HydraCache: LLM Inference Prefill Parallelization through Distributed Cache Blending

Adib Rezaei Shahmirzadi*
adibzi@vt.edu
Virginia Tech
Blacksburg, VA, USA

Shayan Shabihi*
shabihi@umd.edu
University of Maryland
College Park, MD, USA

Mona Moghadampanah
monamp@vt.edu
Virginia Tech
Blacksburg, VA, USA

Furong Huang
furongh@umd.edu
University of Maryland
College Park, MD, USA

Dimitrios S. Nikolopoulos
dsn@vt.edu
Virginia Tech
Blacksburg, VA, USA

Abstract

The prefill phase of Large Language Model (LLM) inference, where the input prompt is processed to generate a Key-Value (KV) cache, is a critical latency bottleneck for input sequences. Existing serving architectures face a trade-off: Data Parallelism (DP) offers flexibility but cannot accelerate a single long prompt, while Tensor Parallelism (TP) parallelizes prefill but at the cost of rigid resource allocation and constant communication overhead at each layer. We introduce HydraCache, a system that resolves this problem by enabling a cluster of independent, data-parallel model replicas to collaborate on-demand to parallelize the prefill of a single long prompt. Our core contribution is DistBlendAttention, a lightweight mechanism that fuses distributed KV caches with minimal communication, avoiding the prohibitive overheads of both TP and traditional Sequence Parallelism. Our evaluation shows that HydraCache significantly reduces Time-To-First-Token (TTFT) up to 7x for requests and enables flexible, SLO-aware serving.

1 Introduction

The compute-intensive prefill phase, with its quadratic complexity in attention [7], creates a dominant latency bottleneck for long contexts, contrasting with the memory-bound decode phase [6]. While systems using techniques like chunked prefill can improve overall throughput [1, 10], they do not fundamentally accelerate a single long prompt, forcing a difficult choice between competing parallelization paradigms. Data Parallelism (DP), while flexible, cannot parallelize a single request, leading to high TTFT. Conversely, Model Parallelism strategies like Tensor Parallelism (TP) [5] and Pipeline Parallelism (PP) [4] parallelize prefill but at the cost of rigid resource allocation, constant communication overhead (TP), or efficiency-reducing pipeline bubbles (PP). Finally, the most direct approach, Sequence Parallelism (SP), is rendered impractical by the prohibitive network cost of transferring massive KV caches between GPUs [2, 8].

To break this paradigm, we propose **HydraCache**, a system that enables practical sequence parallelism on a flexible, decoupled DP architecture. HydraCache partitions a long input sequence, not the model, across multiple independent GPU instances. These instances concurrently compute partial KV caches, which are then efficiently synchronized using our novel DistBlendAttentionmechanism. This approach provides the prefill acceleration of parallelism without sacrificing the architectural flexibility of DP, enabling a hybrid, SLO-aware serving model. Our contributions are:

- A novel sequence-parallel prefill strategy built on a flexible data-parallel foundation.
- DistBlendAttention, a mechanism to fuse distributed KV caches with minimal, minimal communication overhead.
- An evaluation demonstrating significant TTFT reduction for requests compared to both DP and TP baselines.

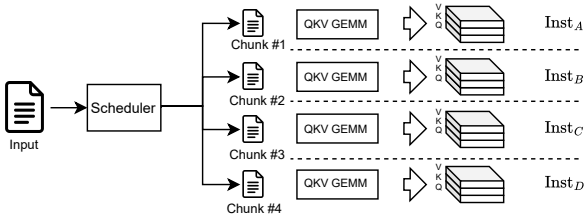


Figure 1: Input sequence is split into chunks and dispatched to independent instances for parallel prefilling.

2 System Design

HydraCache parallelizes prefill in a three-stage process: parallel initial computation, selective token identification, and efficient, pipelined cache blending.

1. Sequence Partitioning & Parallel Prefill. As shown in Figure 1, a single long input sequence is split into N chunks. Each chunk is dispatched to an independent GPU instance (hosting a full model replica) and prefilled concurrently. This initial step is completely parallel and communication-free, but results in N partial, isolated KV caches that cannot attend to each other.

2. Identifying Cross-Chunk Dependencies. To synchronize these caches with minimal overhead, we identify only the subset of tokens most in need of global context. By recomputing the first attention layer and measuring deviation, we mark the top $\approx 15\%$ of tokens for synchronization in subsequent layers. This heuristic is shown to result in less than 1% generation accuracy loss [9].

3. Pipelined Blending via DistBlendAttention. To fuse the knowledge from the distributed caches for the marked tokens, we

*Equal contribution.

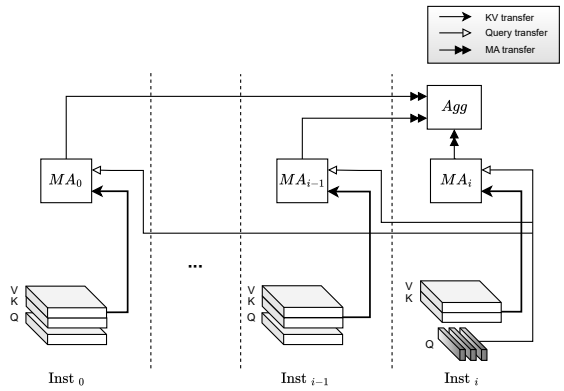


Figure 2: The DistBlendAttention mechanism: instances exchange queries (Q), compute local Micro-Attentions (MA), and aggregate them.

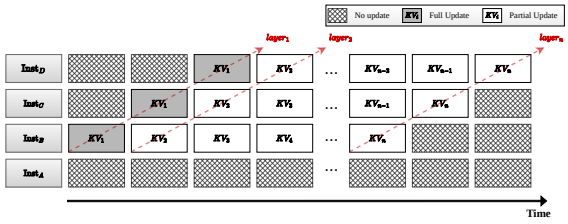


Figure 3: Pipelining the blending process across layers and GPU instances to overlap computation and communication.

use our **DistBlendAttention** mechanism, illustrated in Figure 2. Instead of transferring large KV caches, this process involves:

- **Query Exchange:** Instances exchange only the small query vectors of the marked tokens.
- **Local MA Computation:** Each instance computes a partial Micro-Attention (MA) [3] output using its local KV chunk.
- **Aggregation:** These small MA results are aggregated via a fast All-Reduce to reconstruct the globally correct attention output, which is then used to update the local KV caches.

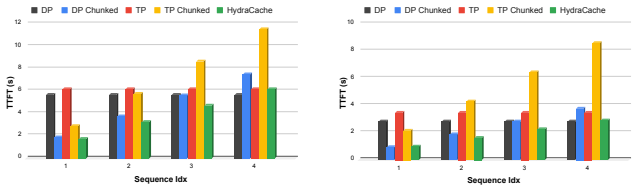
To execute this layer-by-layer blending efficiently, we schedule the operations in a pipeline across instances, as shown in Figure 3. This overlaps the communication of one layer with the computation of the next, effectively hiding network latency and maximizing GPU utilization during the synchronization phase.

3 Evaluation

We evaluate HydraCache on an NVIDIA DGX node with 4x A100 GPUs. Baselines include a standard Data Parallel (DP) setup and a Tensor Parallel (TP) setup. All methods using chunking have a chunk size of 500. We use Llama 3.1 8B for all results.

3.1 SLO-Aware Prioritization & TTFT Reduction

We measure the TTFT for each sequence in a batch. As shown in Figure 4, HydraCache's parallel prefill delivers the first token for the highest-priority request (and the subsequent higher priority sequences among the four total requests) significantly faster than the baselines.

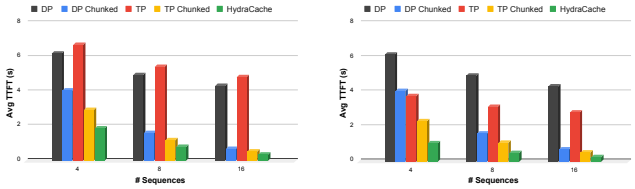


(a) 2 GPUs **(b) 4 GPUs**

Figure 4: TTFT per sequence index. HydraCache prioritizes the first sequence, delivering its first token faster.

3.2 Scaling with Number of Sequences

We fix the total context length (128k tokens) and vary the number of sequences. Figure 5 demonstrates that HydraCache achieves the lowestt average TTFT across all batch sizes, and its performance advantage grows as the workload becomes more distributed (as demonstrated for the cases having 2 and 4 GPUs for the parallelization, respectively).

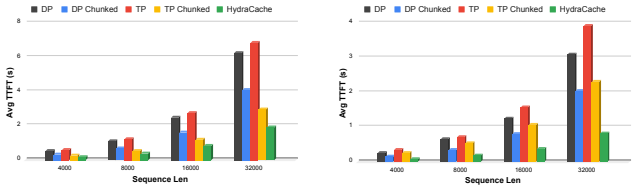


(a) 2 GPUs **(b) 4 GPUs**

Figure 5: Average TTFT vs. number of sequences. HydraCache demonstrates superior parallelization efficiency.

3.3 Scaling with Sequence Lengths

We fix the batch size and increase the sequence length. Figure 6 demonstrates that HydraCache's latency scales far better with longer contexts, much better countering the quadratic prefill bottleneck that slows down baselines.



(a) 2 GPUs **(b) 4 GPUs**

Figure 6: Average TTFT vs. sequence length. HydraCache effectively mitigates the quadratic complexity of prefill.

4 Conclusion and Future Work

HydraCachepresents an effective solution to the problem of efficient hybrid serving of LLMs. By enabling parallellized prefilling on a flexible data-parallel architecture, it significantly reduces TTFT for long prompts while retaining the ability to serve standard requests with high throughput. HydraCache's lightweight DistBlendAttentionmechanism is key to this performance, achieving parallelization without the high costs of existing methods. For future work, we plan a comprehensive evaluation of our approach on decoding and hybrid batching scenarios.

References

[1] R. Agrawal, S. Subramanya, D. Narayanan, M. Zaharia, and I. Stoica. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[2] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Reza Yazdani Aminadabi, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2024. System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing* (Nantes, France) *(PODC '24)*. Association for Computing Machinery, New York, NY, USA, 121–130. doi:10.1145/3662158.3662806

[3] Bin Lin, Chen Zhang, Tao Peng, Hanyu Zhao, Wencong Xiao, Minmin Sun, Anmin Liu, Zhipeng Zhang, Lanbo Li, Xiafei Qiu, Shen Li, Zhigang Ji, Tao Xie, Yong Li, and Wei Lin. 2024. Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache. arXiv:2401.02669 [cs.DC] https://arxiv.org/abs/2401.02669

[4] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient large-scale language model training on GPU clusters using megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. doi:10.1145/3458817.3476209

[5] D. Narayanan, M. Shoeybi, M. Patwary, et al. 2021. Efficient Large-Scale Language Model Training on GPU Clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[6] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132. doi:10.1109/ISCA59077.2024.00019

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[8] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (Austin, TX, USA) *(SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 640–654. doi:10.1145/3694715.3695948

[9] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 94–109. doi:10.1145/3689031.3696098

[10] X. Zhong et al. 2024. DistServe: Disaggregating Prefill and Decoding for Optimized LLM Serving. *arXiv preprint arXiv:2401.09670* (2024).