FusedXpert: GPU Kernel for Fine-Grained Mixture of Experts

Arthur Feeney[1] Ying Wai Li[2] Aparna Chandramowlishwaran[1]
[1]University of California, Irvine [2]Los Alamos National Lab
{afeeney,amowli}@uci.edu

1 Introduction

Recent advances in large language models (LLMs) have largely been driven by empirical scaling laws that relate a model's quality to its size, the amount of training data, and number of training steps [2, 3]. This partly motivates why the community has largely converged on the transformer architecture: they are quite general and typically scale well to large parameter counts and datasets.

The Sparsely-gated Mixture of Experts (MoE) has recently become a popular alternative to the dense feed-forward layers used in transformers [10]. The primary motivation is to further increase the model's parameter count, without a proportional increase in the number of floating point operations (FLOPs) [4, 10]. Using an MoE to scale the model size is a general trick and is not specific to language models: they are seeing a surge in use in broad application areas, including vision [7] and reinforcement learning [6].

There is an additional architectural trend towards *fine-grained MoEs*, which have many small experts. This has seen use in models like DeepSeek-V3, Llama 4, Qwen3, and OLMoE [1, 5, 14]. When doing inference or finetuning, routing tokens across a large number of experts can result in many small matrix multiplications of different sizes. One way to handle these small products is to use *grouped GEMMs*, as supported by libraries ike CUTLASS and cuBLAS to fuse a "group" of small matrix multiplications into a single kernel. Grouped kernels have a similar motivation to batched APIs.

In this work, we develop high-performance *single-GPU MoE kernels* using the triton DSL [12]. We implement kernels that fuse parts of the token routing with grouped GEMMs and grouped GLUs. This is an important application area as many researchers have had a noted difficulty with achieving real performance improvements when attempting to use MoEs [5]. This work prioritizes fine-tuning and inference, where inputs tend to be relatively small and all experts in a particular layer may fit on single GPU.

2 Background: MoE

When using a MoE in transformers, the dense feed forward layers are replaced by an MoE layer which consists of a *router* and e *experts*. The number of experts e typically varies from 8 up to 256. The router selects a subset of $k \ll e$ experts to route each token to. The number of FLOPs scales linear with k, but is constant with respect to e. Experts are typically simple MLPs. Many recent models like DeepSeek-V3 and Qwen3 choose to make all experts identical Gated Linear Units (GLU) [1, 8].

The specific details of an MoE can vary substantially. However, the general design has two main parts: 1. for each token, the router produces scores and 2. tokens are processed by the experts with the highest scores. One possible MoE is illustrated in Figure 1.

A simple way to implement an MoE is to 1. loop over all of the experts, 2. get the subset tokens that were routed to the current expert, 3. apply the expert to this subset of tokens, and 4. accumulate the output onto an output cache. This is how popular libraries like huggingface transformers implement MoEs [13]. The main issue with using a python for loop over the experts is that when using fine-grained MoEs, there can be a large number of small kernels that are executed serially. It is possible that the GPU will be underutilized if the number of tokens routed to an expert is too small.

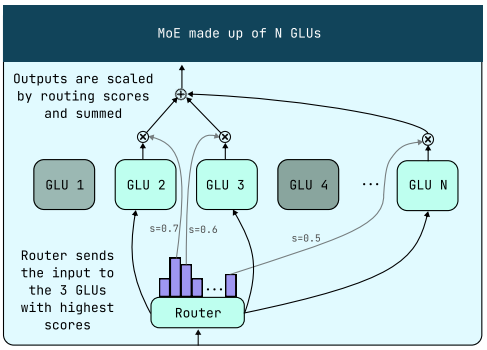


Figure 1: Diagram of an MoE. For each input token, the router produces a score for each expert. The token is then sent to the experts with the highest scores. The experts' outputs are scaled by the routing probabilities, and then accumulated.

3 MoE Routing Analysis

There is a phemonenon called "expert collapse" that may occur when training an MoE, in which the router begins prioritizing only one expert [10]. One common solution to this is to use an additional auxiliary loss to encourage the router to distribute tokens evenly across experts. In spite of this, as illustrated in figure 2, we find that the routing can still be quite biased when processing real datasets. This is important for constructing performant implementations of MoEs because the sizes matrix multiplications in each expert can vary substantially. This also highlights why using separate GEMM kernels for a fine-grained MoE can fail to achieve good performance.

4 Kernel Design

We briefly illustrate how an MoE can be implemented with grouped GEMMs and discuss where there are opportunities for fusion.

ROUTE(X, W_{router}, k)
1 $logits = XW_{router}$
2 $topk\text{-}scores, topk\text{-}indices =$ TOPK(SOFTMAX$(logits), k)$
3 $perm\text{-}tokens, groups =$ PERM-TO-EXPERTS$(tokens, topk\text{-}indices)$
4 **return** $perm\text{-}tokens, groups$

The router takes t tokens $X \in \mathbb{R}^t \times d$, a router weight $W_{router} \in \mathbb{R}^{d \times e}$, and the number of experts to route to, k. The PERM-TO-EXPERTS

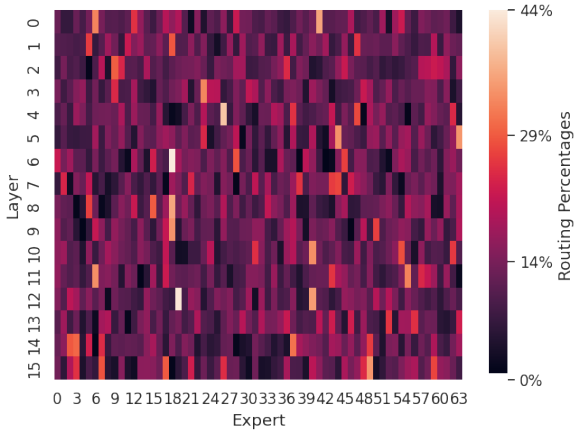


Figure 2: Olmoe router distribution on wikitext. The router in each of the sixteen layers can be heavily skewed. Some experts receive over 40% of the tokens, while others receive less than 1%.

function produces the new array, *perm-tokens*, which contains the routed tokens. Each group corresponds to one expert. I.e., $perm\text{-}tokens[groups[e] : groups[e+1]]$ contains the tokens routed to expert e. The *groups* array defines the extent of each group. So, $groups[e+1] - groups[e]$ is the number of tokens routed to expert e. These are the groups that can be processed by grouped GEMMs. The described MoE uses GLUs [9].

MoE-Grouped-GEMM($perm\text{-}tokens, groups, W_{up}, W_{gate}, W_{down}$)

1 up = Grouped-gemm($perm\text{-}tokens, groups, W_{up}$)
2 $gate$ = Grouped-gemm($perm\text{-}tokens, groups, W_{gate}$)
3 $gated = up \times \text{silu}(gate)$
4 $down$ = grouped-gemm($gated, groups, W_{down}$)
5 out = scale-and-acc($down, groups$)
6 **return** out

There are a number of improvements that can be made to this high-level implementation:

(1) The token permutation can be fused with the grouped GEMMs that produce *up* and *gate* [11].
(2) The two grouped GEMMs producing up and gate can be fused, as the process the same input.
(3) The SiLU activation can then be fused with the GEMMs.

The scale-and-acc function appears to be difficult to fuse. It is left unfused in our current implementation.

5 Results and Discussion

We present performance results for our implementation. The implementations for each model are tested against their corresponding implementation on huggingface transformers [13]. However, as the implementations on hugging transformers have relatively poor performance, we exclude them from our results. We use ScatterMoE as our baseline [11]. Our experiments are run one Nvidia A30 GPU. All models use the bfloat16 data type. Results are shown in Figure 3.

The unfused MoE clearly becomes the slowest, while ScatterMoE and our fused implementation scale similarly. As with our fused implementation, ScatterMoE fuses some of the routing operations into its GEMM kernels. The main culprit for the unfused MoEs poor performance is the output scaling and reduction, shown in Figure 1. This has a poor memory access pattern. Fusing routing with the grouped GEMM is effective at hiding the latencies of some of these memory accesses.

We note that ScatterMoE performs poorly on very small inputs because it does not tune the M-dimension of the GEMM tile size. The number of tokens per expert may be extremely small and if the tile size is too large, it is necessary to pad the tile with zeros, resulting in extra computation.

References

[1] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[2] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[3] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[4] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, et al. 2024. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871* (2024).

[5] Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. 2024. OLMoE: Open Mixture-of-Experts Language Models. arXiv:2409.02060 [cs.CL] https://arxiv.org/abs/2409.02060

[6] Johan Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. 2024. Mixtures of experts unlock parameter scaling for deep rl. *arXiv preprint arXiv:2402.08609* (2024).

[7] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems* 34 (2021), 8583–8595.

[8] Noam Shazeer. 2020. GLU Variants Improve Transformer. arXiv:2002.05202 [cs.LG] https://arxiv.org/abs/2002.05202

[9] Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202* (2020).

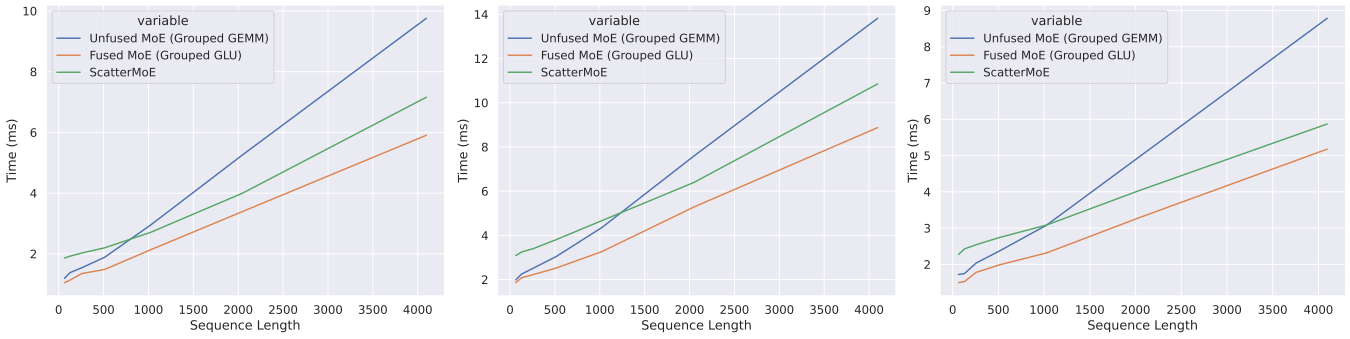
[10] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[11] Shawn Tan, Yikang Shen, Rameswar Panda, and Aaron Courville. 2024. Scattered Mixture-of-Experts Implementation. *arXiv preprint arXiv:2403.08245* (2024).

[12] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (Phoenix, AZ, USA) *(MAPL 2019)*. Association for Computing Machinery, New York, NY, USA, 10–19. doi:10.1145/3315508.3329973

[13] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL] https://arxiv.org/abs/1910.03771

[14] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su,



(a) OLMoE-1B-7B **(b) Ernie4-21B-A3B** **(c) Qwen3-30B-A3B**

Figure 3: Performance results on OLMoE, Ernie 4.5, and Qwen3. We compare three versions: 1. An MoE relying on a triton implementation of a grouped GEMM, with no other operations fused. 2. Our described MoE implementation that fuses the GLU operations. 3. The baseline ScatterMoE, which fuses routing operations with a GEMM, but does not fuse the GLU or activation functions.

Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388* (2025).