In Defense of Structural Sparse Adapters for Concurrent LLM Serving

Junda Su
Rice University
js202@rice.edu

Zirui Liu
Rice University
zl105@rice.edu

Zeju Qiu and Weiyang Liu
Max Planck Institute for Intelligent Systems
zeju.qiu@tuebingen.mpg.de
wl396@cam.ac.uk

Zhaozhuo Xu
Stevens Institute of Technology
zxu79@stevens.edu

Abstract

Adapting large language models (LLMs) to specific tasks remains challenging due to the extensive retraining required, prompting the need for efficient adapter techniques. Despite this, the concurrent serving of multiple adapters, each with unique matrix shapes, poses significant system-level challenges. To address these issues, we identify an opportunity in structurally sparse adapters, which, unlike low-rank adapters, maintain consistent matrix shapes while varying in sparsity patterns. Leveraging this characteristic, we introduce SpartanServe, a system designed for efficient concurrent serving of LLMs using multiple structurally sparse adapters. SpartanServe employs a unified matrix multiplication operation and a novel memory management technique to enable effective batching. Furthermore, the incorporation of Triton kernels enhances the acceleration of matrix multiplication in the serving process. Experimental results demonstrate that SpartanServe achieves 2.12× speedup over S-LoRA when serving 96 adapters using a single NVIDIA A100 GPU (40GB), showcasing its efficacy in concurrent LLM serving.

1 Introduction

As the field of natural language processing (NLP) continues to advance, large language models (LLMs) (Brown et al., 2020) have emerged as a cornerstone technology, powering a wide range of applications from automated customer service (Pandya and Holia, 2023) to sophisticated content generation (Imani et al., 2023).

The Need for Adapters. Large Language Models (LLMs) require adapters like Low-Rank Adaptation (LoRA) (Hu et al., 2021) to efficiently fine-tune their performance on specific tasks while managing computational and resource constraints. LLMs, with their extensive parameters and capabilities, still face challenges in adapting to new or niche applications without exhaustive retraining. Adapters offer a solution by introducing lightweight, task-specific adaptations that modify only a subset of the model's parameters (Dettmers et al., 2024; Liu et al., 2024; Qiu et al., 2023; Liu et al., 2023). This approach significantly reduces the memory and computational power needed for fine-tuning, making it feasible to deploy LLMs in varied and resource-limited environments.

Expensiveness of Multi-Adapter Concurrent LLM Serving. With the rise of LLM adaptation techniques, serving multiple adapters on the same base LLM has become standard practice. As a consequence, finding efficient methods to serve several adapters concurrently has become an interesting research area. One intuitive approach to efficient adapter serving is batching the input requests and their corresponding adapters for faster inference. However, this concurrent serving of multiple adapters is costly due to the irregularity of the adapters. Specifically, adapters like LoRA require specifying the hidden low-rank dimension as a hyper-parameter, resulting in different adapters having different low-rank matrix shapes. To address this issue, Sheng et al. (2024) proposed S-LoRA, which tackles the heterogeneous shapes of low-rank matrices. Despite these significant system-level improvements, challenges persist in concurrently serving LLMs with numerous, heterogeneous LoRA adapters.

An Opportunity from Structural Sparse Adapters. In this work, we identify an opportunity for efficient adapter batching presented by structurally sparse adapters (Qiu et al., 2023; Liu et al., 2023). We argue that there is a fundamental difference between structurally sparse adapters and low-rank adapters: for the same large language model (LLM), structurally sparse adapters maintain the same matrix shape but exhibit different sparsity patterns. Leveraging this advantage, we propose a unified matrix multiplication approach

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 4948–4953

for concurrent LLM serving.

Our Contributions. In this work, we propose SpartanServe, a system designed for concurrent LLM serving using multiple structurally sparse adapters. We present a unified matrix multiplication operation for these adapters, along with a memory management technique that enables efficient batching. Additionally, we incorporate Triton kernels to further accelerate matrix multiplication in the concurrent LLM serving process. We show that SpartanServe is able to achieve 2.12× speedup over S-LoRA when serving 96 adapters using a single NVIDIA A100 GPU (40GB).

2 Structural Sparse LLM Adapters

2.1 LLM Fine-Tuning

Fine-tuning allows pre-trained LLMs to adapt to specific tasks through retraining on domain specific data and updating model weights. Given the substantial parameter counts in LLMs, fine-tuning models in entirety is resource-intensive. Thus, parameter-efficient fine-tuning (PEFT) approaches (Hu et al., 2021; Dettmers et al., 2024; Qiu et al., 2023) have garnered considerable interest. These PEFT methods involve modifying or inserting only a small set of parameters, reducing computational costs and making the fine-tuning process more feasible for practical applications.

2.2 Low-Rank Adaptation and Variations

Low-Rank Adaptation (LoRA) (Hu et al., 2021) represents one category of PEFT methods that has attracted momentum. The fundamental principle of LoRA is based on the hypothesis that the essential updates required during the model adaptation process can be encapsulated in a low-dimensional space. In practice, LoRA implements this by freezing the original weights of the model and introducing two smaller, trainable matrices whose product forms a low-rank approximation of the weight changes. This method significantly reduces the quantity of trainable parameters, decreasing both memory overhead and computational expense. Consequently, LoRA maintains comparable accuracy to traditional full-parameter fine-tuning but with much lower resource demands. Several variants of LoRA have been developed to enhance adapter efficiency such as Adaptive Low-Rank Adaptation (AdaLoRA) (Zhang et al., 2023) and Weight-Decomposed Low-Rank Adaptation (DoRA) (Liu et al., 2024).

We argue that low-rank style adapters are facing a similar challenge in LLM concurrent serving: the matrix shapes of different adapters are not the same. Specialized optimizations are needed to improve system-level performance (Sheng et al., 2024).

2.3 Structural Sparse LLM Adapters

Besides low-rank matrices, another class of structured matrices used for compressed matrix representation is structural sparse matrices, which only store the nonzero elements and their indices, thus drastically reducing the memory required to store the matrices' information.

Butterfly Orthogonal Fine-Tuning (BOFT) is one such fine-tuning method, using block-sparse matrices to achieve parameter-efficiency. It builds upon Orthogonal Fine-Tuning (OFT), which applies the insight that orthogonal transformations preserves hyper-spherical energy by maintaining the pair-wise angle between neurons, and leverages butterfly factorization to efficiently parameterize dense orthogonal matrices. Butterfly factorization was used in the Cooley-Tukey Fast Fourier Transform Algorithm (Cooley and Tukey, 1965), which uses a recursive structure to write a matrix in $R^{d\times d}$ as the product of sparse matrix products. This method has been adopted in other works of fast linear transforms and efficient training (Chen et al., 2021; Dao et al., 2019, 2022).

Formally, we start by defining a butterfly factor $B^F(k)$ as $B^F(k) = \begin{bmatrix} D_1(k/2) & D_2(k/2) \\ D_3(k/2) & D_4(k/2) \end{bmatrix}$, where each $D_i(k/2)$ is a diagonal matrix in $R^{\frac{k}{2}\times\frac{k}{2}}$. Each butterfly component $\tilde{B}(d,k) \in R^{d\times d}$ is a block diagonal matrix composed of $\frac{d}{k}$ butterfly factors of size $k \times k$. i.e.:

$$\tilde{B}(d,k) = diag(B_1^F(k)...B_{d/k}^F(k))$$

Using butterfly factorization, for a dense matrix $B(d) \in R^{d\times d}$, we can write it as

$$B(d) = \tilde{B}(d,d)\tilde{B}(d,d/2)...\tilde{B}(d,2)$$

where each $\tilde{B}(d,k), k > 2$ is a sparse matrix with fixed sparsity pattern. Each factor's non-zero pattern can be created by a block-wise permutation of the $\tilde{B}(d,2)$ non-zero patterns. As a result, we can preserve the orthogonality of the dense matrix $B(d)$ by enforcing the blocks of $\tilde{B}(d,2)$ to be orthogonal matrices. This can be generalized to using block butterfly components where each $B^b(d,k)$ is composed of block butterfly factors of $k\times k$ blocks,

with each block having a size of $b \times b$. Using this formulation, a foward pass using a BOFT adapter can be expressed as:

$$z = ((\prod_{i=1}^{m} \tilde{B}^b(d,i) \cdot W_{base})^T x$$

3 SpartanServe: Concurrent Serving of Multiple Structural Sparse Adapters

3.1 Our Proposed Optimization

Unified Operation for Multiple Structural Sparse Adapters. BOFT adapters offers an opportunity for simple batching in transformer-based models. These adapters are particularly suitable to unified operations due to their inherent structural properties. This simplicity in batching originates from the fact that each butterfly component of a BOFT adapter can be represented in a dense format as square matrices, where both the rows and columns correspond to the size of the input shape. In this section, we introduce our proposed method, SpartanServe, emphasizing how the consistent shape in BOFT adapters enables a unified operation that facilitates efficient adapter serving.

Adapter Representation. At every decoder layer, each BOFT adapter can be represented as a $B \times D \times D$ tensor, where B represents the number of butterfly components, and D is the hidden size. Each $D \times D$ component is a block sparse matrix with a block size of K, which is a hyperparameter that affects the sparsity patterns of the butterfly components.

The vanilla implementation stored butterfly components in a block sparse format. This means that only one $N \times K \times K$ matrix is stored, where N denotes the number of non-zero blocks. At inference time, these N blocks are used to reconstruct the full $D \times D$ matrix. This does not leverage the memory efficiency of sparse matrices during inference, and creates slowdowns due to the matrix reconstruction process.

To optimize memory usage and compute efficiency during inference, SpartanServe stores two parts of the $D \times D$ butterfly components: the first part is the $N \times K \times K$ tensor, which contains the non-zero weight values, arranged in a row-wise traversal order; the second component is a $(D/K) \times (D/K)$ *layout* matrix, where each element is a binary value indicating the presence (1) or absence (0) of a non-zero block in the corresponding block position in the $D \times D$ matrix.

Adapter Batching. For batching M adapters together, SpartanServe consolidates the block sparse weights into a single tensor, and merges the layouts of each individual tensor into a unified layout. The unified layout thus has a shape of $M \times (D/K) \times (D/K)$. During inference, the batched adapters are multiplied with the computation result from the base model. For this purpose, we applied Triton's block sparse matrix multiplication, which generates a Triton kernel based on the created unified layout and block size to facilitate efficient batch computation. Figure 1 illustrates the adapter batching pipeline.

Notably, when different adapters have different block sizes, we cannot simply batch the non-zero blocks together due to their varying shape. To address this issue, we select a maximal common block size that divides all block sizes. Using this common block size, we generated a layout for each butterfly component, which specifies the placement of each block within the sparse matrix relative to its position in the corresponding full matrix. Figure 1 demonstrates batching multiple adapters with different block sizes.

Memory Management. The batching of BOFT adapters requires the creation of a new adapter tensor, effectively doubling the number of adapters in the GPU memory. This increase often leads to out-of-memory issues when batching multiple adapters simultaneously. We developed a solution where adapter weights are initially loaded onto CPU memory. Only the necessary adapter weights are loaded to GPU memory during the batching process and subsequently offloaded to the CPU upon completion. This approach limits the number of adapters in a single batch to the capacity of the GPU memory at the batch's conclusion, rather than the considerably higher memory demands during the batching process itself, mitigating the risk of memory overflow during extensive batching operations.

Speed Up Triton Kernels. Due to the design of the JIT compiler used to create the Triton (Tillet et al., 2019) block-sparse matrix multiplication API, initializing the kernel induces significant overhead even with warm-up, causing inference to slow down. To address this issue, we used PyTorch's CUDA graph integration to optimize kernel launching. CUDA graph enables us to define and store CUDA kernels as a single unit, rather than a sequence of individually launched operations. This allows us to launch Triton kernels in one single CPU operation, reducing launch overheads.

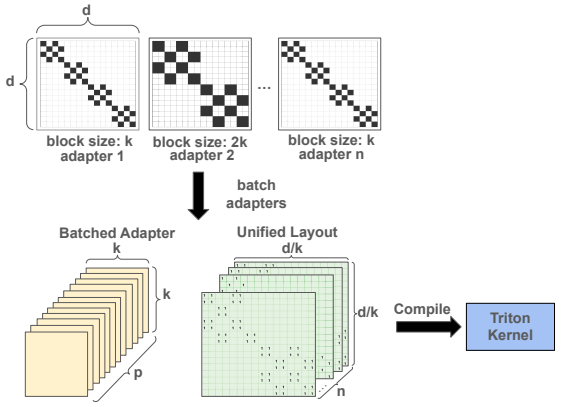

Figure 1: Batching adapters of different block size. We use BOFT adapters with 1 butterfly component as an example. The top three matrices represent butterfly components with different block sizes (k, 2k, and k). The stack of yellow squares represent batched non-zero matrices with the maximal common block size (k), with p denoting the total number of such blocks. The stack of green squares represent the unified layout for all adapters. The unified layout is then used to create a Triton kernel for block sparse matrix multiplication between base model outputs and batched adapters.

4 Experiment

4.1 Settings

Models. We evaluated BOFT adapter inference performance on foundation models using LLama2-7B (Touvron et al., 2023), one of the most popular generative text models that uses transformer architecture as a core component. The adapters are added to the "k_proj", "q_proj", "v_proj", "o_proj" modules in each self-attention layer. We considered 3 different model and adapter configurations, listed in Table 1. The settings are chosen based on previously reported parameter count ratio that yielded similar performance between BOFT and LoRA (Liu et al., 2023). The evaluation was conducted with a single A100 GPU with 40GB of memory.

Table 1: Model and adapter settings. m denote the number of butterfly components used to create the BOFT adapter, b denote the block size. The %Params column denotes the percentage of parameters of one adapter compared to the base model parameter count.

Setting	BOFT			LoRA	
	m	b	%Param	Rank	%Param
1	2	32	0.48%	64	0.96%
2	2	{64, 32}	0.72%	64	0.96%

Baselines. We evaluated our results against two systems capable of serving multiple LoRA adapters: Huggingface PEFT (Mangrulkar et al., 2022) and S-LoRA (Sheng et al., 2024).

Huggingface PEFT is a framework engineered to adapt extensive pretrained models to diverse tasks while optimizing resource utilization. This library collaborates seamlessly with various other libraries such as Transformers, Diffusers, and Accelerate. Although it supports the batching of multiple adapters, its efficiency in this regard is limited.

S-LoRA is designed for the scalable deployment of multiple LoRA adapters across single or multiple GPUs. It achieves high adapter capacity by hosting adapters on main memory and dynamically loading requested ones to GPU memory. S-LoRA also improves throughput and latency through the implementation of Unified Paging and specialized CUDA kernels. However, it operates independently and lacks integration with other large language model frameworks and libraries.

Dataset. We created a dataset comprised of 960 text data samples. These samples have lengths ranging from 10 to 50 words, intentionally structured to simulate a spectrum of user queries that might be observed in real-world settings. In each experiment setting, for the n input text data samples, the corresponding adapter configuration was selected using a round-robin approach. For each request, we set the output length to 100 tokens.

4.2 Main Results

Comparison with Huggingface PEFT. We compare SpartanServe and Huggingface PEFT LoRA adapters for performing inference with multiple adapters in terms of request latency. BOFT adapters consistently exhibit lower latency, as shown in Table 2. Furthermore, as the number of adapters scales up, the latency associated with Huggingface PEFT LoRA adapters increases, whereas SpartanServe maintain a stable latency profile. Here, we don't compare with Huggingface PEFT using larger number of adapters as it already performs much slower under small adapter counts.

Comparison with S-LoRA We evaluated SpartanServe and S-Lora on performing inference for 512 user requests with varying numbers of adapters. We use setting 1 for comparing larger number of adapters as setting 2 would cause out-of-memory issues for SpartanServe when using one 40GB GPU. In Table 3, we present the average latency comparison between SpartanServe and S-LoRA. Our

Table 2: Average latency (s/req) for SpartanServe and Huggingface PEFT when serving multiple adapters

Setting	# adapters	SpartanServe	LoRA
S1	2	19.23	13.77
S1	4	8.99	22.22
S1	16	18.90	76.28
S1	32	11.67	142.24
S1	64	12.10	260.79

Table 3: Average latency (s/req) for SpartanServe and S-LoRA when serving multiple adapters

Setting	# adapters	SpartanServe	S-LoRA
S2	2	17.88	13.27
S2	4	21.51	12.46
S2	16	23.07	15.12
S2	32	19.23	22.33
S2	64	21.78	27.49
S1	72	13.04	29.52
S1	84	14.14	30.37
S1	96	15.30	32.46

results indicate that as the number of adapters increases, SpartanServe exhibits lower latency. When serving 96 adapters, SpartanServe is able to achieve 2.12 $\times$ speedup compared to S-LoRA, demonstrating its concurrent adapter serving ability.

5 Conclusion

In this work, we study the concurrent LLM serving with multiple adapters. We have explored the potential of structurally sparse adapters, which maintain consistent matrix shapes while varying in sparsity patterns, unlike low-rank adapters. This insight led to the development of SpartanServe, a system designed for the efficient concurrent serving of LLMs using multiple structurally sparse adapters. SpartanServe leverages a unified matrix multiplication operation and an innovative memory management technique to enable effective batching. Additionally, the use of Triton kernels enhances the acceleration of matrix multiplication during the serving process. Experimental results demonstrate that SpartanServe achieves 2.12$\times$ speedup over S-LoRA when serving 96 adapters using a single NVIDIA A100 GPU (40GB).

Limitations

In our existing batching mechanism, different users can specify varying block sizes for their adapters; however, the batching process is constrained by the requirement that all adapters share the same number of butterfly factors. Due to the recursive characteristics of butterfly factorization, it is theoretically feasible to merge consecutive butterfly factors while maintaining the sparsity of the resulting butterfly factor, thus aligning the butterfly factor count across all batched adapters. Exploring efficient methods for such butterfly factor merging represents a promising avenue for future research.

Ethics Statement

The primary focus of our research was on improving model performance during inference time, with minimal direct ethical concerns. We expect that the methodologies developed will foster more sustainable and efficient use of natural resources in machine learning. However, this progress requires careful regularization to prevent potential misuses in harmful applications. Such considerations are essential for ensuring that the deployment of new technologies is aligned with societal well-being and ethical standards.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. 2021. Pixelated butterfly: Simple and efficient sparse training for neural network models. In *International Conference on Learning Representations*.

James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. 2022. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. 2019. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, pages 1517–1527. PMLR.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 37–42.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, et al. 2023. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Keivalya Pandya and Mehfuza Holia. 2023. Automating customer service using langchain: Building custom open-source gpt chatbot for organizations. *arXiv preprint arXiv:2310.05421*.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2023. Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2024. Slora: Scalable serving of thousands of lora adapters. *Proceedings of Machine Learning and Systems*, 6:296–311.

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*, pages 10–19.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview.