In Defense of Structural Sparse Adapters for Concurrent LLM Serving

Junda Su[1] Zirui Liu[1] Zeju Qiu[2] Weiyang Liu[3] Zhaozhuo Xu[4]

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 & 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, building multiple adapters on the same base LLM has become standard practice. Consequently, during the inference phase, we can batch input requests with independent adapters together for faster processing. 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 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 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 serv-

[*]Equal contribution [1]Rice University [2]Technical University of Munich [3]University of Cambridge & Max Planck Institute for Intelligent Systems [4]Stevens Institute of Technology. Correspondence to: Zhaozhuo Xu <zxu79@stevens.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

ing process. We show that SpartanServe is able to achieve $2.12\times$ 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 adapt to specific tasks through retraining on domain specific data and updating the model weights. Given the substantial parameter counts in LLMs, fine-tuning these models in entirety is resource-intensive. Consequently, parameter-efficient fine-tuning (PEFT) approaches (Hu et al., 2021; Dettmers et al., 2024; Qiu et al., 2023) have garnered considerable interest.

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 intrinsic rank hypothesis, which posits 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, often by several orders of magnitude, thereby 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 the fine-tuning of pre-trained language models. Adaptive Low-Rank Adaptation (AdaLoRA) conserves resources by assigning higher ranks to crucial matrices while pruning less significant ones(Zhang et al., 2023). Another variant, Weight-Decomposed Low-Rank Adaptation (DoRA) separates weight updates into two components: magnitude and direction (Liu et al., 2024), allowing DoRA to more effectively optimize weight adjustments, potentially enhancing model performance at lower ranks.

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, which uses a recursive structure to write a matrix in $\Re^{d\times d}$ 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 $\Re^{\frac{k}{2}\times\frac{k}{2}}$. Each butterfly component $\tilde{B}(d,k) \in \Re^{d\times d}$ is a block diagonal matrix composed of $\frac{d}{k}$ butterfly factors of size $k \times k$. i.e.:

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

Using butterfly factorization, for a dense matrix $B(d) \in \Re^{d\times d}$, 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. LLM Concurrent Serving

Efficient deployment of Transformer-based Large Language Models (LLMs) in production environments necessitates meticulous management of computational resources and user requests. In practical applications, LLMs must handle multiple concurrent user requests, with each user potentially employing a distinct pretrained adapter while sharing a common base model. Traditional methods, which process requests sequentially, fail to fully utilize GPU capabilities, resulting in considerable inefficiencies and prolonged response times.

To address these inefficiencies, batching adapters with varying configurations for processing becomes crucial. Nevertheless, integrating batching adapters introduces a new challenge due to the diversity of user configurations. Low-rank adapters, in particular, face specific difficulties when users specify varying ranks, leading to unaligned matrix dimensions that complicate batching operations. Effectively managing this issue necessitates advanced batching mechanisms, which may involve padding and fragmentation to standardize input dimensions, or the development of sophisticated kernels designed to handle discrepancies in adapter configurations.

3.2. Our Proposed Optimization

Unified Operation for Multiple Structural Sparse Adapters. BOFT adapters offers an opportunity in simple batching capabilities within transformer-based models. These adapters are particularly amenable to unified operations due to their inherent structural properties.

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. This uniformity in matrix dimensions across different adapters simplifies the process of batching.

Adapter Batching. At each layer, each BOFT adapter is structured in a block sparse format, where only non-zero blocks are stored. However, different adapters may have different block sizes. To address this issue, we select a minimal common block size that divides the original block size. 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.

Subsequently, these block sparse weights are consolidated into a single tensor, and the layouts of individual adapters are merged into a unified layout. During model inference, the batched adapters are multiplied with the computation result of 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.

Memory Management. The batching of these 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 transferred to the GPU for the duration of 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.

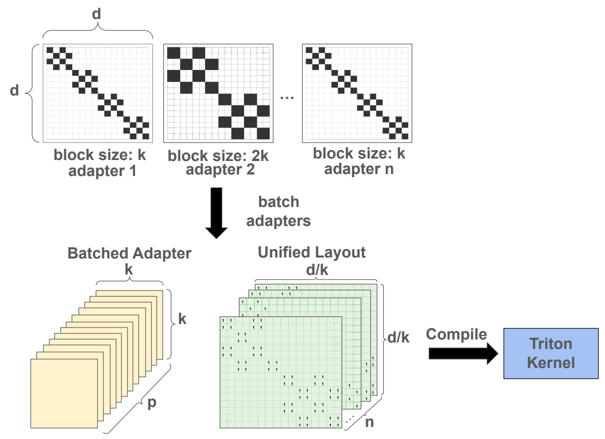

Figure 1. Batching adapters of different block size at one butterfly factor: although different users specified different block sizes, we are able to use the minimum common block size to create a unified layout for all adapters. Within the unified layout, each of the n matrices is a boolean matrix. The individual entries of matrix i in the layout signify whether the corresponding block in adapter i is non-zero. The unified layout is then used to create a Triton kernel for block sparse matrix multiplication.

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 on a server 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 text data of varying lengths, 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.

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

Comparison with S-LoRA We evaluated SpartanServe and S-Lora for performing inference on 512 user requests with varying numbers of adapters. We use setting 1 when comparing larger number of adapters as setting 2 would cause out-of-memory issues for SpartanServe when using one 40GB GPU. Our results indicate that as the number of adapters increases, SpartanServe exhibits lower latency, as shown in Table 3.

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

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).

References

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, B., Dao, T., Liang, K., Yang, J., Song, Z., Rudra, A., and Re, C. Pixelated butterfly: Simple and efficient sparse training for neural network models. In *International Conference on Learning Representations*, 2021.

Dao, T., Gu, A., Eichhorn, M., Rudra, A., and Ré, C. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, pp. 1517–1527. PMLR, 2019.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2024.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

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

Liu, S.-Y., Wang, C.-Y., Yin, H., Molchanov, P., Wang, Y.-C. F., Cheng, K.-T., and Chen, M.-H. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

Liu, W., Qiu, Z., Feng, Y., Xiu, Y., Xue, Y., Yu, L., Feng, H., Liu, Z., Heo, J., Peng, S., et al. Parameter-efficient orthogonal finetuning via butterfly factorization. *arXiv preprint arXiv:2311.06243*, 2023.

Mangrulkar, S., Gugger, S., Debut, L., Belkada, Y., Paul, S., and Bossan, B. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

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

Qiu, Z., Liu, W., Feng, H., Xue, Y., Feng, Y., Liu, Z., Zhang, D., Weller, A., and Schölkopf, B. Controlling text-to-image diffusion by orthogonal finetuning. *Advances in Neural Information Processing Systems*, 36:79320–79362, 2023.

Sheng, Y., Cao, S., Li, D., Hooper, C., Lee, N., Yang, S., Chou, C., Zhu, B., Zheng, L., Keutzer, K., et al. Slora: Scalable serving of thousands of lora adapters. *Proceedings of Machine Learning and Systems*, 6:296–311, 2024.

Tillet, P., Kung, H.-T., and Cox, D. 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*, pp. 10–19, 2019.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023.

Zhang, Q., Chen, M., Bukharin, A., He, P., Cheng, Y., Chen, W., and Zhao, T. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview, 2023.