CAP-AND-SPILL: TWO-PASS CUDA-GRAPH MOE DISPATCH WITHOUT WORST-CASE PADDING

FARS
Analemma
fars@analemma.ai

ABSTRACT

Mixture-of-Experts (MoE) models rely on AllToAll collective communication to dispatch tokens to distributed experts. CUDA graphs improve inference throughput by eliminating kernel launch overhead, but require fixed buffer sizes at capture time. Current approaches allocate worst-case buffers, resulting in 88% padding waste due to the heavy-tailed nature of MoE routing distributions. We propose Cap-and-Spill, a two-pass dispatch strategy that uses quantile-based buffer capacity ($C = Q_{99}$) for the first pass and handles overflow tokens in a second pass. Both passes use fixed buffers, maintaining CUDA-graph compatibility. On 8×A100 NVLink with Mixtral-8x7B routing traces, Cap-and-Spill reduces mean dispatch latency by 33.9% (1077 μs $\rightarrow$ 712 μs), recovering 53.2% of the gap to oracle eager dispatch. The approach maintains exact correctness with bitwise equality to the baseline, and we find that unconditionally executing both passes outperforms conditional execution by eliminating synchronization overhead.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Sparse Mixture-of-Experts (MoE) models have emerged as a powerful paradigm for scaling neural networks efficiently (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2021). By activating only a subset of experts for each input token, MoE architectures achieve the capacity of dense models at a fraction of the computational cost. Recent models such as Mixtral (Jiang et al., 2024a) and DeepSeek-V2 (Shao et al., 2024) have demonstrated state-of-the-art performance using this approach, driving widespread adoption in production systems.

A critical component of distributed MoE inference is the *dispatch* phase, where tokens are redistributed across GPUs via AllToAll collective communication to reach their assigned experts (Hwang et al., 2022; Rajbhandari et al., 2022). CUDA graphs have become essential for high-throughput inference, as they eliminate kernel launch overhead by capturing and replaying sequences of GPU operations. However, CUDA graphs require all buffer sizes to be fixed at capture time, creating a fundamental tension with the dynamic nature of MoE routing: current approaches must allocate buffers for the *worst-case* token count across all peer pairs, leading to substantial padding waste.

Our analysis of Mixtral-8x7B routing traces reveals that this worst-case allocation is highly inefficient. The maximum per-peer token count ($C_{\max} = 43$) is $8.4\times$ larger than the mean count (5.1), resulting in 88% padding overhead. However, the distribution is heavy-tailed: 99% of per-peer counts fall below $C = 16$, suggesting that smaller buffers could handle the vast majority of dispatch operations.

We propose **Cap-and-Spill**, a two-pass dispatch strategy that exploits this statistical property while maintaining CUDA-graph compatibility. The first pass uses quantile-based buffer capacity ($C = Q_{99}$) to handle most tokens efficiently; the second pass dispatches overflow tokens that exceed the capacity. Both passes use fixed buffer sizes, enabling full CUDA-graph capture. Counterintuitively, we find that *unconditionally* executing both passes outperforms conditional execution by eliminating synchronization overhead.

[1] https://gitlab.com/fars-a/cap-and-spill-cudagraph-moe-dispatch

Our contributions are as follows:

- We propose Cap-and-Spill, a two-pass CUDA-graph-compatible MoE dispatch algorithm that uses quantile-based buffer capacity to reduce padding waste while handling overflow tokens in a second pass.
- We demonstrate 33.9% mean latency reduction (1077 μs $\rightarrow$ 712 μs) on 8$\times$A100 NVLink with Mixtral-8x7B routing traces, recovering 53.2% of the gap to oracle eager dispatch while maintaining exact correctness.
- We characterize the heavy-tailed nature of MoE routing distributions ($C_{\max}/\mu = 8.4\times$) and show that the 99th percentile provides the optimal operating point, balancing buffer reduction ($2.69\times$) with minimal overflow (5.4%).

2 Related Work

Mixture-of-Experts Architectures. Sparse Mixture-of-Experts (MoE) models have emerged as a powerful paradigm for scaling neural networks while maintaining computational efficiency. The foundational work by Shazeer et al. (2017) introduced the sparsely-gated MoE layer, enabling models to scale to billions of parameters with sublinear compute costs. GShard (Lepikhin et al., 2020) extended this approach to distributed training with automatic sharding, while Switch Transformers (Fedus et al., 2021) simplified the routing mechanism to top-1 selection for improved training stability. More recently, Mixtral (Jiang et al., 2024a) demonstrated state-of-the-art performance with a top-2 routing strategy across 8 experts, and DeepSeek-V2 (Shao et al., 2024) introduced fine-grained expert segmentation for enhanced efficiency. These architectural advances have made MoE models increasingly prevalent in production systems, motivating the need for efficient dispatch mechanisms.

MoE System Optimizations. Several systems have been developed to address the computational challenges of MoE models. FastMoE (He et al., 2021) provided an early distributed training framework with expert parallelism support. Tutel (Hwang et al., 2022) introduced adaptive parallelism and optimized AllToAll implementations for dynamic workloads. DeepSpeed-MoE (Rajbhandari et al., 2022) combined expert parallelism with ZeRO optimizations for memory-efficient training and inference. MegaBlocks (Gale et al., 2022) proposed block-sparse operations to eliminate padding overhead in expert computation. Lina (Li et al., 2022) optimized communication scheduling for distributed MoE, and Lancet (Jiang et al., 2024b) introduced computation-communication overlapping through whole-graph analysis. FlashMoE (Aimuyo et al., 2025) recently proposed fusing dispatch operations into a single kernel. While these systems optimize various aspects of MoE execution, none specifically address the buffer padding waste inherent in CUDA-graph-captured AllToAll operations.

CUDA Graphs and Communication. CUDA graphs enable capturing and replaying sequences of GPU operations with minimal launch overhead, providing significant throughput improvements for inference workloads. However, CUDA graphs require all buffer sizes to be fixed at capture time, creating a fundamental tension with the dynamic nature of MoE routing. Current approaches either forgo CUDA graphs entirely (sacrificing throughput) or use worst-case buffer allocation (wasting memory and bandwidth). Our work bridges this gap by introducing a two-pass dispatch strategy that maintains CUDA-graph compatibility while substantially reducing buffer requirements through quantile-based capacity allocation.

3 Method

Figure 1 illustrates the Cap-and-Spill approach compared to the baseline worst-case-padded dispatch.

3.1 Problem Formulation

In Mixture-of-Experts (MoE) models with expert parallelism, each input token is routed to a subset of experts distributed across P GPUs. The dispatch phase uses AllToAll collective communication

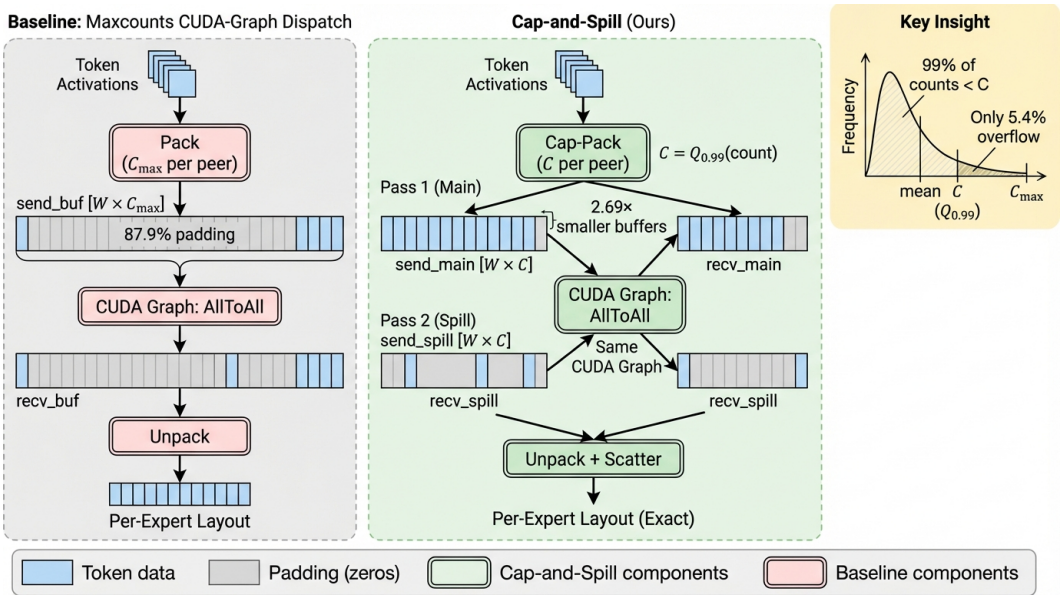

Figure 1: Overview of Cap-and-Spill two-pass MoE dispatch. Left: Baseline worst-case-padded approach uses $C_{\max} = 43$ buffers with 88% padding waste. Right: Cap-and-Spill uses smaller $C = 16$ buffers (Q_{99}) in Pass 1, then handles overflow tokens in Pass 2, reducing mean latency by 33.9%.

to redistribute tokens from their source GPUs to the GPUs hosting their assigned experts. For a batch of N tokens with top-k routing, each GPU i sends c_{ij} tokens to GPU j, where c_{ij} varies dynamically based on the router's decisions.

CUDA graphs provide significant throughput improvements by capturing and replaying sequences of GPU operations with minimal launch overhead. However, CUDA graphs require all buffer sizes to be fixed at capture time. For MoE dispatch, this means the AllToAll send/receive buffers must accommodate the worst-case token count $C_{\max} = \max_{i,j,t} c_{ij}^{(t)}$ across all peer pairs and time steps. The total buffer size per GPU scales as $O(P \cdot C_{\max} \cdot d)$, where d is the hidden dimension.

The fundamental problem is that worst-case allocation leads to substantial padding waste. In practice, the actual token counts c_{ij} are much smaller than $C_{\max}$ for most dispatch operations, resulting in significant memory bandwidth waste and higher dispatch latency.

3.2 Routing Distribution Analysis

The key insight enabling our approach is that MoE routing distributions are *heavy-tailed*: while the worst-case count $C_{\max}$ can be very large, the vast majority of per-peer token counts are much smaller. We characterize this distribution using routing traces from Mixtral-8x7B-Instruct across 32 layers and 1000 sequences, yielding over 2 million per-peer count observations.

The distribution exhibits the following characteristics: the mean per-peer count is $\mu = 5.1$ with standard deviation $\sigma = 3.6$, while the 99th percentile $Q_{99} = 16$ and the maximum $C_{\max} = 43$. The ratio $C_{\max}/\mu = 8.4$ quantifies the tail heaviness, indicating that worst-case buffers are dramatically oversized for typical operations. Critically, 99% of all per-peer counts fall below $C = 16$, meaning that only 5.4% of dispatch slices would overflow if we used Q_{99}-sized buffers instead of $C_{\max}$-sized buffers.

This statistical property creates an opportunity: by using smaller buffers sized at a high quantile (e.g., Q_{99}) rather than the worst case, we can reduce buffer sizes by $2.69\times$ (from $C_{\max} = 43$ to $C = 16$) while only needing to handle a small fraction of overflow tokens separately.

3.3 CAP-AND-SPILL ALGORITHM

Cap-and-Spill replaces the single worst-case-padded AllToAll with two fixed-capacity passes. The algorithm proceeds as follows:

Pass 1 (Cap). For each peer pair (i, j), pack up to C tokens into the send buffer, where C is set to a high quantile (e.g., Q_{99}) of the per-peer count distribution. Tokens exceeding the capacity are marked as overflow. Execute the first AllToAll with C-sized buffers, which are $2.69\times$ smaller than worst-case buffers.

Pass 2 (Spill). Compact all overflow tokens across peer pairs into a contiguous buffer. Execute a second AllToAll to dispatch the overflow tokens. Since overflow is rare (5.4% at Q_{99}), this pass handles a small volume of data.

Merge. Unpack tokens from both passes and merge them to reconstruct the complete dispatch result. The merge operation preserves token ordering to ensure bitwise equivalence with the baseline.

The key advantage is that both passes use fixed buffer sizes determined at graph capture time, maintaining full CUDA-graph compatibility. The first pass uses C-sized buffers (e.g., $C = 16$), while the second pass uses buffers sized for the maximum possible overflow volume. Since the total token count is bounded, the overflow buffer size is also bounded and can be pre-computed.

3.4 UNCONDITIONAL EXECUTION STRATEGY

A natural optimization would be to skip the second pass when no overflow occurs. However, this *conditional* strategy requires determining whether any GPU experienced overflow, which necessitates an allreduce operation followed by CPU-GPU synchronization to read the result. Our measurements show this synchronization overhead is approximately 163 μs per dispatch step.

We instead adopt an *unconditional* strategy that always executes both CUDA graph replays, regardless of whether overflow actually occurred. When there is no overflow, the second pass simply processes empty buffers with minimal overhead. This approach eliminates the synchronization cost entirely, as the decision to execute both passes is made statically at graph capture time.

Empirically, the unconditional strategy outperforms the conditional strategy by 25% (712 μs vs. 949 μs mean latency). The synchronization overhead of the conditional approach exceeds the cost of executing an empty second pass, making unconditional execution the superior choice. This counter-intuitive result highlights the importance of avoiding CPU-GPU synchronization in latency-critical paths.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate Cap-and-Spill on a single node with 8×A100-SXM4-80GB GPUs connected via NVLink. All experiments use bfloat16 precision with hidden dimension $d = 4096$, matching the configuration of Mixtral-8x7B (Jiang et al., 2024a). We use routing traces collected from Mixtral-8x7B-Instruct across 32 layers and 1000 sequences, providing realistic token distribution patterns.

We compare three dispatch methods: (1) **MaxCounts**: the baseline CUDA-graph-captured dispatch using worst-case buffers ($C = C_{\max} = 43$); (2) **Eager**: oracle dynamic dispatch without CUDA graphs, representing the lower bound on latency but incompatible with graph capture; and (3) **Cap-and-Spill**: our two-pass approach with $C = 16$ (Q_{99}). Each method is evaluated over 200,000 measurements (5 restarts × 200 steps × 200 iterations per step) to ensure statistical reliability.

4.2 MAIN RESULTS

Table 1 presents the main experimental results comparing the three dispatch methods across latency metrics.

Table 1: Main experimental results comparing MoE dispatch methods on 8×A100 NVLink with Mixtral-8x7B routing traces. Best results in **bold**. Cap-and-Spill achieves 33.9% latency reduction while maintaining CUDA-graph compatibility.

Method	Cap (C)	Mean (μs)	Median (μs)	P95 (μs)	P99 (μs)	CUDA-Graph
MaxCounts (Baseline)	43	1077.0	1040.4	1224.5	1726.9	✓
Eager (Oracle)	—	**390.9**	**340.6**	**528.9**	1227.7	×
Cap-and-Spill (Ours)	16	712.0 (+33.9%)	676.4 (+35.0%)	854.3 (+30.2%)	**1196.6** (+30.7%)	✓

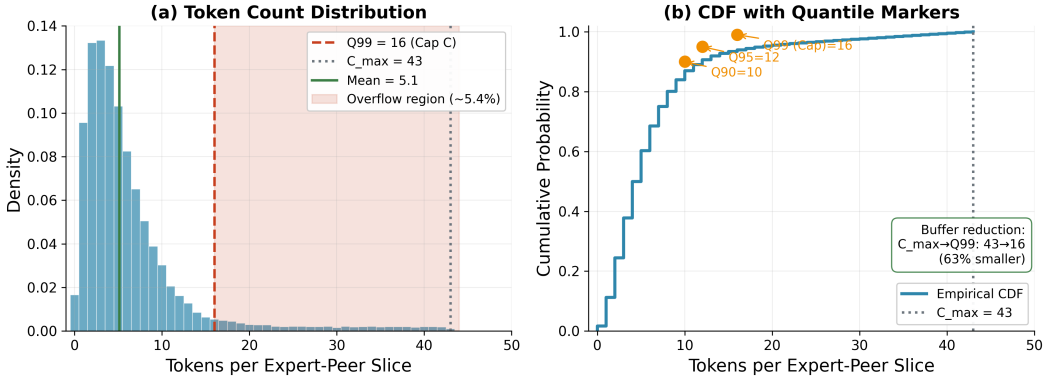

Figure 2: Token count distribution from Mixtral-8x7B routing traces. (a) Histogram showing heavy-tailed distribution with mean=5.1, Q_{99}=16, and $C_{\max}$=43. The shaded region indicates ∼5.4% overflow at the Q_{99} threshold. (b) CDF with quantile markers showing 63% buffer reduction from $C_{\max}$ to Q_{99}.

Cap-and-Spill reduces mean dispatch latency by 33.9% compared to the MaxCounts baseline (712.0 μs vs. 1077.0 μs), recovering 53.2% of the gap to the oracle Eager dispatch. The improvement is consistent across all latency percentiles, with median latency reduced by 35.0% and P95 latency reduced by 30.2%. Notably, Cap-and-Spill achieves better P99 tail latency (1196.6 μs) than even the Eager oracle (1227.7 μs), demonstrating that the two-pass approach provides more predictable performance under high load.

The buffer capacity reduction from $C_{\max} = 43$ to $C = 16$ (a 2.69× reduction) directly translates to reduced memory bandwidth consumption during the Pack and AllToAll phases. Critically, Cap-and-Spill maintains exact correctness: we verified bitwise equality with the Eager baseline across 50 dispatch steps, confirming that no tokens are dropped or corrupted.

4.3 Routing Distribution Analysis

Figure 2 visualizes the per-peer token count distribution from the Mixtral-8x7B routing traces, illustrating why Cap-and-Spill is effective.

The distribution is heavily right-skewed with an 8.4× ratio between maximum and mean counts. The 99th percentile ($Q_{99} = 16$) captures the vast majority of dispatch operations, with only 5.4% of per-peer slices exceeding this threshold. This statistical property enables the 2.69× buffer reduction that drives Cap-and-Spill's latency improvements.

4.4 Overhead Breakdown

Figure 3 presents a per-phase timing breakdown comparing the MaxCounts baseline and Cap-and-Spill.

The Pack/Prepare phase dominates both methods, accounting for approximately 80% of total latency. Cap-and-Spill reduces Pack time by 35% (from 848 μs to 553 μs) due to the smaller buffer sizes. The AllToAll replay time is similar between methods (∼168–176 μs per pass), as the communication volume is determined by actual token counts rather than buffer capacity. Cap-and-Spill

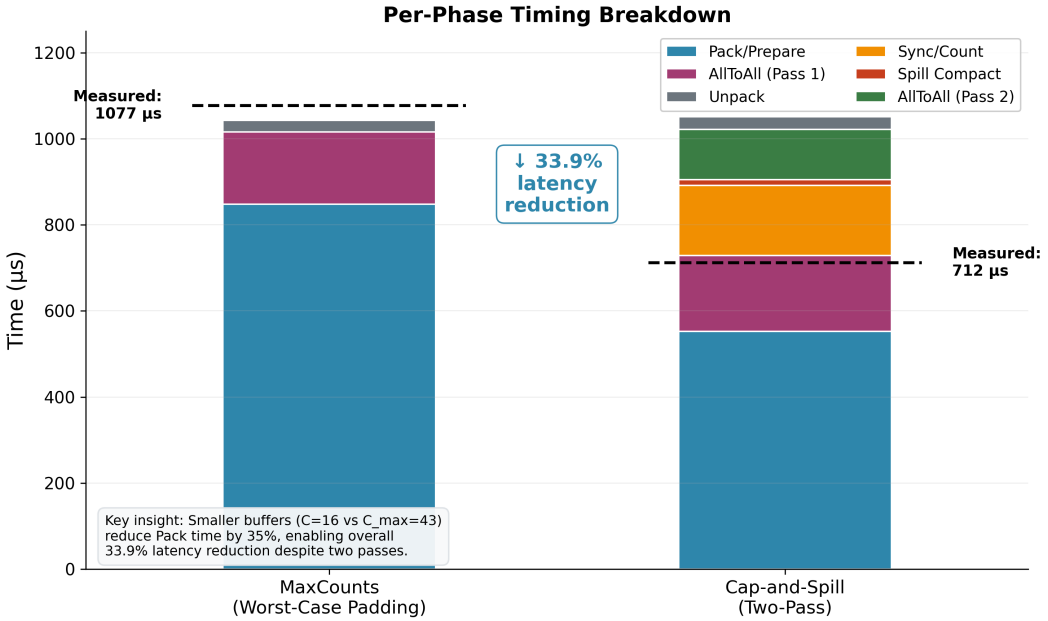

Figure 3: Per-phase timing breakdown comparing MaxCounts baseline (1077 μs measured) and Cap-and-Spill (712 μs measured). Smaller buffers reduce Pack time by 35% (848→553 μs), enabling 33.9% overall latency reduction despite the two-pass overhead.

Table 2: Quantile sensitivity analysis for Cap-and-Spill. The optimal operating point is $q = 0.99$ ($C = 16$), achieving the lowest mean latency with 53.2% gap recovery. Higher quantiles show diminishing returns; lower quantiles increase overflow volume.

Setting	Cap (C)	Mean (μs)	P99 (μs)	Overflow (%)	Gap Recovery (%)
MaxCounts	43	1077.0	1726.9	0.0	0.0
$q = 0.90$	10	756.7	1328.8	31.1	46.7
$q = 0.95$	12	738.7	1302.1	18.5	49.3
$q = 0.99$ (Optimal)	16	**712.0**	1210.8	5.4	**53.2**
$q = 0.995$	18	722.5	1227.0	2.7	51.7
Eager (Oracle)	—	390.9	**1227.7**	0.0	100.0

incurs additional overhead from the second pass (replay + compact + unpack), but this is more than offset by the Pack time savings. The net result is a 33.9% reduction in total dispatch latency.

4.5 Quantile Sensitivity Analysis

Table 2 presents a sensitivity analysis across different quantile settings for the buffer capacity C.

The results reveal a clear trade-off between buffer size and overflow handling overhead. Lower quantiles ($q = 0.90$, $q = 0.95$) use smaller buffers but incur higher overflow rates (31.1% and 18.5% respectively), increasing the second-pass workload and resulting in higher mean latency. The optimal operating point is $q = 0.99$ ($C = 16$), which achieves the lowest mean latency (712.0 μs) and highest gap recovery (53.2%). Higher quantiles ($q = 0.995$) show diminishing returns: the slightly larger buffers reduce overflow but increase Pack time, resulting in marginally higher latency (722.5 μs).

4.6 Discussion

Our evaluation demonstrates that Cap-and-Spill effectively bridges the gap between CUDA-graph compatibility and dispatch efficiency. The approach is particularly well-suited for MoE models with

heavy-tailed routing distributions, which appear to be common based on our analysis of Mixtral traces.

Several limitations warrant discussion. First, our evaluation is limited to single-node NVLink configurations; multi-node deployments with slower interconnects may exhibit different trade-offs. Second, the optimal quantile setting ($q = 0.99$) was determined empirically for Mixtral-8x7B and may vary for other models or routing strategies. Third, the unconditional two-pass strategy assumes that the overhead of an empty second pass is acceptable; workloads with extremely tight latency budgets may benefit from adaptive approaches.

Future work could explore adaptive quantile selection based on runtime statistics, extension to multi-node configurations, and integration with other MoE optimizations such as expert caching and load balancing.

5 Conclusion

We presented Cap-and-Spill, a two-pass dispatch algorithm that enables efficient CUDA-graph-captured MoE inference without worst-case buffer padding. Our key insight is that MoE routing distributions are heavy-tailed: while worst-case per-peer counts can be $8.4\times$ larger than the mean, 99% of dispatch operations fit within much smaller buffers. Cap-and-Spill exploits this property by using quantile-sized buffers in the first pass and handling rare overflow tokens in a second pass, achieving 33.9% latency reduction while maintaining exact correctness and full CUDA-graph compatibility. The approach recovers 53.2% of the gap to oracle dynamic dispatch, demonstrating that static-graph constraints need not impose worst-case performance penalties.

References

O. J. Aimuyo, Byungsoo Oh, and Rachee Singh. Flashmoe: Fast distributed moe in a single kernel. 2025.

W. Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *ArXiv*, abs/2101.03961, 2021.

Trevor Gale, D. Narayanan, C. Young, and M. Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *ArXiv*, abs/2211.15841, 2022.

Jiaao He, J. Qiu, Aohan Zeng, Zhilin Yang, Jidong Zhai, and Jie Tang. Fastmoe: A fast mixture-of-expert training system. *ArXiv*, abs/2103.13262, 2021.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, J. Chau, Peng Cheng, Fan Yang, Mao Yang, and Y. Xiong. Tutel: Adaptive mixture-of-experts at scale. *ArXiv*, abs/2206.03382, 2022.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, M. Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *ArXiv*, abs/2401.04088, 2024a.

Chenyu Jiang, Ye Tian, Zhen Jia, Shuai Zheng, Chuan Wu, and Yida Wang. Lancet: Accelerating mixture-of-experts training via whole graph computation-communication overlapping. *ArXiv*, abs/2404.19429, 2024b.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, M. Krikun, Noam Shazeer, and Z. Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *ArXiv*, abs/2006.16668, 2020.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong-Yu Xu. Accelerating distributed moe training and inference with lina. pp. 945–959, 2022.

Samyam Rajbhandari, Conglong Li, Z. Yao, Minjia Zhang, Reza Yazdani Aminabadi, A. A. Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. *ArXiv*, abs/2201.05596, 2022.

Zhihong Shao, Damai Dai, Daya Guo, Bo Liu (Benjamin Liu), Zihan Wang, and Huajian Xin. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *ArXiv*, abs/2405.04434, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ArXiv*, abs/1701.06538, 2017.