Deep Kernel Fusion for Transformers

Zixi Zhang
Imperial College London
London, UK
b.zhang25@imperial.ac.uk

Zhiwen Mo
Imperial College London
London, UK
zhiwen.mo25@imperial.ac.uk

Yiren Zhao
Imperial College London
London, UK
a.zhao@imperial.ac.uk

Robert Mullins
University of Cambridge
Cambridge, UK
robert.mullins@cl.cam.ac.uk

Abstract

Agentic LLM inference with long contexts is increasingly limited by memory bandwidth rather than compute. In this setting, SwiGLU MLP blocks, whose large weights exceed cache capacity, become a major yet under-optimized bottleneck in the Transformer architecture. We propose **DeepFusionKernel**, a deeply fused kernel that cuts HBM traffic and boosts cache reuse, delivering up to 13.2% speedup on H100 and 9.7% on A100 over SGLang. Integrated with SGLang and paired with a kernel scheduler, DeepFusionKernel ensures consistent accelerations across generation lengths, while remaining adaptable to diverse models, inference configurations, and hardware platforms.[1]

1 Introduction

The Transformer architecture (Vaswani et al., 2023) underpins modern large language models (LLMs) (Brown et al., 2020; OpenAI et al., 2024; DeepSeek-AI et al., 2025a), and as these models are deployed in agentic, real-world workloads, inference efficiency becomes a first-order concern (Aminabadi et al., 2022; Zhou et al., 2024; Yao et al., 2023). Agentic workloads require the model to handle very long contexts, maintain many persistent state items, and often produce long-form outputs (for example, large codebases). As a result, they process on the order of $100\times$ more tokens per inference than typical chatbot-style workloads (Wu et al., 2025). In this regime, caching and fast access to KV values shift the primary bottleneck from raw compute to memory capacity and memory bandwidth. That shift in turn changes the optimization landscape: permissible batch sizes shrink, GEMM inputs become "fat," Tensor Core utilization drops, and available compute is severely underused.

Most recent GPU optimizations focus on attention (Shah et al., 2024; Kwon et al., 2023; Zheng et al., 2024; Ye et al., 2025), but in autoregressive decoding, the SwiGLU MLP blocks (Shazeer, 2020; Grattafiori et al., 2024; Bai et al., 2023; DeepSeek-AI et al., 2025b) dominate parameter count and drive memory-bandwidth pressure. Although modern accelerators offer high HBM bandwidth (NVIDIA Corporation, 2021, 2024, 2025a), memory scaling has not kept pace with compute, so memory-bound kernels remain the limiting factor for throughput on many real deployments.

To close this gap, we propose **aggressive deep kernel fusion** to reduce memory traffic and improve utilization in FFN blocks in the attention mechanism. We implement a highly optimized fused operator, the **DeepFusionKernel**, which combines the separate GEMMs and pointwise kernels used in common four-kernel and two-kernel SwiGLU implementations (e.g., PyTorch (Paszke et al., 2019), SGLang (Zheng et al., 2024), and vLLM (Kwon et al., 2023)) into a single fused kernel that minimizes intermediate memory reads/writes and exposes more efficient work for Tensor Cores. Integrated with a lightweight, profile-driven kernel scheduler and deployed inside SGLang, DeepFusionKernel yields consistent, deployable speedups in bandwidth-bound agentic scenarios: up to 9.7% on A100 and 13.2% on H100 clusters compared to the SOTA SGLang implementation.

2 Background

Modern Transformer-based LLMs replace the original two-layer ReLU design in the MLP with the gating-integrated SwiGLU variant (Shazeer, 2020):

$$A_{\text{gate}} = XW_{\text{Gate}}, \quad A_1 = XW_{\text{Up}},$$
$$A_{\text{silu}} = \text{SiLU}(A_{\text{gate}}) = \text{Sigmoid}(A_{\text{gate}}) \otimes A_{\text{gate}},$$
$$A_2 = A_1 \otimes A_{\text{silu}}, \quad Y = A_2 W_{\text{Down}}.$$

Here $W_{\text{Up}}, W_{\text{Gate}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ and $W_{\text{Down}} \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$, with d_{ff} being 3.5 to 4 times of the

[1] Our code is available at https://github.com/ZixiBenZhang/deepfusionkernel.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 166–173
July 2-7, 2026 ©2026 Association for Computational Linguistics

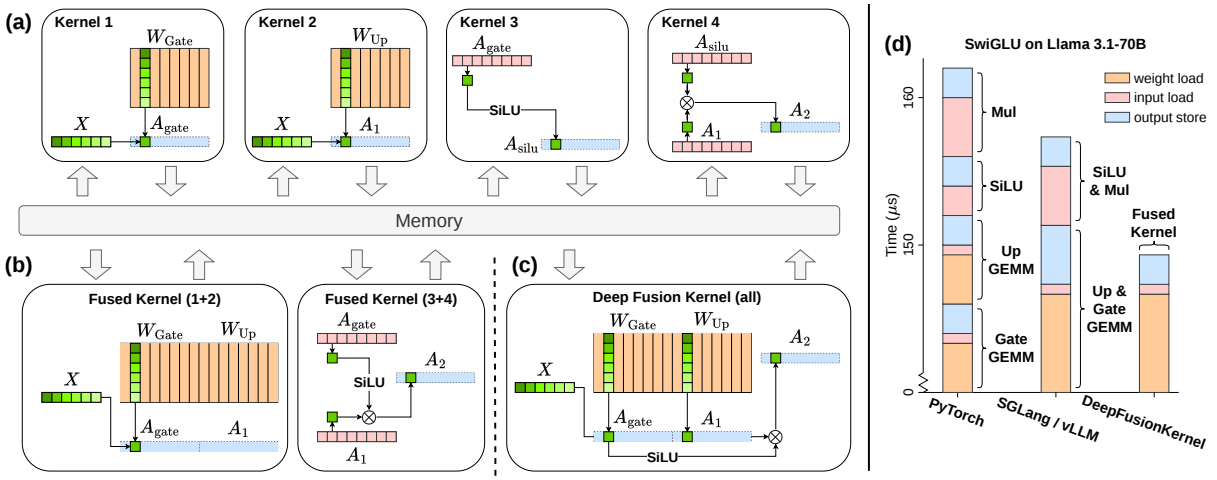

Figure 1: DeepFusionKernel leverages aggressive kernel fusion on SwiGLU blocks to eliminate intermediate activations and reduce memory traffic—without increasing FLOPs. Panels show implementation layouts: **(a)** naive PyTorch with four kernel launches; **(b)** the two-kernel design used by SGLang and vLLM; and **(c)** our single, deeply fused kernel that streams data through GEMMs and nonlinearities to avoid extra loads/stores. Speedup over PyTorch and SGLang / vLLM is displayed in **(d)**. By removing these redundant reads/writes, DeepFusionKernel yields up to 9.7% and 13.2% throughput improvements on A100 and H100 GPUs, respectively, on bandwidth-bound autoregressive decoding workloads.

d_{model}. These large matrices dominate the model's parameter count and memory footprint.

3 Method

Kernel fusion (Aminabadi et al., 2022; NVIDIA Corporation, 2025b; NVIDIA, 2018) reduces end-to-end latency and memory traffic by combining sequences of operations (GEMMs, pointwise nonlinearities, and simple elementwise math) into a single GPU kernel, thereby eliminating intermediate reads and writes—the dominant cost in bandwidth-bound LLM decoding (Kwon et al., 2023; Zheng et al., 2024). Fusion is most effective for elementwise and tile-local computations; true reductions (e.g., Softmax) introduce long-range dependencies that limit cross-SM streaming and therefore are not good fusion targets. Guided by this observation, we split the Transformer MLP into two stages:

$$A_2 = (XW_{\text{Up}}) \otimes \text{SiLU}(XW_{\text{Gate}}), \quad Y = A_2 W_{\text{Down}}.$$

and focus fusion effort on the first stage, which contains the GEMMs and pointwise gating that dominate memory traffic during autoregressive decoding.

As illustrated in Figure 1, DeepFusionKernel fuses the separate GEMMs and pointwise kernels (four launches in a naive PyTorch implementation, two in SGLang/vLLM) into a single deeply fused operator. The fused kernel streams intermediate values through the computation, avoiding materialization of large temporaries and drastically cutting reads/writes to HBM. To realize this in practice, we systematically explore loop ordering and tiling strategies that maximize on-chip reuse and arithmetic intensity. Concretely, we observe the following trade-offs in our experiments:

- **Row-major tiling:** improves locality for the input activation X, reducing repeated loads of input rows and benefiting scenarios with larger batch sizes or when activations dominate memory traffic.
- **Column-major tiling:** better reuses weight tiles, which is preferable when model weights dominate memory footprint (a common case in agentic decoding with small batch sizes and large models).

We integrate these tiling and loop-ordering choices into the fused kernel implementation and tune tile sizes to balance register usage, shared-memory occupancy, and Tensor Core utilization. Because the best kernel configuration depends on model shapes, batch size, GPU microarchitecture, and distributed interconnects, we accompany DeepFusionKernel with a lightweight profiler-driven scheduler. At deployment time, the scheduler quickly benchmarks the set of candidate kernels prior to the inference on the target hardware and selects

the highest-throughput option for the given model and workload, ensuring robust performance across architectures and real-world inference conditions.

4 Experiments

We integrate DeepFusionKernel into a complete inference pipeline and measure end-to-end decoding throughput in a realistic framework to isolate practical benefits and deployment considerations. To minimize kernel invocation and CPU overhead while maintaining a consistent evaluation environment, we run all end-to-end experiments inside the SGLang inference framework, with FlashInfer and CUDA Graphs enabled. We compare against three baselines: PyTorch (naive distributed), SGLang (default kernels), and vLLM. A lightweight kernel scheduler selects the best fused-kernel variant for each hardware/configuration before measurement.

4.1 Experiment Setup

Kernel performance depends on memory bandwidth, compute capability, model shape, and interconnect topology. To stress memory behavior, we run tensor-parallel inference across four NVIDIA A100 or H100 80 GB SXM GPUs (TP=4), which exposes both HBM access and inter-GPU communication effects. We first evaluate LLaMA 3.1 70B in FP16 with fixed prompt length 1 and target output lengths of 1024 tokens; batch sizes vary from 1 to 64. Each configuration is measured four times; we report mean throughput and standard deviation. For long-generation experiments (Section 4.3), output length is swept to probe KV-cache and attention effects.

4.2 Full-Model Throughput

Table 1 compares end-to-end decoding throughput across frameworks and GPUs. Consistent with prior work, SGLang substantially outperforms a naive distributed PyTorch baseline. Integrating DeepFusionKernel into SGLang yields additional gains—up to **9.7%** on A100 and **13.2%** on H100 for typical batch-size-limited decoding workloads—by reducing memory traffic in the SwiGLU MLPs.

The observed behavior follows expected bottleneck shifts: On A100, the benefit is largest at small batch sizes, where the workload is strongly memory-bandwidth-bound, and reduced reads/writes directly raise throughput. On H100, the kernel maintains an advantage across a wider batch-size range; H100's much higher compute throughput (1979 TFLOPs/s versus 312 TFLOPs/s on A100) means memory bandwidth continues to be a salient limiter, so memory-traffic reductions from fusion still translate to measurable speedups, though with diminishing marginal returns at very large batches.

Throughput variance grows with batch size across all setups, driven primarily by jitter in inter-GPU communication. Because DeepFusionKernel reuses SGLang's existing all-reduce and collective primitives, fusion does not change communication patterns and thus inherits this variance.

4.3 Agentic Long-Generation Evaluation

Agentic workloads generate very long outputs and accumulate large KV caches, increasing off-chip memory pressure and per-token latency. To evaluate this regime, we benchmark LLaMA 3.1 70B for output lengths from 1024 up to 16384 tokens, using three representative batch sizes: $B = 1$ (single-query), $B = 4$ (light concurrent inference), and $B = 16$ (moderate concurrency). Tests are run in FP16 and under TP=4 on both A100 and H100 clusters. For consistency, we measure only the decoding stage.

Results in Table 2 show DeepFusionKernel consistently improves throughput versus SGLang and vLLM across generation lengths and concurrency levels. Speedup variability is mostly explained by communication jitter and occasional system-level noise at high concurrency; nevertheless, the MLP remains a significant fraction of per-token latency even for long sequences, so memory-traffic reduction from fusion yields persistent gains.

4.4 Discussion

Our experimental results demonstrate that DeepFusionKernel with a profiler-driven scheduler consistently improves decoding throughput across realistic, bandwidth-bound inference scenarios. Key observations are:

- **Robust gains in memory-bound regimes.** Fusion produces the largest improvements when workloads are memory-bandwidth-limited (small batches, large models, long contexts). Even on H100, where compute is abundant, reducing memory traffic yields meaningful speedups.
- **Stable behavior for long-context generation.** As sequence length and KV cache grow, attention cost increases, but the MLP still con-

Table 1: Decoding throughput and standard deviation of Llama 3.1 70B with 1024 output length, using Torch, SGLang, and DeepFusionKernel. Throughput speedup rates of DeepFusionKernel against SGLang are presented.

Batch size	A100 GPU cluster				H100 GPU cluster			
	Torch Throughput	SGLang Throughput	DeepFusionKernel		Torch Throughput	SGLang Throughput	DeepFusionKernel	
			Throughput	Speedup			Throughput	Speedup
1	2.97±0.12	35.64±0.04	37.66±0.04	+5.7%	3.55±1.08	60.2±0.5	62.22±0.14	+3.4%
2	5.65±0.41	67.43±2.64	72.6±0.04	+7.7%	8.39±1.4	107.05±18.21	121.13±0.57	+13.2%
4	11.83±0.37	134.02±0.06	140.55±0.13	+4.9%	18.11±1.95	228.9±0.24	237.05±1.05	+3.6%
8	22.98±1.67	255.29±0.09	270.74±0.46	+6.1%	34.30±4.31	451.64±0.72	470.63±0.33	+4.2%
16	40.57±15.69	453.75±1.42	497.69±4.04	+9.7%	52.37±15.86	878.08±16.33	914.62±0.81	+4.2%
32	86.43±17.27	815.37±0.4	847.42±2.55	+3.9%	118.56±13.25	1551.33±11.67	1635.94±11.39	+5.5%
64	169.33±33.11	1391.76±1.97	1410.39±37.06	+1.3%	129.81±42.25	3016.4±9.74	3119.55±1.91	+3.4%

Table 2: Throughputs and standard deviation across batch sizes and output lengths using SGLang, vLLM, and DeepFusionKernel with kernel scheduler. Best throughputs are marked in bold.

GPU		A100			H100		
Batch size	Output len	1024	4096	16384	1024	4096	16384
1	SGLang	35.68±0.01	35.53±0.01	34.97±0.01	60.20±0.50	60.77±0.46	60.18±0.07
	vLLM	33.47±0.21	33.64±0.05	32.93±0.51	58.37±0.12	58.70±0.05	57.98±0.12
	DeepFusionKernel	**37.57±0.10**	**37.50±0.01**	**36.79±0.05**	**61.22±0.59**	**61.08±0.31**	**60.72±0.81**
4	SGLang	135.76±0.11	132.66±0.44	126.81±0.16	228.90±0.24	219.48±3.74	213.94±1.03
	vLLM	126.10±0.93	125.10±0.08	119.30±0.63	227.69±1.18	227.70±0.18	218.73±0.12
	DeepFusionKernel	**140.15±0.11**	**138.16±0.02**	**131.75±0.14**	**230.98±1.79**	**234.61±3.03**	**227.73±0.04**
16	SGLang	453.75±1.42	452.60±0.31	396.70±0.16	878.08±16.33	815.40±1.57	732.18±5.07
	vLLM	462.01±1.56	448.93±0.43	390.94±0.63	865.37±3.98	850.33±0.47	746.42±0.47
	DeepFusionKernel	**493.60±2.75**	**478.66±2.47**	**416.19±0.79**	**894.32±0.39**	**863.40±0.13**	**764.57±8.82**

tributes substantially to per-token cost; fusion therefore remains beneficial across long-generation workloads.

- **Low deployment overhead.** Kernel selection is performed with a short pre-inference profiling step; after selection, we use CUDA Graphs for capture, so there is no recurring inference-time dispatch overhead from the scheduler.

Overall, DeepFusionKernel provides a practical, deployable improvement to existing inference stacks: it reduces memory traffic in the critical SwiGLU path, integrates with current frameworks, and produces repeatable throughput gains across hardware and workload regimes.

5 Related work

Existing frameworks like Apex (NVIDIA, 2018), TensorRT-LLM (NVIDIA Corporation, 2025b), and DeepSpeed-MII (Microsoft, 2022) use shallow fusion (e.g., GEMM+activation), leaving significant overheads for larger MLPs. Automatic compile-time fusion approaches exist: Welder (Shi et al., 2023) fuses based on tile-graph cost models but is limited to linear chains; TVM (Chen et al., 2018) applies pattern matching and heuristics but its template-driven method mostly handles small trees; Blockbuster (Dekel, 2025) uses algebraic rules and demonstrated a SwiGLU prototype, but remains a standalone compiler study and lacks runtime feedback and hardware-aware tuning. In contrast, DeepFusionKernel deeply fuses the full SwiGLU into a single kernel. Paired with the kernel scheduler, it adapts fusion depth to workload and GPU, delivering consistent, branch-free speedups.

6 Conclusion

We present **DeepFusionKernel**, an aggressively fused CUDA operator that eliminates intermediate buffers in the SwiGLU MLP and rebalances the trade-off between memory traffic and on-chip compute. When integrated into SGLang and driven by a lightweight profiler-based scheduler, the fused kernel produces consistent, deployable throughput improvements—up to 9.7% on A100 and 13.2% on H100—across batch sizes and long-generation agentic workloads. By targeting the

memory-bandwidth bottleneck that dominates autoregressive decoding, DeepFusionKernel lets modern GPUs better realize their available compute, making it a practical optimization for real-world LLM inference pipelines.

Limitations

Due to limited computing resources, we do not exhaustively evaluate different GPU cluster interconnects, though our tests indicate that any resulting performance degradation is minimal for the workloads studied. Similarly, we do not quantify performance variation from inter-GPU communication, but its impact is mitigated by using the same inter-device reduction strategy as the baseline framework.

References

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed inference: Enabling efficient inference of transformer models at unprecedented scale. *Preprint*, arXiv:2207.00032.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Meghan Cowan, Haichen Shen, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An automated end-to-end optimizing compiler for deep learning. *Preprint*, arXiv:1802.04799.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 181 others. 2025a. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025b. DeepSeek-V3 technical report. *Preprint*, arXiv:2412.19437.

Ofer Dekel. 2025. Blockbuster, part 1: Block-level AI operator fusion. *Preprint*, arXiv:2505.07829.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Shengyu Liu Jiashi Li. 2025. FlashMLA: Efficient MLA decoding kernels. https://github.com/deepseek-ai/FlashMLA.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with PagedAttention. *Preprint*, arXiv:2309.06180.

Microsoft. 2022. DeepSpeed Model Implementations for Inference (MII). https://github.com/deepspeedai/DeepSpeed-MII. Accessed: 2025-05-26.

NVIDIA. 2018. Apex. https://github.com/nvidia/apex.

NVIDIA Corporation. 2021. NVIDIA A100 Tensor Core GPU datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf. Accessed: 2025-05-26.

NVIDIA Corporation. 2024. NVIDIA H100 Tensor Core GPU datasheet. https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306. Accessed: 2025-05-26.

NVIDIA Corporation. 2025a. NVIDIA DGX B200 datasheet. https://resources.nvidia.com/en-us-dgx-systems/dgx-b200-datasheet. Accessed: 2025-05-26.

NVIDIA Corporation. 2025b. NVIDIA TensorRT-LLM documentation. https://nvidia.github.io/TensorRT-LLM/. Accessed: 2025-05-26.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei,

Allison Tam, and 244 others. 2024. OpenAI o1 system card. *Preprint*, arXiv:2412.16720.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. PyTorch: An imperative style, high-performance deep learning library. *Preprint*, arXiv:1912.01703.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. *Preprint*, arXiv:2407.08608.

Noam Shazeer. 2020. GLU variants improve transformer. *Preprint*, arXiv:2002.05202.

Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. 2023. Welder: Scheduling deep learning memory access via tile-graph. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 701–718, Boston, MA. USENIX Association.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Haoran Wu, Can Xiao, Jiayi Nie, Xuan Guo, Binglei Lou, Jeffrey T. H. Wong, Zhiwen Mo, Cheng Zhang, Przemyslaw Forys, Wayne Luk, Hongxiang Fan, Jianyi Cheng, Timothy M. Jones, Rika Antonova, Robert Mullins, and Aaron Zhao. 2025. Combating the memory walls: Optimization pathways for long-context agentic llm inference. *Preprint*, arXiv:2509.09505.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. FlashInfer: Efficient and customizable attention engine for LLM inference serving. *Preprint*, arXiv:2501.01005.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient execution of structured language model programs. *Preprint*, arXiv:2312.07104.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *Preprint*, arXiv:2404.14294.

A Technical Appendices and Supplementary Material

A.1 Distributed Inference

In distributed inference, computation is spread across multiple GPUs, either within a single server or across multiple server nodes. This setup introduces communication overheads, primarily due to all-reduce (for aggregating results) and all-gather (for collecting distributed outputs) operations. The frequency and cost of these operations are influenced by the kernel fusion strategy employed. The magnitude of the overheads generally scales with the volume of data transferred and can vary logarithmically, linearly, or quadratically with the number of devices or nodes, depending on the interconnect architecture.

A common strategy for distributed computation in large language models is **tensor parallelism (TP)**. For a matrix multiplication $Y = XW$, where $X \in \mathbb{R}^{M\times N}$, $W \in \mathbb{R}^{N\times K}$, and $Y \in \mathbb{R}^{M\times K}$, TP involves partitioning the weight matrix W across devices, either row-wise (producing tall matrices) or column-wise (producing wide matrices). Each GPU computes a slice of the output Y, after which an all-gather operation assembles the full result.

However, in the case of compound matrix operations like SwiGLU MLP—which comprises two consecutive matrix multiplications with an intermediate nonlinearity—we can reduce communication overhead. Specifically, as demonstrated in Figure 2, we partition W_{Up} and W_{Gate} column-wise and W_{Down} row-wise. This configuration allows intermediate computations to remain local to each device, requiring only a single all-reduce at the end to aggregate the final output. Consequently, we reduce the number of collective operations to just one all-reduce per SwiGLU MLP block.

A.2 Full-Model Performance Evaluation Experiment Setup

SGLang (Zheng et al., 2024) is a high-performance inference framework for large language models (LLMs), demonstrating up to $6.4\times$ higher throughput than other state-of-the-art systems, including

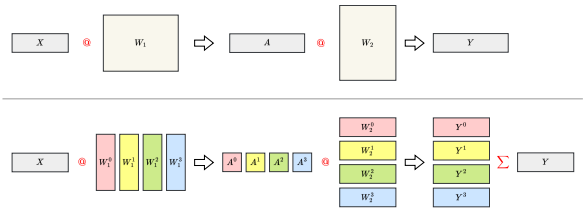

Figure 2: A matrix splitting scheme of consecutive matrix multiplications under tensor parallelism (TP) that contains a single all-reduce operation, denoted as $\sum$.

vLLM (Kwon et al., 2023) and LMQL (Beurer-Kellner et al., 2023).

SGLang achieves this through a suite of custom optimizations tailored for efficient single- and multi-GPU inference, including:

- **RadixAttention** for efficient key-value (KV) cache management;
- **Grouped GEMMs** for kernel fusion across parallel matrix multiplications;
- **Fused kernels** for common patterns of elementwise operations;
- **Optimized scheduling** for consecutive matrix multiplications under tensor parallelism (TP);
- **Custom cross-device all-reduce kernels**;
- **TP-adapted model head**, combining a linear projection and a Softmax for next-token prediction.

SGLang supports several SOTA attention backends, including Triton (Tillet et al., 2019), FlashAttention-3 (Shah et al., 2024), FlashMLA (Jiashi Li, 2025), and FlashInfer (Ye et al., 2025). Additionally, SGLang captures CUDA Graphs during a warm-up phase, allowing inference to proceed without per-iteration PyTorch API or kernel launch overheads.

As kernel performance depends on both memory bandwidth and compute capacity, we evaluate DeepFusionKernel inference across multiple SOTA GPUs. Due to the large model size, we use TP to partition weight matrices across GPUs. TP does not increase the number of FLOPs or alter arithmetic intensity, assuming the fused computation strategy described in Appendix A.1 is followed.

However, TP introduces extra HBM access and communication latency during the all-reduce step that follows the TP-adapted W_{Down} projection. The impact of this latency grows with the amount of data to be synchronized and is sensitive to both the number of GPUs and their interconnect configuration. For example, in an NVIDIA A100 SXM cluster, each pair of GPUs is connected via NVIDIA NVLink (600GB/s), while the connections to the host system and between more GPUs employ the standard PCIe Gen4 links (64 GB/s) (NVIDIA Corporation, 2021). Inter-device communication is also susceptible to packet drops, which introduces latency variability.

For this evaluation, we use SGLang with the FlashInfer attention backend (Ye et al., 2025) and CUDA Graphs enabled so that once profiling identifies the best kernel, SGLang captures a complete CUDA Graph, eliminating branching during inference. We compare three setups:

- Torch distributed inference
- SGLang with default kernels
- SGLang with DeepFusionKernel

We run Llama 3.1 70B in FP16 with TP across four NVIDIA A100 80GB SXM GPUs, all on a single node. To focus on decoding performance, we use an input sequence length of 1 and an output sequence length of 1024. Batch sizes range from $B = 1$ to $B = 64$. Each experiment is repeated four times, and we report the mean and standard deviation of decoding throughput.

A.3 Additional Experiment Results

A.3.1 Comparison with Torch Inductor

We evaluated `torch.compile` using the Inductor backend on four H100 GPUs. As shown in Table 3, Torch Inductor is uncompetitive for high-performance LLM inference in this regime. It lags behind our kernel by orders of magnitude, achieving up to 126× lower throughput at small batch sizes, and suffers from *Out-Of-Memory (OOM)* errors as sequence lengths increase, making it unsuitable for long-context agentic workloads.

A.3.2 Sensitivity Analysis of Tile Sizes

We utilize Triton's JIT autotuner to perform an exhaustive search over the tile dimensions $(d_B, d_{\text{model}}, d_{\text{ff}})$ using candidates from $\{32, 64, 128, 256\}$. The autotuner optimizes for minimum latency, effectively balancing hardware constraints. To provide insight into this search space, Figure 3 illustrates a tile-size sweep for a single DeepFusionKernel in Llama-3.1-70B (TP=4, batch size 1) on A100 SXM GPUs. We observe

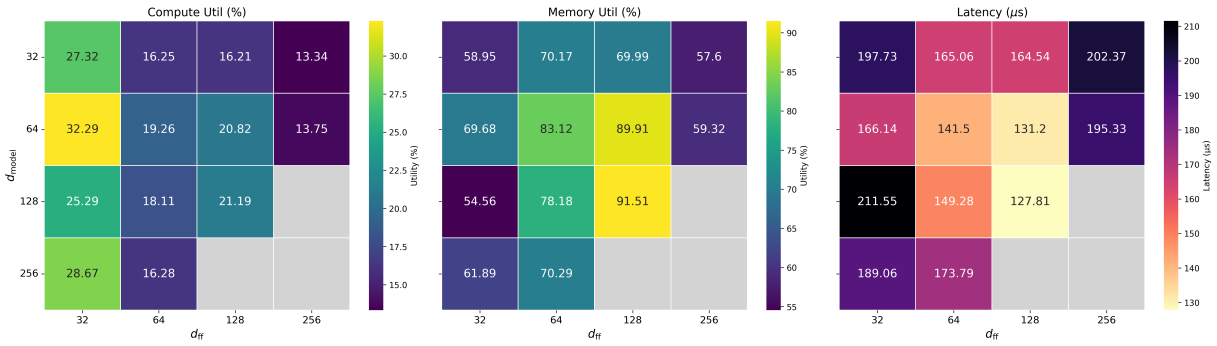

Figure 3: Sensitivity of DeepFusionKernel performance to model hidden dimensions (d_{model}) and MLP intermediate dimensions (d_{ff}). Metrics include compute utilization, memory bandwidth utilization, and latency (BS=1, TP=4, 4×A100). Grey entries denote shared memory capacity exhaustion (*Cache OOM*).

Table 3: Throughput comparison across PyTorch Eager mode, `torch.compile` (Inductor backend), and DeepFusionKernel. Benchmarked on 4×H100 GPUs. DeepFusionKernel consistently achieves higher throughput (tokens/s) by mitigating memory-bound bottlenecks.

Batch size	torch (eager)	torch (Inductor)	DeepFusionKernel
1	3.55 ± 1.08	0.49 ± 0.004	62.22 ± 0.14
2	8.39 ± 1.40	0.96 ± 0.002	121.13 ± 0.57
4	18.11 ± 1.95	38.48 ± 0.30	237.05 ± 1.05
8	34.30 ± 4.31	66.33 ± 0.99	470.63 ± 0.33
16	52.37 ± 15.86	122.31 ± 2.98	914.62 ± 0.81
32	118.56 ± 13.25	194.31 ± 1.15	1635.94 ± 11.39
64	129.81 ± 42.25	429.64 ± 11.66	3119.55 ± 1.91

three primary behaviors:

1. **Shared Memory Limits:** Larger tile sizes improve data reuse but increase the shared memory (SRAM) footprint. Configurations that exceed hardware's SRAM capacity (e.g. 128×256) result in cache OOM errors, defining the hard upper bound of our search space.
2. **Register Pressure and Occupancy:** Conversely, very small tile sizes (e.g. 32×32) reduce register pressure but fail to saturate the GPU's memory controllers, leading to excessive memory access overhead.
3. **Optimal Balance:** The autotuner converges on a $(128, 128)$ configuration. This setup maximizes Memory Bandwidth Utilization (91.51%), the critical bottleneck in agentic decoding, achieving a minimum latency of 127.81 μs by pushing the hardware to its "shared memory wall."

A.3.3 Single MLP Micro-benchmark

To isolate the source of our end-to-end performance gains, we profiled the latency of individual MLP blocks across SGLang, vLLM, and our DeepFusionKernel. Using a Llama-3.1-70B model on four A100 GPUs (batch size 16), Table 4 shows that our method achieves a latency reduction of 5.89% and 5.53% per block over SGLang and vLLM, respectively. While these percentages may appear modest at the kernel level, they compound across the 80 layers of the Llama-3.1-70B architecture. These per-block efficiencies directly validate the significant end-to-end throughput improvements reported in our main evaluation.

Table 4: Latency comparison for single MLP block across SGLang, vLLM, and DeepFusionKernel. Data are reported as the median ± half interquartile range.

Method	SGLang	vLLM	DeepFusionKernel
Latency (μs)	249.78 ± 4.41	248.84 ± 9.19	$\mathbf{235.07} \pm 5.80$

A.4 Licenses of Models and Frameworks

- Llama 3.1 70B (Grattafiori et al., 2024): Llama 3.1 Community License Agreement. Link: https://huggingface.co/meta-llama/Llama-3.1-70B and https://huggingface.co/meta-llama/Llama-3.1-405B.
- PyTorch 2.6.0 (Paszke et al., 2019): Please refer to the official GitHub page: https://github.com/pytorch/pytorch/tree/v2.6.0.
- SGLang v0.4.6.post4 (Zheng et al., 2024): Apache-2.0 license. GitHub page link: https://github.com/sgl-project/sglang/tree/v0.4.6.post4.
- vLLM v0.10.1.1 (Kwon et al., 2023): Apache-2.0 license. Github page link: https://github.com/vllm-project/vllm/tree/v0.10.1.1.