints defining the polytope Q.*

Proof. This property is guaranteed by construction. As defined in the algorithm description, the random walk is confined within the polytope Q. The update direction $\mathbf{g}^{(t)}$ is drawn from a subspace orthogonal to all tight constraints, ensuring the solution does not violate them. Any "clamping" of variables is a projection onto the boundary of Q. Thus, the updated solution x' remains feasible. □

We define the *scaled variance* of a linear constraint $\Lambda(x) = \mathbf{c}^T x$ as $\text{Var}_s(\Lambda) = \sum_v c_v^2 s_v^2$, where the sum is over the set of fractional variables.

Lemma 1.2. *For any linear latency constraint $\Lambda(x) = \mathbf{c}^T x$, the total violation $|\Lambda(x') - \Lambda(x)|$ in a single macro-iteration is bounded by $C_1 \cdot \gamma \cdot \sqrt{Var_s(\Lambda)}$ with high probability, for some constant C_1.*

Proof. The total change, $\Delta\Lambda = \sum_{t=0}^{T_{\text{micro}}-1} Z_t$, is a sum of martingale differences $Z_t = \gamma \mathbf{c}^T \mathbf{g}^{(t)}$. The conditional variance of each step is bounded by $\mathbb{E}[Z_t^2|\text{history}] \leq \gamma^2 \text{Var}_s(\Lambda)$. The total conditional variance over T_{micro} steps is thus $W \leq T_{\text{micro}} \gamma^2 \text{Var}_s(\Lambda)$. By applying martingale concentration inequalities, such as Freedman's Theorem [38]—a tool also central to the analysis in [39]—the probability of the total deviation $|\Delta\Lambda|$ exceeding a threshold proportional to $\gamma\sqrt{W/\gamma^2} = \gamma\sqrt{T_{\text{micro}} \text{Var}_s(\Lambda)}$ is exponentially small. A union bound over all latency constraints completes the argument, with C_1 absorbing terms like $\sqrt{T_{\text{micro}}}$. □

Claim 1.1. *For any linear latency constraint Λ and a solution with f fractional variables, the scaled variance is bounded: $Var_s(\Lambda) \leq C_2 \cdot f$, for a constant C_2.*

Proof. The scaled variance is a sum over the f fractional variables: $\text{Var}_s(\Lambda) = \sum_{v \in F} c_v^2 s_v^2$. The coefficients c_v are derived from the workload matrices and are bounded by a maximum value $c_{\max}$. The scales s_v are, by definition, at most $1/2$, so $s_v^2 \leq 1/4$. Thus, $\text{Var}_s(\Lambda) \leq \sum_{v \in F} c_{\max}^2 \cdot (1/4) = f \cdot (c_{\max}^2/4)$. Let $C_2 = c_{\max}^2/4$. □

Lemma 1.3. *Let F and F' be the sets of fractional variables before and after a single macro-iteration. With high probability, $|F'| \leq (1-\delta)|F|$ for some constant $\delta > 0$.*

Proof. The proof uses a potential function argument, similar to [40], [41]. The total expected increase in the potential function $\Phi(x)$ over a macro-iteration is lower-bounded: $\mathbb{E}[\Phi(x') - \Phi(x)] \geq T_{\text{micro}} \gamma^2 \mathbb{E}[\dim(\mathcal{V}')] - C_{\text{trunc}}$, where C_{trunc} is a small constant accounting for truncation effects.

The potential function $\Phi(x')$ is bounded above by $O(N_{\text{vars}})$ due to the scale-based constraints $x_v \leq \alpha \cdot s_v$ within the polytope Q. This implies that the average walk space dimension, $\mathbb{E}[\dim(\mathcal{V}')] = \mathbb{E}[|F| - |\mathcal{C}_{\text{tight}}|]$, must be small, which forces the expected number of tight constraints $\mathbb{E}[|\mathcal{C}_{\text{tight}}|]$ to be large.

Crucially, these tight constraints must correspond to variables becoming integral. The number of tight latency constraints is shown to be small by applying Lemma 1.2. The number of variables clamped to non-integral boundaries is limited by the scale-preserving constraints. As detailed in [39], these constraints imply that only a small fraction of variables can be clamped at non-integral boundaries. Therefore, the majority of tight constraints must originate from variables being clamped to $\{0, 1\}$, forcing a constant fraction of fractional variables to become integral. □

Proof of Theorem 1. The theorem's three properties follow directly from Lemmas 1.1, 1.2, and 1.3. The polynomial runtime follows from its construction from polynomially-solvable linear algebra operations. □

Theorem 2. *For any valid input, Algorithm 1 is a randomized polynomial-time procedure that, for any $\epsilon > 0$, returns an integer placement P with total latency $T(P) \leq (1+\epsilon)\beta^\star$ with high probability, where $\beta^\star$ is the optimal LP latency.*

Proof. We begin by solving the LP with a budget of $\beta' = (1 + \epsilon/2)\beta^\star$, creating an error slack of $(\epsilon/2)\beta^\star$. The rounding process is then executed for $T = O(\log N_{\text{vars}})$ macro-iterations. The final latency is $T(P) \leq \beta' + \sum_{t=1}^{T} \Delta_t$, where Δ_t is the latency violation from macro-iteration t.

We now bound the total accumulated violation. From Lemma 1.2 and Claim 1.1, the violation in a single step t (starting with f_{t-1} fractional variables) is bounded by:

$$\Delta_t \leq C_1 \cdot \gamma \cdot \sqrt{\text{Var}_s(\Lambda_t)} \leq C_1 \cdot \gamma \cdot \sqrt{C_2 \cdot f_{t-1}}$$

Let $C_3 = C_1\sqrt{C_2}$. From Lemma 1.3, we know that the number of fractional variables decreases geometrically: $f_t \leq (1-\delta) f_{t-1}$. Let $f_0 \leq N_{\text{vars}}$ be the initial number of fractional

MoE Model	Cascade		Quantization	
	Acc.(%)	Size (%)	Acc.(%)	Size (%)
Mixtral-8×7B	91.44	0.09	94.10	27.36
DeepSeekMoE-16B	84.81	0.07	90.98	29.51
DeepSeek-V2-Lite	84.62	0.08	95.68	35.62
Qwen3-30B-A3B	76.99	0.04	94.36	28.11

TABLE II: Expert routing prediction accuracy and predictor size relative to the served MoE for the two options

variables. The total violation is a sum over a converging geometric series:

$$\begin{aligned}\sum_{t=1}^{T}\Delta_t &\le \sum_{t=1}^{T} C_3\cdot\gamma\cdot\sqrt{f_{t-1}}\\ &\le C_3\cdot\gamma\cdot\sqrt{f_0}\sum_{k=0}^{T-1}(\sqrt{1-\delta})^k \\ &\le C_3\cdot\gamma\cdot\sqrt{N_{\text{vars}}}\cdot\frac{1}{1-\sqrt{1-\delta}}\end{aligned} \tag{10}$$

Let $C_4 = C_3\cdot(1-\sqrt{1-\delta})^{-1}$. The total error is bounded by $C_4\cdot\gamma\cdot\sqrt{N_{\text{vars}}}$. To ensure this error is bounded by the allocated slack, we can select the step size γ such that:

$$\gamma \le \frac{(\epsilon/2)\beta^\star}{C_4\sqrt{N_{\text{vars}}}}$$

This choice guarantees that the total added violation is no more than $(\epsilon/2)\beta^\star$. This trade-off between error magnitude and runtime (a smaller γ may require a larger polynomial T_{micro}) is a standard component of such rounding frameworks [42].

Consequently, the final latency is bounded by $T(P) \le (1+\epsilon/2)\beta^\star + (\epsilon/2)\beta^\star = (1+\epsilon)\beta^\star$. □

Theorem 3. *Algorithm 1 runs in polynomial time.*

Proof. The algorithm consists of two phases. The binary search phase requires only a logarithmic number of calls to a standard polynomial-time LP solver. The subsequent iterative rounding phase is also fast, consisting of a logarithmic number of macro-iterations, each involving only efficient linear algebra operations. Since both sequential phases run in polynomial time, the time complexity of Algorithm 1 is polynomial. □

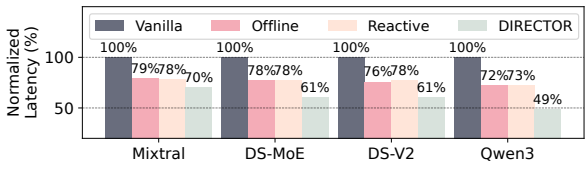
VIII. Evaluation

A. Implementation

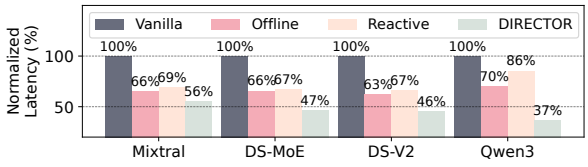
We implement our DIRECTOR prototype in Python on top of Colossal-AI [43]. The relaxation-based placement optimizer solves LP using SciPy [44], and we use Activation-aware Weight Quantization (AWQ) [45] for 4-bit quantization. Besides, the PyTorch version is 2.3, and the CUDA version is 12.4. The communication backend is based on NCCL.

B. Experimental Setup

Testbed. We evaluate DIRECTOR on two representative clusters with heterogeneous hardware configurations: (1) Cluster A: 4 nodes, each with 8 NVIDIA RTX 4090 GPUs; GPUs are connected by PCIe 4.0 within a node and by InfiniBand across nodes. (2) Cluster B: 4 nodes, each with 8 NVIDIA H200 GPUs; GPUs are connected by NVLINK within a node and by InfiniBand across nodes.



(a) RTX 4090 Cluster



(b) H200 Cluster

Fig. 7: End-to-end normalized latency of DIRECTOR against baselines on different models and clusters.

Models and Datasets. We evaluate DIRECTOR on four representative MoE models: Mistral 8×7B [46], DeepSeekMoE-16B [5], DeepSeek-V2-Lite [6], and Qwen3-30B-A3B [4]. Selected models cover a wide spectrum of configurations: the number of experts per layer varies from 8 to 128, and the top-k routing from 2 to 8 (details in Table I). To emulate a realistic multi-task scenario, we sample prompts from three widely used datasets: MATH500 [34], WikiText-103 [33], and Live Code Bench [35].

Baselines. We compare DIRECTOR with following baselines: (1)Vanilla: the default expert-placement strategy in DeepSpeed-MoE [47]. (2) Offline (static) placement: an optimal placement is computed once from historical traces and kept fixed during serving, as in [13]. (3) Online reactive placement: the system observes expert-routing patterns of the current and recent batches and adjusts placement accordingly at runtime, as in [14], [15], [17].

Metrics. We evaluate DIRECTOR and all baselines using three metrics, reporting the mean over five random seeds for each experiment. (1) *End-to-end latency*: the wall-clock time from a request's arrival in the queue to its completion, capturing both computation and any migration overhead. (2) *Throughput*: the number of tokens served per second under steady load, reflecting aggregate system efficiency. (3) *Prediction accuracy*: the top-k match rate between the predictor's forecasted experts and the ground-truth routing.

C. Results

Table II summarizes the accuracy and footprint of our two predictor modes. The cascade predictor (two-stage) achieves $77 \sim 91\%$ accuracy with a minimal footprint ($< 0.1\%$). Accuracy peaks on Mixtral (8 experts) but declines on complex routing tasks, where a deeper cascade (§ V-A) can maintain performance. Alternatively, the quantized predictor (4-bit BF16 replica) reaches $91 \sim 96\%$ accuracy, nearly matching full-precision routing. These results reveal a trade-off: the

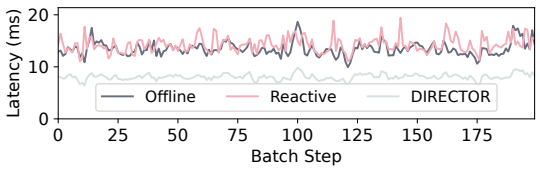


Fig. 8: The batch-level latency traces under different expert placement approaches on H200 cluster.

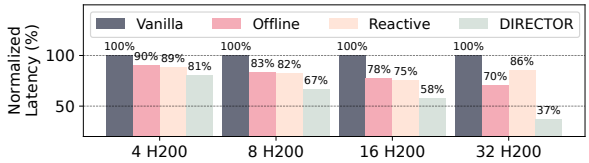


Fig. 9: End-to-end normalized latency of DIRECTOR against baselines on different number of GPUs.

cascade predictor offers high efficiency, while the quantized predictor ensures maximum fidelity.

Figure 7 compares the end-to-end latency of DIRECTOR against baselines. Our approach consistently achieves the lowest latency, yielding $11\% \sim 55\%$ gains over the offline baseline. This advantage grows with routing complexity; as experts increase (from 8 in Mixtral to 128 in Qwen3), load imbalance risks—resolved by our proactive placement—worsen. These gains are amplified on the H200 cluster, where high bandwidth reduces communication penalties, allowing DIRECTOR to focus on balancing computational load. We also note that baselines are limited by their reactive nature, though the offline approach generally outperforms the myopic reactive strategy due to its longer profiling window.

To analyze robustness, Figure 8 shows the per-batch latency trace. All methods show similar fluctuations, suggesting intrinsic demand variations. However, the reactive strategy exhibits sharper spikes than offline, as its myopic focus makes it vulnerable to transient extremes. Conversely, DIRECTOR demonstrates superior stability, achieving the lowest latency and variance, which indicates robust performance under dynamic loads.

We further evaluate scalability by measuring latency as H200 GPUs increase from 4 to 32. As shown in Figure 9, DIRECTOR's advantage becomes more pronounced at scale. Latency reduction over the offline baseline grows from 11% (4 GPUs) to 48% (32 GPUs). This highlights a key insight: larger systems are more sensitive to load imbalance. A single mismanaged expert can create a straggler that bottlenecks more devices, amplifying suboptimal placement costs. DIRECTOR's proactive approach effectively mitigates this, whereas baseline efficacy diminishes as the system scales.

D. Ablation Study

To quantify the performance impact of predictor inaccuracy and validate the effectiveness of our relaxation-based placement algorithm, we conduct the following two ablation studies.

First, we evaluate how predictor fidelity affects overall performance. We compare our two predictor options from

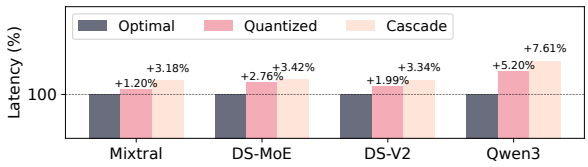


Fig. 10: Performance under different predictor options.

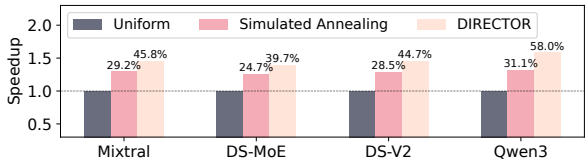


Fig. 11: Performance under various optimization algorithms.

§ V-A against an oracle with ground-truth routing on a 32-GPU RTX 4090 cluster, with results summarized in Figure 10. Across the four tested models, the performance gap between our predictors and the oracle is modest, ranging from 1% to 8%. The cascade predictor results in a performance gap of 3% to 7%, while the gap from the quantized predictor is almost negligible at 1% to 6%. This small performance gap demonstrates that our predictors achieve sufficient accuracy and do not introduce a significant performance gap.

Second, we evaluate our relaxation-based placement algorithm against a no-prediction baseline and a Simulated Annealing (SA) heuristic. SA is a common heuristic for large optimization spaces, and we ensure it runs for sufficient iterations to guarantee a fair comparison. As shown in Figure 11, our algorithm achieves a substantial 39% to 58% performance improvement over the baseline. In contrast, the SA heuristic, despite an extensive search, yields a more modest gain of 24% to 32%. This result demonstrates that our relaxation-based approach consistently finds higher-quality solutions than a well-established heuristic, highlighting its effectiveness in navigating the complex placement landscape.

IX. CONCLUSION

This paper proposes DIRECTOR, a distributed MoE serving system designed to overcome the performance limitations of existing offline and reactive placement strategies under dynamic workloads. DIRECTOR materializes this proactive paradigm through its key technical contributions: it employs adaptive predictors to predict routing from pending requests; utilizes a relaxation-based optimizer to find a near-optimal placement in polynomial time with a provable approximation guarantee; and enacts reconfigurations via a computation-overlapped migration module to ensure near-zero downtime. Through extensive evaluations, including targeted ablation studies that validate each design component, our prototype demonstrates that this proactive approach reduces end-to-end latency by a significant $11 - 55\%$ compared to existing systems. In future work, we plan to generalize our proactive placement paradigm beyond expert parallelism, applying its principles to other parallel dimensions.

REFERENCES

[1] Meta AI Team, "The llama 4 herd: Native multimodality with a mixture-of-experts architecture," https://ai.meta.com/blog/llama-4-multimodal-intelligence/, April 2025.

[2] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[3] Qwen Team, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[4] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[5] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu *et al.*, "Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models," in *ACL*, 2024.

[6] A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[7] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

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

[9] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, 2022.

[10] J. Zhou, Y. Chen, Z. Hong, W. Chen, Y. Yu, T. Zhang, H. Wang, C. Zhang, and Z. Zheng, "Training and serving system of foundation models: A comprehensive survey," *IEEE Open Journal of the Computer Society*, 2024.

[11] 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.

[12] Deepseek Team, "Expert Parallelism Load Balancer," 2025. [Online]. Available: https://github.com/deepseek-ai/EPLB

[13] S. Go and D. Mahajan, "Moetuner: Optimized mixture of expert serving with balanced expert placement and token routing," *arXiv preprint arXiv:2502.06643*, 2025.

[14] 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 *PPoPP*, 2022.

[15] 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," in *SIGMOD*, 2023.

[16] 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 *MLSys*, 2023.

[17] 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 *ATC*, 2023.

[18] J. Liu, J. H. Wang, and Y. Jiang, "Janus: A unified distributed training framework for sparse mixture-of-experts models," in *SIGCOMM*, 2023.

[19] M. Savelsbergh, "A branch-and-price algorithm for the generalized assignment problem," *Operations research*, 1997.

[20] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," in *ICLR*, 2017.

[21] X. Lu, Q. Liu, Y. Xu, A. Zhou, S. Huang, B. Zhang, J. Yan, and H. Li, "Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models," in *ACL*, Aug. 2024.

[22] E. Liu, J. Zhu, Z. Lin, X. Ning, M. B. Blaschko, S. Yan, G. Dai, H. Yang, and Y. Wang, "Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs," *arXiv preprint arXiv:2407.00945*, 2024.

[23] C. Chen, M. Li, Z. Wu, D. Yu, and C. Yang, "Ta-moe: topology-aware large scale mixture-of-expert training," in *NIPS*, 2022.

[24] 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 *EuroSys*, 2024.

[25] 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 *ASPLOS*, 2025.

[26] S. Shi, X. Pan, X. Chu, and B. Li, "Pipemoe: Accelerating mixture-of-experts through adaptive pipelining," in *INFOCOM*, 2023.

[27] Z. Fang, Y. Huang, Z. Hong, Y. Lyu, W. Chen, Y. Yu, F. Yu, and Z. Zheng, "Klotski: Efficient mixture-of-expert inference via expert-aware multi-batch pipeline," in *ASPLOS*, 2025.

[28] X. Pan, W. Lin, S. Shi, X. Chu, W. Sun, and B. Li, "Parm: Efficient training of large sparsely-activated models with dedicated schedules," in *INFOCOM*, 2024.

[29] X. He, S. Zhang, Y. Wang, H. Yin, Z. Zeng, S. Shi, Z. Tang, X. Chu, I. Tsang, and O. Y. Soon, "Expertflow: Optimized expert activation and token allocation for efficient mixture-of-experts inference," *arXiv preprint arXiv:2410.17954*, 2024.

[30] X. Liu, Y. Wang, F. Fu, X. Miao, S. Zhu, X. Nie, and C. Bin, "Netmoe: Accelerating moe training through dynamic sample placement," in *ICLR*, 2025.

[31] F. Chen, P. Li, Z. Hong, Z. Su, and S. Guo, "Communication-efficient sparsely-activated model training via sequence migration and token condensation," *IEEE Transactions on Networking*, 2024.

[32] J. Zhang, C. Ma, X. Wang, Y. Nie, Y. Li, Y. Xu, X. Liao, B. Li, and H. Jin, "{PopFetcher}: Towards accelerated {Mixture-of-Experts} training via popularity based {Expert-Wise} prefetch," in *ATC*, 2025.

[33] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016.

[34] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, "Let's verify step by step," in *ICLR*, 2023.

[35] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, "Livecodebench: Holistic and contamination free evaluation of large language models for code," *arXiv preprint arXiv:2403.07974*, 2024.

[36] Z. Du, S. Li, Y. Wu, X. Jiang, J. Sun, Q. Zheng, Y. Wu, A. Li, H. Li, and Y. Chen, "Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models," 2024.

[37] H. Wang, Q. zhou, Z. Hong, and S. Guo, "D2moe: Dual routing and dynamic scheduling for efficient on-device moe-based llm serving," in *Mobicom*, 2025.

[38] D. A. Freedman, "On tail probabilities for martingales," *the Annals of Probability*, pp. 100–118, 1975.

[39] N. Bansal and V. Nagarajan, "Approximation-friendly discrepancy rounding," in *IPCO*, 2016.

[40] N. Bansal, "Constructive algorithms for discrepancy minimization," in *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. IEEE, 2010, pp. 3–10.

[41] S. Lovett and R. Meka, "Constructive discrepancy minimization by walking on the edges," *SIAM Journal on Computing*, vol. 44, no. 5, pp. 1573–1582, 2015.

[42] L. C. Lau, R. Ravi, and M. Singh, *Iterative methods in combinatorial optimization*. Cambridge University Press, 2011, vol. 46.

[43] 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," 2023.

[44] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, 2020.

[45] J. Lin, J. Tang, H. Tang, S. Yang, G. Xiao, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *GetMobile: Mobile Comp. and Comm.*, Jan. 2025.

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

[47] 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 *ICML*, 2022.