CHUNKBALANCE: CHUNK REDISTRIBUTION FOR DATA-PARALLEL BALANCING IN MOE PREFILL

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models are typically deployed with data parallelism (DP) and expert parallelism (EP). In real-world MoE prefill scenarios, where requests are long and exhibit substantial variation in arrival rates and sequence lengths, DP-EP deployment can suffer from severe DP imbalance caused by two synchronous all-to-all communication barriers, leading to degraded system performance.

To address this problem, we propose ChunkBalance, a data-parallel balancing method based on chunk redistribution for MoE prefill. ChunkBalance is built on two-chunk splitting and compute-communication overlap. We observe that: (1) unlike EP workloads, which are bound to specific ranks, DP workloads can be flexibly redistributed across the DP group; and (2) synchronization occurs only at the boundaries between the DP and EP stages, while there is no synchronization barrier within the DP stage. Based on these observations, ChunkBalance redistributes the DP workloads of the second chunk across the DP group, allowing workloads across chunks to compensate for one another and thereby improving load balance. ChunkBalance operates offline and can be combined with request-level scheduling algorithms and heterogeneous parallelism configurations. Its communication overhead is nearly 2/3 smaller than the total intermediate-state size, while the remaining overhead can be effectively hidden through self-overlapping. Experiments across various workload and system configurations show that ChunkBalance reduces MoE prefill end-to-end latency by up to 40%.

1 INTRODUCTION

Mixture-of-Experts (MoE) architectures (Jacobs et al., 1991) have become increasingly popular for scaling the capacity of Large Language Models (LLMs) while keeping the computation per token manageable. Unlike dense models, an MoE layer contains multiple expert networks and a router that dynamically assigns each token to a small subset of experts, which enables substantially larger parameter capacity without proportionally increasing computation (Fedus et al., 2022).

Deploying MoE models at scale typically combines data parallelism (DP) and expert parallelism (EP). DP replicates the model parameters, except for the experts, across multiple workers to process different requests concurrently, while EP distributes experts across specific ranks to increase the effective model capacity. Since experts account for the majority of model parameters, an EP group typically consists of multiple DP subgroups to provide concurrency, increase expert compute intensity, and enable parameter scaling (Kwon et al., 2023; SGLang Team, 2024). For each MoE layer, tokens routed to experts that reside on remote devices are dispatched to the corresponding workers through an all-to-all communication. After expert computation, the processed tokens are exchanged back through a second all-to-all and subsequently aggregated. Consequently, each MoE layer introduces two communication phases, along with the corresponding synchronization barriers, which can significantly affect execution efficiency (Huang et al., 2024; Hwang et al., 2023).

For MoE prefill, the communication overhead can be largely mitigated by chunk pipelining, which interleaves the computation and communication of multiple chunks to concurrently utilize GPU compute and memory bandwidth (Liu et al., 2024), yet the synchronization barriers remain problematic. In real-world scenarios, requests can be long and arrive at highly variable rates and sequence lengths, resulting in substantial workload variation across DP replicas over time (Xiang et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

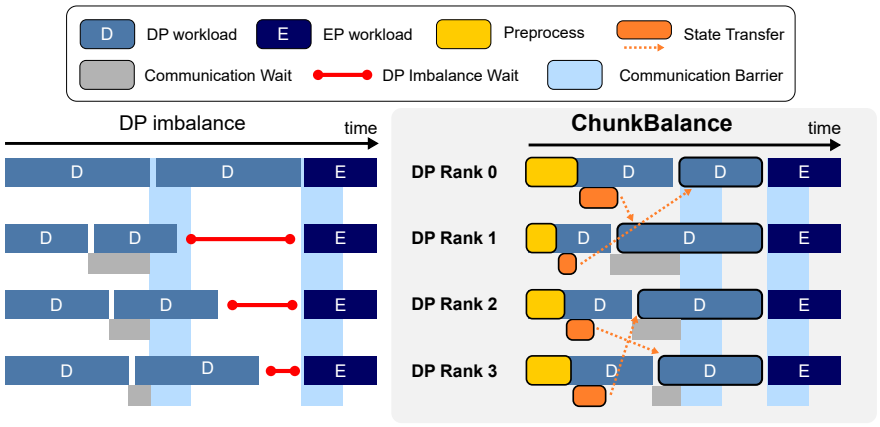

Figure 1: Demonstration of DP imbalance and overview of ChunkBalance. The workload imbalance across the DP group causes idling by the sychronization barriers, while ChunkBalance mitigates it by redistributing the second chunk across ranks.

Under DP-EP deployment, these variations lead to computational imbalance: the synchronization barriers force all workers to wait for the slowest rank, while faster ones remain idle at the barriers, resulting in GPU under-utilization and degraded system performance.

Existing approaches to mitigating DP imbalance primarily rely on request-level scheduling (Sun et al., 2024; Wu et al., 2024), which aims to distribute prefill requests evenly across DP replicas as they arrive. Despite their effectiveness, these approaches are inherently limited by their request-level granularity and lack of awareness of model execution. In practice, requests may be difficult to place optimally due to their varying lengths (Choi et al., 2021). Moreover, request placement may be constrained by request priorities and SLO requirements, further limiting scheduling flexibility. Consequently, request-level scheduling alone may fail to fully eliminate DP imbalance and can result in suboptimal workload distribution.

To address these limitations, we propose ChunkBalance, a chunk redistribution method for data-parallel balancing in MoE prefill. ChunkBalance operates within model execution and is built upon two-chunk pipelining. Our key insights are as follows:

(1) DP workloads are freely redistributable, unlike EP workloads. EP workloads are bound to the expert-to-device mapping and therefore to specific EP ranks, whereas DP workloads can be flexibly redistributed across the DP group because model parameters are replicated across DP replicas.

(2) Workloads of two chunks within a DP-EP stage do not require synchronization. Synchronization occurs at the boundaries between DP and EP stages, but workloads of two chunks can proceed independently within each stage (Wu et al., 2026). Specifically, consider two chunks of a request batch, whose DP-stage workloads are denoted by $w_0, w_1 \in \mathbb{R}^d$, where d is the DP size. Since the two chunks can execute independently without synchronization, their workloads on each DP rank accumulate, resulting in an aggregate workload of $w_0 + w_1$ across the DP group.

Based on these observations, ChunkBalance redistributes the DP workloads of two chunks differently to improve load balance. For the workloads of the two chunks be $w_0 = (w_{00}, \ldots, w_{0d})$ and $w_1 = (w_{10}, \ldots, w_{1d})$, where w_{ki} denotes the workload of chunk k assigned to DP rank i, ChunkBalance applies a permutation σ to the workload of the second chunk as $w_1' = (w_{1\sigma(0)}, \ldots, w_{1\sigma(d)})$, seeking a more balanced aggregate workload across DP ranks that $\max(w_0 + w_1') < \max(w_0 + w_1)$.

ChunkBalance operates offline and introduces no scheduling overhead to the model-forwarding process. As a DP-level balancing mechanism, ChunkBalance is orthogonal to request-level scheduling and heterogeneous parallelism configurations such as tensor parallelism (TP) and pipeline parallelism (PP). If a request sequence is split into half during chunking, DP rank i needs to communicate with rank $\sigma(i)$ to send sequence-dependent intermediate states (e.g., KV caches), while the additional communication overhead can be effectively self-overlapped with computation. Experi-

ments across various scenarios show that ChunkBalance acheives constant effectiveness, reducing MoE prefill Time-To-First-Token by up to 40%.

2 PRELIMINARIES

2.1 MIXTURE-OF-EXPERT ARCHITECTURE

A transformer layer typically consists of two main modules: self-attention and multi-layer perceptron (MLP) (Vaswani et al., 2017). Given the input hidden state h_l of layer l,

A Mixture-of-Expert (MoE) transformer layer typically consists of two main modules: attention module and expert module. The output of the two modules is typically aggregated with residual connections (He et al., 2016) as:

$$h'_l = h_l + \text{Attention}(h_l), \quad h_{l+1} = h'_l + \text{MoE}(h'_l)$$

The expert module are sparsely activated through a routing mechanism. For each token, the router selects the top-k experts according to the gating function G:

$$W_l, I_l = \text{topk}(G(h'_l)).$$

Each selected expert processes the token independently, and their outputs are aggregated using a weighted sum:

$$\text{MoE}(h'_l) = \sum_{i \in I_l} W_{l,i} * \text{expert}_i(h'_l),$$

where $W_{l,i}$ denotes the routing weight for expert i. The aggregated result is then passed to the next layer as the output of the MoE module.

2.2 DISTRIBUTED LLM INFERENCE

LLM inference systems typically distribute computation across multiple devices to scale memory capacity and computational throughput. Data Parallelism (DP) replicates the model across multiple devices and partitions input requests or batches among replicas to increase aggregate throughput, while Tensor Parallelism (TP) partitions individual model operators across devices so that each device computes a portion of the same layer (Shoeybi et al., 2019; Zhao et al., 2022). Pipeline Parallelism (PP) partitions the model layers into sequential stages and assigns different stages to different devices (Huang et al., 2019; Shoeybi et al., 2019). Expert Parallelism (EP) instead distributes the experts of a Mixture-of-Experts (MoE) layer across devices and routes tokens to their selected experts (Lepikhin et al., 2021; Rajbhandari et al., 2022).

2.2.1 DP-EP DEPLOYMENT

Mixture-of-Experts (MoE) models are commonly deployed using a combination of Data Parallelism (DP) and Expert Parallelism (EP). EP distributes experts across GPUs, allowing the system to scale the model's parameter capacity without requiring every GPU to store all experts. Meanwhile, DP replicates the EP-sharded model across multiple groups, increasing aggregate compute capacity and allowing requests to be processed concurrently. The combination of DP and EP therefore provides a natural way to scale both parameter capacity and inference throughput.

Within each MoE layer, tokens are routed to their selected experts. When an expert is located on a different EP rank, the corresponding tokens are dispatched to the destination rank through an all-to-all communication, processed by the local experts, and then exchanged back through a second all-to-all communication. Consequently, DP-EP execution introduces synchronization points around the expert dispatch and combine operations. Workload imbalance across DP replicas can therefore cause faster replicas to wait for slower replicas at these synchronization points, resulting in synchronization stalls and degraded inference performance.

2.2.2 PREFILL-DECODE DISAGGREGATION

LLM inference typically consists of two phases: prefill processes the input prompt and generates the initial KV cache, while decode autoregressively generates output tokens using the cached states. These two phases exhibit substantially different computational and memory characteristics, and their resource requirements can interfere when they share the same workers (Yu et al., 2022). Prefill-decode (PD) disaggregation separates the two phases onto different sets of GPUs or inference instances. By independently scaling the two types of workers, PD disaggregation allows system resources and parallelism configurations to be tailored to the distinct characteristics of the two phases (Zhong et al., 2024). ChunkBalance targets the prefill stage, where long and heterogeneous input requests can lead to substantial workload imbalance across DP replicas.

2.3 CHUNK PIPELINING

Distributed inference introduces communication between devices. As model size and inference scale increase, communication overhead can become a significant performance bottleneck (Liu et al., 2024). Chunk pipelining mitigates this overhead by splitting the workload into smaller chunks and overlapping communication with computation, allowing communication for one chunk to proceed while another chunk is being computed. In a DP-EP deployment, the dispatch and combine all-to-all operations can similarly be overlapped with attention and MoE computation through fine-grained or microbatch-level scheduling (vLLM Project, 2025).

3 METHOD

3.1 PROBLEM AND MOTIVATION

As discussed above, DP imbalance arises from variations in request workloads and the synchronization barriers inherent in DP-EP execution. At each barrier, all DP ranks must synchronize with one another, and the system can proceed only after the slowest rank completes its workload. Consequently, faster ranks remain idle while waiting for the straggler, resulting in synchronization stalls and degraded performance (Rajbhandari et al., 2022; Zhao et al., 2025).

Formally, consider a DP group of size d and two chunks with DP workloads $w_0 = (w_{00}, w_{01}, \ldots, w_{0(d-1)})$ and $w_1 = (w_{10}, w_{11}, \ldots, w_{1(d-1)})$, where w_{ki} denotes the workload of chunk k assigned to DP rank i. Without workload redistribution, the effective workload of DP rank i is$w_i = w_{0i} + w_{1i}$. Since the two chunks must synchronize at the DP-EP barriers, the execution time of the synchronized stage is determined by the maximum workload across DP ranks:

$$W = \max_{i \in [d]} w_i.$$

Existing request-level approaches typically address DP imbalance by careful request assignment to the DP ranks at the input stage, for example, by interleaving short and long requests. However, these approaches implicitly rely on the availability of requests with complementary workloads and sufficient scheduling flexibility to form a balanced request group. In practice, long and highly variable sequences can make it difficult to find requests with complementary lengths at the right time, and SLO constraints often limit how long requests can wait for batching (Choi et al., 2021). Consequently, the system may have to admit an inherently imbalanced workload despite careful request-level scheduling. This limitation motivates load balancing within model execution, where the system can compensate for imbalance after requests have entered the model rather than relying solely on future request arrivals.

We address the above limitations based on two key observations.

First, DP workloads can be flexibly redistributed across the DP group. Unlike EP workloads, which are bound to expert placement and therefore must execute on specific EP ranks, DP replicates the model parameters across replicas. As a result, DP ranks are symmetric with respect to model computation, allowing a workload originally assigned to one DP rank to be executed by another rank.

Second, in two-chunk pipelining (Liu et al., 2024), synchronization barriers do not require workloads of chunks within a DP stage to remain aligned. Although the dispatch and combine operations

form two synchronization boundaries, the computation between these boundaries consists of workloads from multiple chunks that can proceed independently without synchronization. Specifically, as shown in Figure 1, the communication operation of chunk 0 may incur waiting (grey bars), but does not introduce a synchronization barrier, as the computation of chunk 1 can proceed concurrently.

Based on these insights, ChunkBalance creates balancing opportunities by redistributing the DP workloads of chunks to compensate for workload differences across ranks while preserving the required DP-EP synchronization semantics.

3.2 Chunk Redistribution

Given the workloads of the two chunks, w_0 and w_1, ChunkBalance searches for a permutation of the DP assignments of one chunk to achieve a more balanced aggregate workload across DP ranks. Since all DP ranks are symmetric, we can leave the assignments of the first chunk unchanged and permute only those of the second chunk without loss of generality. Formally, let σ be a permutation of $[d]$ and the redistributed workload of the second chunk is

$$w_1' = \left(w_{1\sigma(0)}, w_{1\sigma(1)}, \ldots, w_{1\sigma(d-1)}\right). \tag{1}$$

The resulting workload of DP rank i is therefore

$$w_i = w_{0i} + w_{1\sigma(i)}.$$

ChunkBalance seeks the permutation that minimizes the maximum workload across DP ranks:

$$\sigma^* = \arg\min_{\sigma \in S_d} \max_{i \in [d]} \left(w_{0i} + w_{1\sigma(i)}\right), \tag{2}$$

where S_d denotes the set of all permutations of d DP ranks.

We conclude that σ^* can be constructed by the order across dimensions of workloads w_0 and w_1.

Theorem 1. *Let σ_0 and σ_1 be the ascending and descending workload permutations of w_0 and w_1 respectively, such that*

$$w_{0\sigma_0(i)} \le w_{0\sigma_0(i+1)} \quad \textit{and} \quad w_{1\sigma_1(i)} \ge w_{1\sigma_1(i+1)}$$

for all $0 \le i < d-1$. Then

$$\sigma^*(\sigma_0(i)) = \sigma_1(i)$$

Proof. Consider any permutation $\sigma \neq \sigma^*$. Since σ^* assigns the w_1 workloads in non-increasing order with respect to the non-decreasing order of w_0, if σ is not of this form, there exist indices $i < j$ such that

$$\sigma(\sigma_0(i)) = m, \sigma(\sigma_0(j)) = n,$$

and $w_{1m} \le w_{1n}$. We construct a new permutation σ' by swapping the assignments of these two positions:

$$\sigma'(\sigma_0(i)) = n, \qquad \sigma'(\sigma_0(j)) = m,$$

while keeping all other assignments unchanged.

Note that $\max_{i\in[d]} \left(w_{0i} + w_{1\sigma(i)}\right) = \max_{i\in[d]} \left(w_{0\sigma_0(i)} + w_{1\sigma(\sigma_0(i))}\right)$ for any σ since σ_0 is a permutation, so the maximum value of σ and σ' differ only in the relevant two terms. Before the swap,

$$A = w_{0\sigma_0(i)} + w_{1m}, \quad B = w_{0\sigma_0(j)} + w_{1n},$$

and after the swap,

$$A' = w_{0\sigma_0(i)} + w_{1n}, \quad B' = w_{0\sigma_0(j)} + w_{1m}.$$

Because $i < j$ and σ_0 orders w_0 non-decreasingly, We have (1) $A' \leq B$ since $w_{0\sigma_0(i)} \leq w_{0\sigma_0(j)}$, and (2) $B' \leq B$ as $w_{1m} \leq w_{1n}$.

Therefore,

$$\max(A', B') \leq B \leq \max(A, B).$$

All other terms remain unchanged, so

$$\max_{i\in[d]} \left(w_{0i} + w_{1\sigma'(i)}\right) \leq \max_{i\in[d]} \left(w_{0i} + w_{1\sigma(i)}\right).$$

We can repeatedly apply this exchange to eliminate every inversion in the sequence, and the resulting permutation assigns the w_1 workloads in non-increasing order, and hence is σ^*.

□

This formulation guarantees a solution no worse than the original permutation ($\hat{\sigma}(i) = i$), which corresponds to execution without chunk redistribution. Since DP workloads can typically be determined before inference starts, the redistribution plan can be computed entirely offline, introducing no runtime scheduling overhead during inference. The resulting plan consists only of a DP-sized permutation and can therefore be computed efficiently.

Furthermore, under chunk pipelining, the two workload splits typically satisfy $w_{0i} \approx w_{1i}$ for all i. In this case, the optimal permutations for the two chunks exhibit a symmetric relationship: $\sigma_1(i) = \sigma_0(d-1-i)$. Therefore, we can simply denote $\sigma = \sigma_0$, yielding

$$\sigma^*(\sigma(i)) = \sigma(d-1-i).$$

3.3 Intermediate State Communication

As the number and lengths of requests vary, a chunk may split a request across its boundary, e.g., $(r_0, r_{10} \mid r_{11}, r_2)$, where r_{10} and r_{11} are consecutive portions of the same request. In this case, the causal dependency between r_{10} and r_{11} requires the intermediate states (e.g., the KV cache for causal attention) produced by r_{10} to be migrated to the DP rank processing r_{11} after redistribution. The communication volume depends on how requests are split, but is typically much smaller than the total intermediate-state size of the chunk, since only the split request needs to migrate. For example, the communication volume is proportional to $|r_{10}|$, whereas the total KV-cache size of the chunk is proportional to $|r_0| + |r_1| + |r_2|$.

To support such request splits, we separate the preprocess phase from the DP workload of chunk 0, with the boundary determined by when the required intermediate states become ready for transfer. Once the intermediate states are ready, they are asynchronously transferred to the destination DP rank while other computations (e.g., attention) continue. Importantly, this communication can be overlapped with the postprocess computation of the chunk itself, because the latter does not depend on the intermediate states generated by the preceding portion. Therefore, the migration of intermediate states introduces negligible additional critical-path overhead.

4 Experiments

Models. We evaluate ChunkBalance on 3 representative open-source MoE models: Qwen3-30B-A3B-Instruct-2507 (Qwen3-30B) (Yang et al., 2025), Moonlight-16B-A3B-Instruct (Moonlight-16B) (Liu et al., 2025) and Qwen3-235B-A22B-Instruct-2507 (Qwen3-235B). Qwen3 uses multi-head attention (Cordonnier et al., 2020), while Moonlight uses multi-head latent attention (Liu et al., 2024).

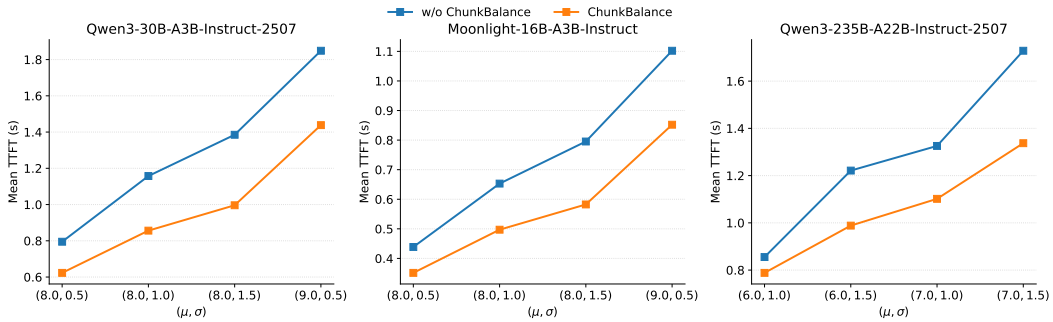

Figure 2: Mean Time-To-First-Token (TTFT) of Qwen-30B, Moonlight-16B and Qwen3-235B under various configurations of request-length distribution.

Configurations. All experiments are conducted on nodes equipped with 8×A800-SXM4 GPUs, each with 80 GB of memory, interconnected via NVLink. Unless otherwise specified, we use a data-parallel size of 8 with no tensor or pipeline parallelism. Request sequence lengths are sampled from a log-normal distribution parameterized by (μ, σ). We use different length configurations for the smaller models (Qwen3-30B and Moonlight-16B) and the larger model (Qwen3-235B). Specifically, sequence lengths are clipped to $[1024, 16384]$ for the smaller models and $[256, 4096]$ for the larger model to ensure sufficient compute intensity while respecting memory constraints. At each scheduling step, the number of newly scheduled requests is sampled uniformly from $U[d, 2d]$, where d is the DP size. We use round-robin request scheduling by default and evaluate other scheduling policies separately.

Benchmark. We use LongBench (Bai et al., 2024) as the evaluation benchmark, which contains large-scale, long-context inference workloads. We sample requests from question-answering tasks to form a synthesized set. When the tokenized input is shorter than the target sequence length sampled from the specified distribution, we pad the input sequence to the target length.

Metrics. We measure inference performance using the mean time-to-first-token (TTFT) per execution step. Specifically, we compute the overall TTFT across the workload and divide it by the total number of execution steps, yielding the average TTFT per step.

4.1 Effectiveness

Figure 2 presents the end-to-end latency and GPU under-utilization of the evaluated models across various request-length distributions. As shown, ChunkBalance consistently improves inference performance across models and configurations, reducing mean TTFT by up to 40% while significantly improving GPU utilization by mitigating DP imbalance. The benefits become more pronounced with larger μ and σ, where the request-length distribution exhibits greater variability.

4.2 Orthogonality

ChunkBalance is orthogonal to existing MoE inference optimization methods, including request-level scheduling and additional parallelism strategies such as tensor parallelism and pipeline parallelism.

4.2.1 Request-Level Scheduling

Existing request-level scheduling algorithms operate at the granularity of individual requests, whereas ChunkBalance operates at the granularity of model execution. To evaluate their compatibility, we consider two representative request-level scheduling policies.

Round-Robin. Requests are assigned to DP ranks in a round-robin manner.

Balance. Each incoming request is assigned to the DP rank with the smallest accumulated workload.

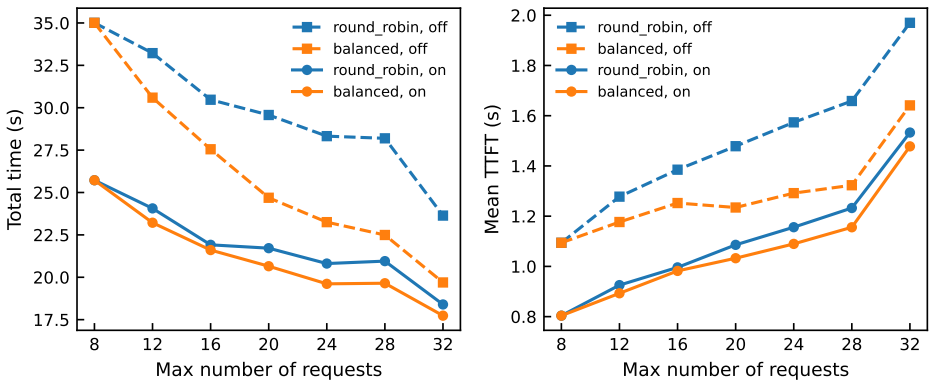

Figure 3: Total time (s) and mean TTFT of Qwen3-30B under various threshold of number of requests per step.

For both policies, we assume that each request is immediately assigned to a DP rank at the next scheduling step. Figure 3 shows the end-to-end performance of applying ChunkBalance on top of these scheduling policies. As demonstrated, ChunkBalance consistently provides additional gains under both policies, proving that request scheduling is still suboptimal. Notably, as the number of requests increases, request scheduling can improve the total time but increases per-batch latency, since more requests are put in a single forward. On the contrary, ChunkBalance can be robustly effective on both the total time and per-batch latency.

4.2.2 Further Parallelism

Beyond DP-EP deployment, tensor parallelism (TP) is commonly used to shard the parameters of the attention module and reduce per-batch latency, while pipeline parallelism (PP) enables the serving of large-scale models that cannot fit on a single device. ChunkBalance operates at the DP-EP level and is orthogonal to these additional parallelism dimensions. Specifically, we adopt a typical hierarchical parallel configuration in which each EP group spans multiple TP ranks, i.e., $\mathrm{EP} = \mathrm{DP} \times \mathrm{TP}$, while pipeline parallelism further partitions the model such that $\mathrm{EP} \times \mathrm{PP} = \mathrm{world\ size}$. As demonstrated in Table 1, ChunkBalance can be applied independently and effective within each DP-EP group under various configurations.

4.3 Communication Volume

We evaluate the KV communication volume relative to the total KV cache size, with the accumulated results for 256 requests shown in Table 2. When the number of requests equals the DP size, the communication volume is exactly 50% of the total KV cache size. As the number of requests increases beyond the DP size, the communication volume decreases further to nearly one-third of the total cache size, since only the chunked request requires communication.

5 Related Works

5.1 MoE Inference Systems

Efficient inference of Mixture-of-Experts (MoE) models has been studied from multiple system perspectives, including parallelism, communication optimization, and execution scheduling. DeepSpeed-MoE (Rajbhandari et al., 2022) develops an end-to-end MoE inference system that combines data parallelism, tensor parallelism, and expert parallelism, together with communication scheduling and model-level optimizations. Tutel (Hwang et al., 2023) further explores adaptive parallelism and pipelining for MoE workloads, dynamically adapting the execution strategy to the workload generated by sparse routing. Sarathi (Agrawal et al., 2023) introduces chunked prefill, partitioning long prefilling requests into smaller chunks and interleaving them with decoding

Table 1: Mean TTFT (s) of models under heterogeneous further parallelism configurations. The (μ, σ) is $(8.0, 1.5)$ and $(9.0, 0.5)$ for smaller and larger models respectively.

TP size	Qwen3-30B		MoonLight-16B		Qwen3-235B (PP size=2)	
1	1.84	**1.43**	1.10	**0.85**	5.50	**4.13**
2	1.09	**0.95**	0.63	**0.55**	5.28	**4.51**
4	0.61	**0.57**	0.39	**0.36**	4.90	**4.59**

Table 2: Sizes (GB) of KV Cache and communicated intermediate states of Qwen3-30B with 256 requests.

Cache size	8	12	16	20	24
95.14	47.57	41.53	36.87	34.94	33.94

to improve GPU utilization and mitigate the interference between prefill and decode workloads. Infinite-LLM (Lin et al., 2024) further addresses long-context serving through distributed KV-cache management and context parallelism, enabling requests with large contexts to be distributed across multiple devices. DeepEP (Zhao et al., 2025) instead focuses specifically on the communication layer of MoE serving, providing an optimized expert-parallel communication library for token dispatch and combine operations.

5.2 Balancing and Scheduling for MoE Serving

Request-level scheduling balances the workload by deciding which DP engine should process each incoming request. Typical policies use request counts, estimated token workloads, or runtime load information to distribute requests across replicas. Llumnix (Sun et al., 2024) performs dynamic request scheduling for LLM serving by continuously monitoring runtime system states and migrating requests to mitigate load imbalance. LoongServe (Wu et al., 2024) introduces elastic sequence parallelism for long-context LLM serving, dynamically adjusting the parallel execution resources assigned to individual requests according to their context lengths and runtime demands. NanoCP (Chen et al., 2026) proposes request-level dynamic context parallelism for data-expert-parallel decoding, dynamically adapting the context-parallel configuration of requests according to their runtime requirements.

ChunkBalance addresses the complementary problem of balancing workloads after requests have already been assigned to DP replicas, and exploits the fact that DP workloads are interchangeable across replicas and redistributes the workload of one chunk according to the workload of another chunk.

6 Conclusion

This paper proposes ChunkBalance, a chunk redistribution method for data-parallel balancing in DP-EP deployed MoE prefill. Its design is based on two key insights: first, unlike EP workloads that are bound to specific experts and devices, DP workloads can be freely redistributed across symmetric DP replicas; second, although dispatch and combine operations introduce synchronization boundaries between DP and EP stages, workloads from different chunks can execute asynchronously within a DP stage. Based on these observations, ChunkBalance apples a permutation of the DP assignments of the second chunk to minimize the maximum aggregate workload across DP ranks. Experiments on various scenarios show that ChunkBalance can achieve up to 40% end-to-end latency reduction.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, assist with translation and interpret results. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments. Generating synthetic data sets, cleaning and reformatting dataset and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for drafting parts of a research paper and polishing the text. We have reviewed all AI-assisted work. We checked LLM-generated mathematical claims and proofs, and LLM-generated code was verified and tested for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Jiefei Chen, Binbin Lin, Jinming Ma, Jiangfei Duan, Haojie Duanmu, Hao Liu, Qinxiu Cheng, Xiuhong Li, Zhilin Pei, Hui Wang, et al. Nanocp: Request-level dynamic context parallelism for data-expert parallel decoding. *arXiv preprint arXiv:2605.21100*, 2026.

Yujeong Choi, Yunseong Kim, and Minsoo Rhu. Lazy batching: An sla-aware batching system for cloud machine learning inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 493–506. IEEE, 2021.

Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*, 2020.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S Lee, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Toward efficient inference for mixture of experts. *Advances in Neural Information Processing Systems*, 37:84033–84059, 2024.

Yanping Huang, Youlong Cheng, Ankur Bapna, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *ICLR*, 2021.

Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, et al. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache. *arXiv preprint arXiv:2401.02669*, 2024.

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

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR*, pp. 18332–18346, 2022.

SGLang Team. Sglang: Efficient execution of structured generation programs. `https://github.com/sgl-project/sglang`, 2024.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. In *arXiv preprint arXiv:1909.08053*, 2019.

Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 173–191, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

vLLM Project. vllm: Expert parallel deployment. `https://docs.vllm.ai/en/latest/serving/expert_parallel_deployment.html`, 2025. Accessed September 2026.

Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, pp. 640–654, 2024.

Yize Wu, Ke Gao, Ling Li, and Yanjun Wu. Easybalance: Cross-layer load balancing in distributed moe inference. *arXiv preprint arXiv:2608.07964*, 2026.

Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, Wenyuan Yu, Ennan Zhai, Xin Jin, and Jingren Zhou. {ServeGen}: Workload characterization and generation of large language model serving in production. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 1845–1859, 2026.

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

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation*, pp. 521–538, 2022.

Chenggang Zhao, Shangyan Zhou, Liyue Zhang, Chengqi Deng, Zhean Xu, Yuxuan Liu, Kuai Yu, Jiashi Li, and Liang Zhao. Deepep: an efficient expert-parallel communication library. `https://github.com/deepseek-ai/DeepEP`, 2025.

Yan Zhao, Lianmin Guo, Zhi Qu, et al. Alpa: Automating inter- and intra-operator parallelism for distributed deep learning. In *OSDI*, 2022.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*, pp. 193–210, 2024.

A MORE RESULTS

A.1 WIDER RANGE OF CONFIGURATIONS

Figure 4 and Figure 5 present the mean TTFT of the evaluated models across a wider range of μ and σ configurations without length clipping. ChunkBalance demonstrates consistent effectiveness across these configurations, while its benefits become marginal when both μ and σ are sufficiently small, resulting in moderate variation in request lengths and making computation overhead less dominant in end-to-end latency.

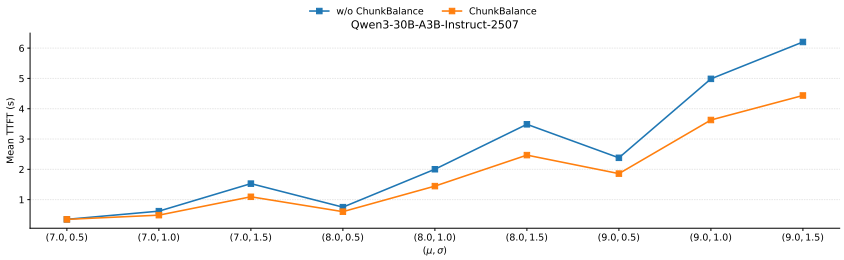

Figure 4: Mean TTFT of Qwen3-30B under various configurations of μ and σ.

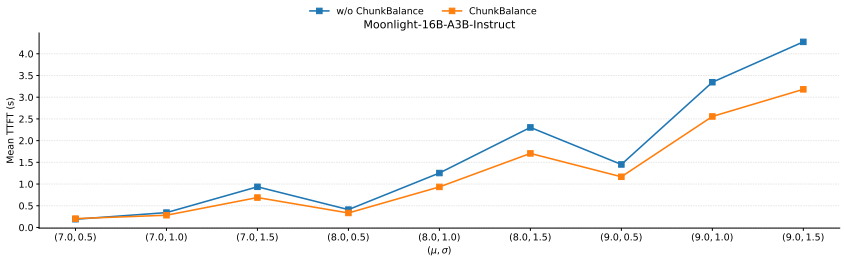

Figure 5: Mean TTFT of Qwen3-30B under various configurations of μ and σ.

A.2 REQUEST-LENGTH DISTRIBUTION

?? shows the request-length distribution under several representative combinations of μ and σ, illustrating how these parameters affect the shape and variability of the generated workloads.

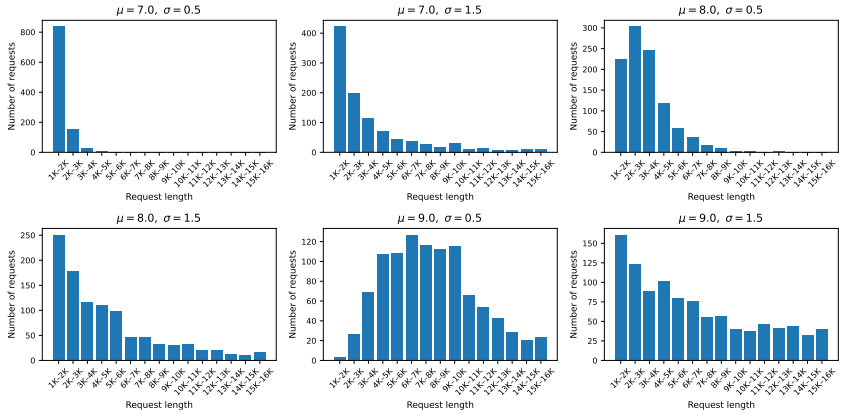

Figure 6: Request length distributions of various configurations of μ and σ.

B LIMITATIONS

ChunkBalance targets MoE prefill acceleration and is not directly applicable to decode inference, where two-chunk pipelining is not typically employed. Our current design is built around two-chunk pipelining; extending the approach to a larger number of chunks is possible in principle, but may require different redistribution strategies and is left for future work. Our experiments are conducted exclusively on the A800 platform. Although the underlying workload-redistribution mechanism is hardware-agnostic, its performance benefits may vary across hardware configurations due to differences in computation and communication characteristics. All experiments constrain the DP group to a single node, although some configurations employ cross-node pipeline parallelism. Extending ChunkBalance to multi-node DP groups may introduce higher redistribution communication costs.