FreeIO: Enabling Zero-Copy, Sub-RTT All-to-All Collective for MoE Inference

Paper #, pages body, pages total

Abstract

aaa

Keywords

Network I/O

1 Introduction

test[5]

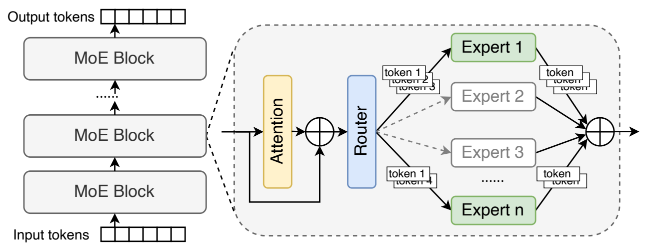

Figure 1: Mixture-of-Experts model architecture.

2 Background and Motivation

2.1 All-to-All Collective in MoE

Large language models (LLMs) have achieved superior performance, however, the cost of these models grows steeply with model size [12, 15]. To decouple model capacity from per-token computation, recent architectures increasingly adopt sparsely-activated Mixture-of-Experts (MoE) layers in place of dense feed-forward sublayers in Transformer blocks [2, 6, 7, 10]. In an MoE layer, the standard dense feed-forward sublayer is replaced by a set of N expert networks, typically independent MLPs, together with a lightweight routing network. The router processes each token produced by the attention sublayer and activates only a small subset of experts to execute, as shown in Figure 1. In practical, a single MoE layer often contains hundreds of experts, which cannot be hosted on a single GPU and are therefore distributed across multiple devices using expert parallelism [1, 4, 8, 9, 11]. Under expert parallelism, the model partitions the expert set across a group of GPUs. A given token may thus be routed to experts residing on different GPUs within the expert-parallel group.

A forward pass through an MoE layer after attention normalization follows a three-stage pattern: token dispatch, expert computation, and result combination. First, each GPU computes for its every local token a top-k set of expert. Then bucket the tokens by expert and dispatch them to the GPUs that host the selected experts. Second, each GPU runs its resident experts, applying the expert MLP to the assigned tokens to produce updated hidden states (expert outputs) for those tokens. Finally, expert outputs are sent back to the token-owning GPUs and combined according to the routing weights to form the MoE layer's output for each token. Viewed from the communication perspective, tokens transmission between each stage naturally induces two all-to-all exchanges. Viewed from the communication perspective, the dispatch and combination stages naturally induce two all-to-all exchanges per MoE layer—one for sending routed activations and one for gathering outputs back. As this communication pattern is exercised at every MoE layer, the latency of the all-to-all collective becomes a first-order determinant of end-to-end MoE performance. Prior work reports that all-to-all communication can account for around 30%-60% of the total training or inference iteration time [9, 16, 17].

2.2 Collective Tax

Under an idealized model with lossless links, full pipelining, and no additional overheads, the latency of an all-to-all operation can be approximated as the ratio between the total communicated volume and the aggregate link bandwidth. Figure 2 compares the latency of MoE all-to-all collectives against this ideal baseline under different system settings and across several representative communication libraries, including NCCL [13], DeepEP [3], and NVSHMEM [14]. We conduct experiments on a 4-node cluster. Each server is equipped with eight NVIDIA H800 GPUs. The detailed configuration is the same in Section 6. We evaluate the single all-to-all collective latency in different expert parallelism degrees. In all cases, the observed latency exceeds the theoretical baseline, with xx%-xx% latency increase. This gap persists despite ample network capacity, indicating that factors beyond raw data transfer play a major role in end-to-end collective latency.

To understand where this additional latency comes from, we inspect the communication subdivisions of the collective communication libraries which is summarized in Figure 3. Across all evaluated libraries, all-to-all collective begins after the routing decisions are finalized. Once the gate computation completes, the expert assignment for each token becomes fixed, and an explicit synchronization point is required to ensure that all GPUs observe a consistent routing result before communication proceeds. This synchronization is common to all implementations and is necessary to preserve correctness, but it already introduces latency that is not captured by the bandwidth-bound model.

Beyond this initial synchronization, existing libraries differ significantly in how they implement the communication. In NCCL and DeepEP, all-to-all collective involves an explicit data shuffling phase both on the sender and receiver side. After gate computation, GPU SMs launch kernels that copy token data from the original tensor buffers into intermediate I/O buffers. During this copy, tokens are reorganized so that data destined for each receiver GPU occupies a contiguous memory region. This cost is independent of network transfer and is therefore invisible to a bandwidth-only performance model. A symmetric process occurs on the receiver side. Once data arrives at the destination I/O buffers, the receiving GPU launches kernels to copy data into the destination tensor buffers. This second copy is again accompanied by a layout transformation, this time to match the memory layout expected by the downstream computation kernels. As a result, a single all-to-all collective under NCCL or DeepEP typically involves memory copies and data shuffles on both the sender

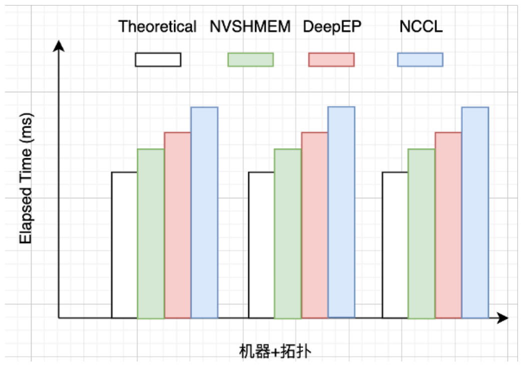

Figure 2: All-to-all collective overhead of existing collective communication libraries.

and receiver, even though the logical communication volume remains unchanged. After our profiling, we find that the data shuffling overhead accounts for around xx%-xx% of the total collective latency.

Alternatively, NVSHMEM follows a different execution path. After gate computation and synchronization, NVSHMEM enables direct remote memory access from the sender's tensor buffers to the receiver's tensor buffers, avoiding explicit data shuffling. This design eliminates intermediate I/O buffers and associated data shuffling, but relies on synchronization to coordinate access to shared tensor regions. As such, while NVSHMEM reduces shuffle-related overheads, it does not remove synchronization costs inherent to all-to-all collective. After our profiling, we find that the synchronization overhead accounts for around xx%-xx% of the total collective latency.

Taken together, these observations suggest that the performance gap between measured all-to-all latency and the theoretical baseline arises primarily from two sources: data shuffling due to memory layout transformations and explicit synchronization required for correct execution. Regardless of the specific implementation, however, at least one of these overheads is incurred in current all-to-all collectives. We collectively refer to the overhead beyond the theoretical baseline as the "collective tax". This tax is not specific to a particular communication library, but instead reflects a fundamental overhead of orchestrating all-to-all collective for MoE layers. Different collective paradigms pay this tax in different ways, trading off memory shuffling overhead, synchronization cost, and implementation complexity. In the next subsection, we examine how existing collective designs handle this orchestration and analyze their inherent limitations through high-level analysis.

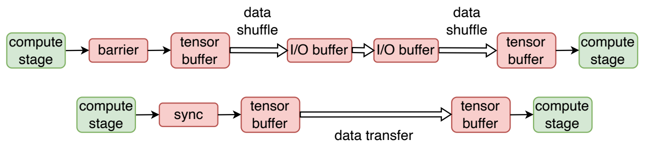

Figure 3: Profiled communication subdivision of different CCLs.

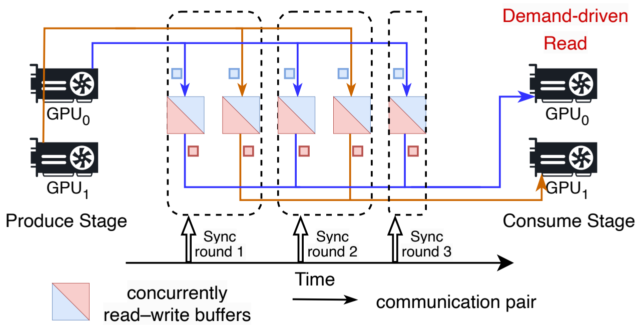

Figure 4: On-demand collective paradigm.

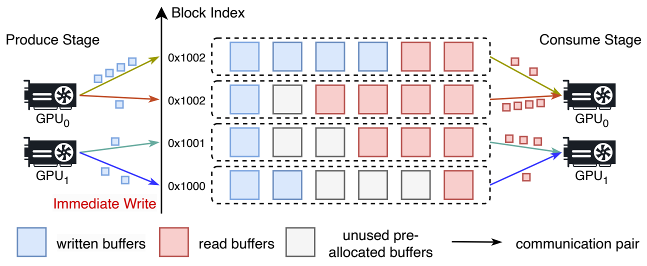

Figure 5: Preload collective paradigm.

3 Design Space Analysis

4 Detailed Design of `FreeIO`

5 Implementation

6 Evaluation

7 Related Work

8 Conclusion

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[3] DeepSeek-AI. 2026. DeepEP github repository. https://github.com/deepseek-ai/DeepEP. Accessed 2026-01-01.

[4] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*. PMLR, 5547–5569.

[5] Kevin R Fall and W Richard Stevens. 2012. *Tcp/ip illustrated*. Vol. 1. Addison-Wesley Professional.

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

[7] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668* (2020).

[8] Wenxue Li, Xiangzhou Liu, Yuxuan Li, Yilun Jin, Han Tian, Zhizhen Zhong, Guyue Liu, Ying Zhang, and Kai Chen. 2024. Understanding communication characteristics of distributed training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*. 1–8.

[9] Xudong Liao, Yijun Sun, Han Tian, Xinchen Wan, Yilun Jin, Zilong Wang, Zhenghang Ren, Xinyang Huang, Wenxue Li, Kin Fai Tse, et al. 2025. Mixnet: A runtime reconfigurable optical-electrical fabric for distributed mixture-of-experts training. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 554–574.

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

[11] Juncai Liu, Jessie Hui Wang, and Yimin Jiang. 2023. Janus: A unified distributed training framework for sparse mixture-of-experts models. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 486–498.

[12] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–15.

[13] NVIDIA. 2026. NCCL github repository. https://github.com/NVIDIA/nccl. Accessed 2026-01-01.

[14] NVIDIA. 2026. NVSHMEM github repository. https://github.com/NVIDIA/nvshmem. Accessed 2026-01-01.

[15] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[16] Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, et al. 2025. Megascale-infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 592–608.

[17] Zhuoran Zhu, Chunyang Zhu, Hao Lin, Xu Fu, Yiming Zhou, Quanlu Zhang, Zhenhua Li, Feng Qian, Chao Yu, Boxun Li, et al. 2025. FUSCO: High-Performance Distributed Data Shuffling via Transformation-Communication Fusion. *arXiv preprint arXiv:2512.22036* (2025).