TurboMoE: Enhancing MoE Model Training with Smart Kernel-Fusion and Data Transformation

Reza Yazdani Aminabadi*, Connor Holmes†, Samyam Rajbhandari*, Zhewei Yao*, Yuxiong He*

* Snowflake AI Research †OpenAI

Abstract

The Mixture of Experts (MoE) model is a powerful architecture that dynamically selects a subset of experts for each input, enabling the model to scale efficiently. However, the gating mechanism, which determines the assignment of tokens to experts, introduces 4-dimensional ($S \times E \times C \times M^1$) computational complexity due to its reliance on sparse representation which results in wasteful dense-computation. In this work, we present TurboMoE, a novel approach to accelerate MoE model training by optimizing the gating logic through smart kernel-fusion and data-layout transformations.

Our method addresses the computational bottlenecks of the gating process by introducing three specialized kernels. The first kernel efficiently computes expert scores and performs top-k expert selection, while the second kernel scatters input tokens into expert-specific buffers, minimizing the need for sparse operations. Furthermore, we introduce the third MoE-Gather kernel, which replaces the traditional sparse matrix multiplication, streamlining the process of combining expert outputs.

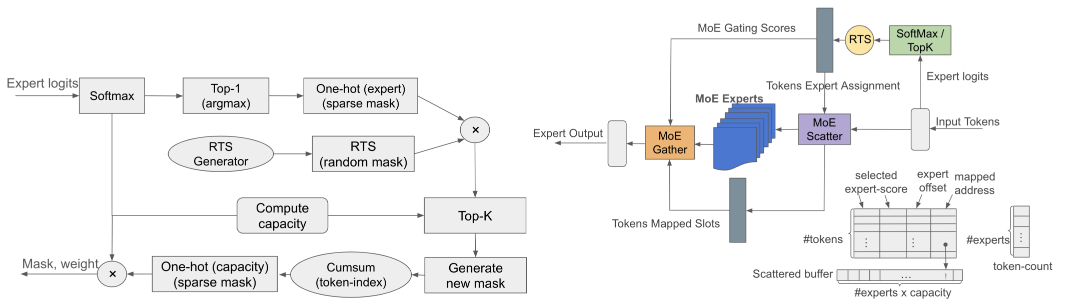
By integrating these kernels, TurboMoE achieves substantial end-to-end speedups over the state-of-the-art solution, MegaBlocks, with a 55% faster training time for top-1 selection and a 41% improvement for top-2 selection configurations. These optimizations significantly reduce the computation overhead of the gating functionality from O($SECM$) $\to O(SM)$. TurboMoE demonstrates that by removing the reliance on sparse computation, MoE models can achieve unprecedented training efficiency, reaching 460 Tera-Flops on 32 NVIDIA-H100 for a 32-expert MoE architecture with Top-2 gating configuration, paving the way for more scalable and effective applications.

1 Introduction

The Mixture of Experts (MoE) architecture has become a cornerstone in modern machine learning for its ability to scale models efficiently [3, 4, 5, 6]. By dynamically selecting a subset of experts for processing each input, MoE models offer a unique advantage: they can scale to immense sizes without a proportional increase in computational costs. This selective activation of experts ensures that computational resources are focused where they are most needed, making MoE models particularly appealing for large-scale applications. However, this architectural efficiency comes with its own set of challenges, primarily centered around the gating mechanism and the all-to-all communication. Here, we focus on the gating function and how the tokens are routed for different experts.

The gating mechanism in MoE models is responsible for deciding which experts should handle each token, a decision that is inherently sparse—only few experts are selected for each token. Figure 1(a) shows the computatuion flow for the top-1 MoE gating. Traditional implementations of the gating logic involve several steps that introduce substantial computational overhead. The process begins with generating a sparse mask through top-k selection for each token. This step is computationally

[1] S: #tokens, E: #experts, C: capacity, M: model_dimension



(a) (b)

Figure 1: (a) MoE-Gating computation overview. (b) TurboMoE data-transformation and kernel optimization.

expensive, as it not only selects the top-k experts but also involves a Random-Token-Selection (RTS) mechanism to balance the load across experts.

Once the experts are selected, the indexing operation uses the generated mask to sequence tokens through a cumulative sum (cumusum) operation, which determines the order in which tokens will be processed by the experts. The cumsum operation is inherently a sequential one, and cannot be efficiently parallelized. This is followed by the creating an output mask, a 3-dimensional one-hot matrix that indicates the specific experts and capacities assigned to each token. The mask guides the subsequent steps of scattering tokens to the appropriate experts and gathering the outputs, both of which involve complex data manipulations and sparse matrix-multiplications (MM).

In conventional frameworks like PyTorch, these operations are typically executed using high-level operations such as dense $einsum$, which internally rely on reshaping, transposing, and dense MMs. These operations, while flexible, come with significant drawbacks: they incur frequent data reordering and copying which makes them inefficient. Furthermore, the use of dense MM to do sparse MM significantly increases the overhead. The result is a gating mechanism that, while functional, becomes a bottleneck in the overall training process, particularly as the number of experts (E) and the model's scale increase.

Recognizing these inefficiencies, we introduce TurboMoE, a novel approach designed to optimize the MoE gating mechanism by eliminating its reliance on sparse computation. The core innovation of TurboMoE lies in the fusion of multiple operations into custom kernels, coupled with smart data-layout transformations that remove the need for costly data reordering and dense operations on sparse data.

The key components of our approach include:

- Logit Softmax + Top-1/2/4 Selection: We generate an assignment table for input tokens, determining expert selection and saving expert scores for subsequent use.
- Random-Token-Selection + TopK: This kernel finalizes expert assignments, ensuring load balancing through random token selection and capacity-based filtering.
- Scatter Tokens Contiguously: Tokens are moved into expert-specific slots, with mapping information stored for efficient retrieval and processing.
- Gather Expert Outputs: We restore the original token order and apply MoE gating scores, replacing wasteful dense multiplication on sparse data with more efficient operations.

By eliminating the overhead associated with data reorganization and sparse computation, TurboMoE achieves a remarkable 3x speedup in training time for gating operations compared to standard implementations, particularly when scaling to 64-expert parallelism. This work not only advances the efficiency of MoE models but also provides a scalable solution for future developments in machine learning architectures.

2 MoE-Gating Optimizations For Irregular and Sparse Operators

In order to remove the sparse complexity of the MoE-gating, TurboMoE employs several buffers storing three main information: the mapped experts (expert-ID and its probability score) for each

Table 1: Computation Complexity of the MoE-Gating operations.

	PyTorch Implementation	**TurboMoE Implementation**
Top-K selection	$S \times E$	S
Scatter to experts	$S \times E \times C$	S
Gather from experts	$S \times E \times C \times M$	$S \times M$

token, the offset of the token at each selected expert, and the number of tokens selected for each expert. Figure 1(b) illustrates the data-structure used by TurboMoE for storing the gating information. By using this meta-data, we can easily compute each token's mapped address to the scatter buffer. We divide the gating functions into three main kernels as shown by Figure 1(b).

- Logit softmax + Top-1/2/4 selection: SoftMax computes the probability of selecting different experts for each token which is then used to choose the ones with the highest likelihoods. This kernel employs the expert-assignment table to store the expert-ID and the expert-score for each token. It also counts the number of tokens mapped for each expert. The score is later used in the gather kernel when merging the expert's outputs.
- Random-Token-Selection + TopK: in this kernel, we finalize the expert-assignment based on the number of tokens assigned to experts. If more than capacity is assigned at each expert, the RTS will drop some based on the random probabilities to enforce load-balancing. For the experts which have lower assignments than capacity, all the assigned tokens are processed. This kernel finally stores the token-offsets assigned to each expert. As Table 1 shows, we reduce the complexity of this part from O($SE) \rightarrow O(S$).
- Scatter tokens in contiguous manner: this kernel employs the expert-assignment table plus the token-offset table to move the tokens into the slots for each selected expert. It also stores the mapping slot information in a small table with the size of #tokens to prevent the recalculation the mapped address for each token. This table will be used both in the Gather kernel, and also in the backward pass. By using this data-representation, we can reduce the cost of this part from O($SEC) \rightarrow O(S$).
- Gather the experts output: by using the mapped slot information from the scatter kernel, we now put the output of each expert in the right token's address. We also multiply the output by the MoE scores computed in the initial kernel. Having the meta-data created in the previous kernel helps prevent the unnecessary matrix-multiplication with the sparse expert-score matrix (*SCE*) that is used for the PyTorch implementation. TurboMoE reduces this expensive operation's complexity from O($SECM) \rightarrow O(SM$) as it only requires to go through all experts' selected tokens rather than traversing the entire expert's capacity and pick the expert which is selected for each token.

For the backward implementation, we use a modified version of the scatter kernel as the gather's backward kernel. This kernel also computes the expert-score's gradient which sums over the expert output's gradient along the model-dimension. Similarly, we employ the gather kernel to implement the backward of the scatter function. One important aspect of the backward implementation is the training stability. For instance, when implementing the gradient of the different parts of the gating, we use the mathematics chain rule of differentiation, and simplify the formula to reduce the number of operations. One basic reduction technique we used is factoring the scale for the accumulating of the partial results of the gradients computation. However, we noticed that such optimization can result in training instability due to numerical underflow and overflow. So, we realized that the scaling, no matter how costly it is, should happen before we sum up the partial results. Overall, we have improved and verified the stability of the TurbMoE's training performance across several MoE architectures with different number of experts. In fact, we leveraged TurboMoE for both pre-training and post-training of Snowflake Arctic [6].

3 Performance Evaluation

In this section, we evaluate TurboMoE's performance, and present four set of results: i) Latency and throughput comparison, ii) Performance Breakdown, iii) Expert Scalability, and iv) Small Batch Performance.

Configuration: Using a 2.4B LLaMA-based model as the base, we assess TurboMoE with various MoE configurations, comparing it against a basic MoE implementation [1] and MegaBlocks, the current SOTA. All experiments run on a 4-node, 32-GPU cluster of NVIDIA H100s, using

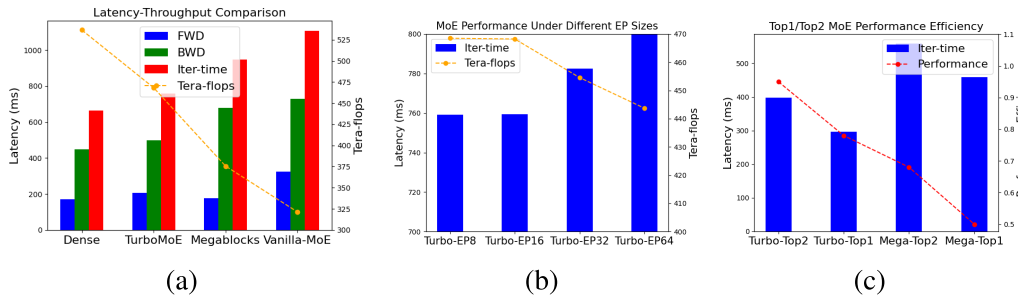


Figure 2: (a) Latency and throughput of the different MoE implementations compared with a strong dense training system. (b) Latency and throughput of TruboMoE using different expert-parallelism degrees: 8, 16, and 32. (c) Comparing latency and performance-efficiency of TurboMoE and Megablocks using different top-k gating at large-scale training.

Megatron-DeepSpeed as the core system. We apply DeepSpeed's ZeRO-2 [7]for efficient gradient communication.

TurboMoE vs Baseline: As the first experiment, we train an 8-expert MoE model (14.7B parameters) with a 4096 context length on 32 H100 GPUs. We compare performance not only with other MoE implementations but also with a dense model to evaluate the overhead of token routing. As shown in Figure 2(a), TurboMoE outperforms basic MoE by 46%, reaching 470 Tera-Flops, while MegaBlocks achieves a 376 Tera-Flops throughput - 25% less than TurboMoE. Compared to dense training, TurboMoE only has 14.5% overhead, with 8% of this from gating computations. Despite the system overhead, MoE models deliver significantly higher accuracy for the same compute (total floating point operations) and can achieve equivalent accuracy using 5x less compute, making them a preferred approach over dense models [5, 2]. Key Results:

- TurboMoE achieves 46% better performance than naive MoE, reaching 470 Tera-Flops, and 25% better system throughput than MegaBlocks.
- At large-scale training system, TurboMoE achieves 1.55x and 1.41x speedup over MegaBlocks for top-1 and top-2 gating, respectively.

Performance Breakdown: In terms of latency breakdown, MegaBlocks's forward pass latency is nearly identical to the dense model, though its backward pass is 51% slower. TurboMoE, on the other hand, has a 20% forward pass overhead and an 11% backward pass overhead. MegaBlocks' handling of sparse tokens during the forward pass increases backward overhead due to sparse gradient computations, while TurboMoE uses a simpler gather-scatter operation by emploting the metadata saved during forward pass, minimizing the backward overhead. This gives TurboMoE a significant performance advantage, yielding 25% better throughput than MegaBlocks.

Expert Scalability: Figure 2(b) demonstrates TurboMoE's scalability, with performance remaining stable when scaling from 8 to 16 experts, and only 3% and 5.5% drop when using 32 and 64 experts, respectively (We use 8-node cluster for the EP-64 experiment to check our system-scalability).

Small Batch Performance: Finally, we evaluate small-batch performance for different top-k gating configurations. This setting is common for the large-scale model training system where vast number of GPUs is equipped to run the training experiments as fast as possible. Under this conditions, the system design needs to be scalable enough to reduce the time even though the workload decreases compared to smaller-scale systems. Figure 2(c) shows that TurboMoE reaches 95% efficiency with top-2 gating and 80% with top-1. TurboMoE outperforms MegaBlocks with a 1.55x speedup for top-1 gating and 1.41x for top-2 gating.

4 Conclusion

In this work, we introduced TurboMoE, an optimized solution that addresses the inefficiencies of traditional MoE gating mechanisms through kernel-fusion and data-layout transformations, eliminating costly sparse operations. Our evaluation shows significant speedups over MegaBlocks, with 1.55x and 1.4x improvements for top-1 and top-2 gating, respectively. TurboMoE scales efficiently across parallelism dimensions, offering high throughput and resource utilization for large-scale systems. We plan to release these optimizations as part of the DeepSpeed library, enhancing MoE training for the broader machine learning community.

References

[1] DeepSpeed. Pytorch moe implementaion of top1 gating. `https://github.com/microsoft/DeepSpeed/blob/master/deepspeed/moe/sharded_moe.py`, 2024.

[2] Google. More efficient in-context learning with glam. `https://ai.googleblog.com/2021/12/more-efficient-in-context-learning-with.html`, 2021.

[3] 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, Marie-Anne 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. https://arxiv.org/abs/2401.04088, 2024.

[4] 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. https://arxiv.org/abs/2006.16668, 2020.

[5] 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. https://arxiv.org/abs/2201.05596, 2022.

[6] Snowflake AI Research. Snowflake arctic: The best llm for enterprise ai — efficiently intelligent, truly open. `https://www.snowflake.com/en/blog/arctic-open-efficient-foundation-language-models-snowflake/`, 2024.

[7] DeepSpeed Team. Zero redundancy optimizer. `https://www.deepspeed.ai/tutorials/zero/`, 2024.