Enhancing Performance and Scalability of Large-Scale Recommendation Systems with Jagged Flash Attention

Rengan Xu, Junjie Yang, Yifan Xu, Hong Li, Xing Liu, Devashish Shankar, Haoci Zhang
Meng Liu, Boyang Li, Yuxi Hu, Mingwei Tang, Zehua Zhang, Tunhou Zhang, Dai Li
Sijia Chen, Gian-Paolo Musumeci, Jiaqi Zhai, Bill Zhu, Hong Yan, Srihari Reddy
{renganxu,junjieyang,xuyifan,hongli,xingl,devashish,haocizhang}@meta.com
{mengliu2019,boyangli,yuxihu,mingwt,zehua,tunhouzhang,daili1}@meta.com
{sijiac,gpmusumeci,jiaqi.zhai2,billzh,hong,sriharir}@meta.com
Meta Platforms
Menlo Park, CA, USA

Abstract

The integration of hardware accelerators has significantly advanced the capabilities of modern recommendation systems, enabling the exploration of complex ranking paradigms previously deemed impractical. However, the GPU-based computational costs present substantial challenges. In this paper, we demonstrate our development of an efficiency-driven approach to explore these paradigms, moving beyond traditional reliance on native PyTorch modules. We address the specific challenges posed by ranking models' dependence on categorical features, which vary in length and complicate GPU utilization. We introduce Jagged Feature Interaction Kernels, a novel method designed to extract fine-grained insights from long categorical features through efficient handling of dynamically sized tensors. We further enhance the performance of attention mechanisms by integrating Jagged tensors with Flash Attention. Our novel Jagged Flash Attention achieves up to 9× speedup and 22× memory reduction compared to dense attention. Notably, it also outperforms dense flash attention, with up to 3× speedup and 53% more memory efficiency. In production models, we observe 10% QPS improvement and 18% memory savings, enabling us to scale our recommendation systems with longer features and more complex architectures.

Keywords

Recommendation Systems, Feature Learning, Triton Kernel, Jagged Tensor, Flash Attention, Jagged Flash Attention

ACM Reference Format:
Rengan Xu, Junjie Yang, Yifan Xu, Hong Li, Xing Liu, Devashish Shankar, Haoci Zhang, Meng Liu, Boyang Li, Yuxi Hu, Mingwei Tang, Zehua Zhang, Tunhou Zhang, Dai Li, Sijia Chen, Gian-Paolo Musumeci, Jiaqi Zhai, Bill Zhu, Hong Yan, Srihari Reddy. 2024. Enhancing Performance and Scalability of Large-Scale Recommendation Systems with Jagged Flash Attention. In *18th ACM Conference on Recommender Systems (RecSys '24), October 14–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3640457.3688040

1 Introduction

Categorical features, such as user-clicked items within the last month, are heavily relied upon by ranking models [5, 6, 10–12]. Unlike dense (float) features, which maintain a fixed size across training samples, the length of categorical feature values can vary among different training samples. Padding has traditionally been used to standardize the sizes of these categorical features across various training samples within a batch [7, 8]. However, while padding can be sufficient in some cases, it has inherent drawbacks. These are particularly noticeable with long length categorical inputs, a common input format for large, complex models trained on GPUs. Padding can introduce significant overhead, leading to increased memory usage, computational demands, and communication overhead. This not only affects the model's efficiency but also hampers scalability, particularly in environments with limited resources.

In this paper, we present our efforts in designing and adopting an efficiency-driven approach in the exploration of computationally expensive ranking paradigms. This shift marks a departure from the conventional approach of relying solely on the combination of native PyTorch modules to achieve algorithmic logic. In the rest of the paper, we will discuss the challenges, the methodologies and the lessons learned from our journey. By sharing our experiences, we aim to contribute to the collective knowledge base of the RecSys community, empowering fellow researchers and practitioners to navigate similar challenges in their pursuit of innovation.

2 Methodology

We propose Jagged Feature Interaction Kernel, an innovative method tailored for extracting fine-grained insights from long categorical

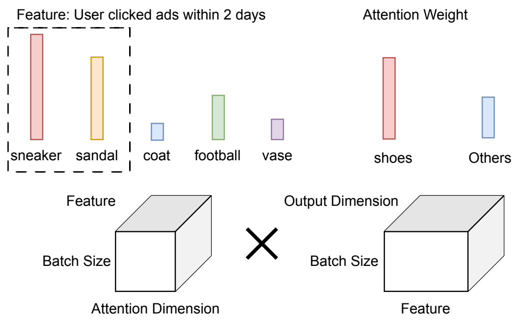

Figure 1: Jagged Feature Interaction Kernel.

Table 1: Triton Kernels Benchmark Result for Jagged Tensor Operators. B: Batch size. sum_B is the jagged dimension with the total sequence length across samples in a batch. Bi: Sequence length for sample i. D: Embedding dimension. T: Hyperparameter. PyTorch version: The implementation with native PyTorch operator which requires padding. Triton version: The implementation with custom Triton operator without padding.

GPU Kernels for Jagged tensors	Description	Version	Memory (MB)	FLOPs (M)	Latency (us)
jagged_dense_bmm	$[sum\_B, D] \times [B, D, T] = [sum\_B, T]$	*PyTorch*	309	838.9	166.6
		Triton	109 (2.83×)	422.5 (1.98×)	99.3 (1.67×)
jagged_jagged_bmm	$[sum\_B, D] \times [sum\_B, T] = [B, D, T]$	*PyTorch*	206	838.9	199.4
		Triton	104 (1.98×)	422.5 (1.98×)	79.1 (2.52×)
jagged_softmax	$sum\_(softmax([Bi, D]))$	*PyTorch*	12.5	4.9	24.9
		Triton	6.3 (1.98×)	2.5 (1.98×)	18 (1.38×)
jagged_jagged_bmm_jagged_out	$[sum\_B, D] \times [sum\_B, D] = [sum\_(Bi * Bi)]$	*PyTorch*	1680	20971	671
		Triton	540 (3.11×)	7200 (2.91×)	293 (2.29×)
array_jagged_bmm_jagged_out	$[sum\_(Bi * Bi)] \times [sum\_B, D] = [sum\_B, D]$	*PyTorch*	4330	20971	755
		Triton	1990 (2.18×)	7200 (2.91×)	585 (1.29×)
jagged2_softmax	$sum\_(softmax([Bi * Bi]))$	*PyTorch*	1530	122.9	2162
		Triton	520 (2.94×)	42.2 (2.91×)	707 (3.06×)

features. Figure 1 demonstrates an overview of our proposed kernel. By focusing on the interactions between feature values and targeting items, Jagged Feature Interaction prioritizes the most relevant feature values, assigning them higher weights. The features are represented with Jagged tensor from TorchRec [3]. The jagged tensor efficiently stores variable-length features from multiple samples in a compact and contiguous manner within memory without the need for padding. We achieve this using two tensors: one for holding all feature values collectively and another offset tensor that determines the sample boundaries for each feature segment.

2.1 Jagged Flash Attention

The flash attention [1, 2] is the state-of-the-art algorithm for accelerating the standard attention. Its core idea is to fuse separate attention operations into a single kernel, minimizing data movement between GPU shared memory and global memory, and maximizing computations within the fast shared memory. Similar to the classic matrix multiplication optimization, flash attention employs tiling optimization to perform two GEMM operations and one softmax block by block. However, applying softmax independently to each block poses a challenge, as it requires the sum of exponentials in the denominator, which depends on information from later blocks. To overcome this challenge, it leverages the online softmax algorithm [4], adjusting results for later blocks as new information becomes available. The flash attention optimization could be applied in both dense and jagged attention. To achieve the best performance and maximize the memory saving, we have combined both jagged tensor and flash attention into jagged flash attention optimization.

2.2 Triton Kernels for Jagged Tensor

We built customized Triton kernels [9] for both forward and backward computations for Jagged tensor operations. Triton is the programming paradigm based on blocked algorithms which can facilitate the construction of high-performance compute kernels for neural networks and allow compilers to aggressively optimize programs for data locality and parallelism. Specifically, we build

- Jagged Tensor (sparse) multiply Jagged Tensor (sparse)
- Jagged Tensor (sparse) multiply Dense Tensor (dense)

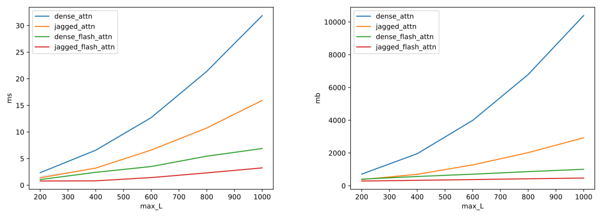

Figure 2: Latency & memory benchmark results for attention. max_L: max sequence length

- Softmax for Jagged Tensor
- MLPs for Jagged Tensor
- Elementwise operations for Jagged Tensor
- Jagged flash attention
- Conversions between Jagged Tensor and Dense Tensor

3 Experiments and Conclusion

Table 1 shows the performance comparison between the custom Triton and the native PyTorch implementations for selective kernels. We demonstrate the relative improvement of Triton over PyTorch in parenthesis. It can be seen that the jagged operators reduce the FLOPs and memory usage significantly and outperform the dense version accordingly.

We also compared different attention implementations with BF16 in Figure 2. The jagged attention mechanism offers significant speedup and memory efficiency improvements over dense attention. Specifically, jagged attention achieves up to 2× speedup compared to dense attention, while jagged flash attention further improves this to up to 9×. Even when compared to dense flash attention, jagged flash attention still offers up to 3× speedup. In terms of memory usage, jagged attention is up to 3.5× more efficient than dense attention, while jagged flash attention reduces memory usage by up to 22×. Notably, the memory usage for both dense and jagged flash attention increases linearly rather than quadratically, with jagged flash attention being up to 53% more memory efficient. These improvements have practical implications for end-to-end model training, where we have observed approximately 10% QPS improvement and 18% memory savings for production models. This enables us to scale our recommendation systems further, accommodating longer features and more complex model architectures.

Acknowledgments

This work is a collective endeavor of many individuals, and the invaluable contributions from the following people (listed in alphabetical order): Adnan Akhundov, Wei Chen, Yang Chen, Lu Fang, Carl Hu, Yuzheng Huang, Mu-Chu Lee, Bert Maher, Andrey Malevich, Sarunya Pumma, Ketan Singh, Adele Sun, Xiaodong Wang, Wei Wei, Xinfeng Xie, Jackie Xu, Chunzhi Yang, and Mengchi Zhang. We would also like to thank Sandeep Pandey, Santanu Kolay, Ajit Mathew, Damien Sereni, Ian Barber, Colin Taylor, Adnan Aziz for their leadership support.

References

[1] Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[2] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 16344–16359.

[3] Dmytro Ivchenko, Dennis Van Der Staay, Colin Taylor, Xing Liu, Will Feng, Rahul Kindi, Anirudh Sudarshan, and Shahin Sefati. 2022. Torchrec: a pytorch domain library for recommendation systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 482–483.

[4] Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867* (2018).

[5] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).

[6] Mingwei Tang, Meng Liu, Hong Li, Junjie Yang, Chenglin Wei, Boyang Li, Dai Li, Rengan Xu, Yifan Xu, Zehua Zhang, et al. 2024. Async Learned User Embeddings for Ads Delivery Optimization. *arXiv preprint arXiv:2406.05898* (2024).

[7] PyTorch Team. 2021. The nestedtensor package prototype:Readme.md. https://github.com/pytorch/nestedtensor/blob/master/nestedtensor/csrc/README.md.

[8] Tensorflow Team. 2022. Ragged Tensors. https://www.tensorflow.org/api_docs/python/tf/RaggedTensor?version=nightly.

[9] Philippe Tillet, Hsiang-Tsung 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*. 10–19.

[10] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. *arXiv preprint arXiv:2402.17152* (2024).

[11] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. 2024. Wukong: Towards a Scaling Law for Large-Scale Recommendation. *arXiv preprint arXiv:2403.02545* (2024).

[12] Buyun Zhang, Liang Luo, Xi Liu, Jay Li, Zeliang Chen, Weilin Zhang, Xiaohan Wei, Yuchen Hao, Michael Tsang, Wenjun Wang, et al. 2022. DHEN: A deep and hierarchical ensemble network for large-scale click-through rate prediction. *arXiv preprint arXiv:2203.11014* (2022).