ClusterFusion: Expanding Operator Fusion Scope for LLM Inference via Cluster-Level Collective Primitive

Xinhao Luo[1,2] **Zihan Liu**[1,2*] **Yangjie Zhou**[3*] **Shihan Fang**[1] **Ziyu Huang**[1,2] **Yu Feng**[1,2]
Chen Zhang[1] **Shixuan Sun**[1] **Zhenzhe Zheng**[1] **Jingwen Leng**[1,2] **Minyi Guo**[1,2]

[1] Shanghai Jiao Tong University [2] Shanghai Qi Zhi Institute [3] National University of Singapore

{lxh666, altair.liu, fang-account, huang_ziyu}@sjtu.edu.cn
{y-feng, chenzhang.sjtu, sunshixuan, zhengzhenzhe}@sjtu.edu.cn
yj_zhou@nus.edu.sg, leng-jw@cs.sjtu.edu.cn, guo-my@cs.sjtu.edu.cn

Abstract

Large language model (LLM) decoding suffers from high latency due to fragmented execution across operators and heavy reliance on off-chip memory for data exchange and reduction. This execution model limits opportunities for fusion and incurs significant memory traffic and kernel launch overhead. While modern architectures such as NVIDIA Hopper provide distributed shared memory and low-latency intra-cluster interconnects, they expose only low-level data movement instructions, lacking structured abstractions for collective on-chip communication. To bridge this software-hardware gap, we introduce two cluster-level communication primitives, `ClusterReduce` and `ClusterGather`, which abstract common communication patterns and enable structured, high-speed data exchange and reduction between thread blocks within a cluster, allowing intermediate results to be on-chip without involving off-chip memory. Building on these abstractions, we design `ClusterFusion`, an execution framework that schedules communication and computation jointly to expand operator fusion scope by composing decoding stages such as *QKV Projection*, *Attention*, and *Output Projection* into a single fused kernels. Evaluations on H100 GPUs show that `ClusterFusion` outperforms state-of-the-art inference frameworks by $1.61\times$ on average in end-to-end latency across different models and configurations. The source code is available at `https://github.com/xinhao-luo/ClusterFusion`.

1 Introduction

Large language models (LLMs) have become a cornerstone of modern artificial intelligence systems. Their applications span natural language processing [49, 25], code generation [17, 42], and mathematical reasoning [45, 54]. LLM inference typically involves two stages: a prefilling phase that encodes the input prompt and a decoding phase that generates output tokens auto-regressively. As sequence length increase and model sizes grow, the decoding phase dominates overall inference latency, making it the primary bottleneck in real-time LLM applications.

To accelerate decoding, recent research has explored various optimizations [10, 19, 26, 29, 15, 16, 13, 14, 26]. A growing body of research focuses on optimizing execution dataflow [10, 19, 55, 22], which refers to the organization of computation and communication across the parallel and memory hierarchy of modern GPUs [34]. Thread blocks serve as the fundamental execution units, each responsible for processing a portion of the data and typically assigned to different hardware unit such

*Corresponding authors.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

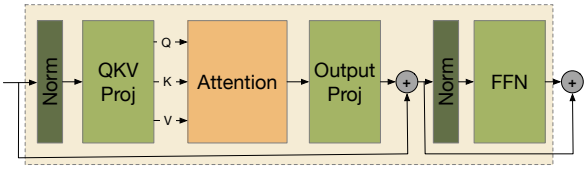

Figure 1: Typical Transformer Block as the fundamental component of the modern LLMs.

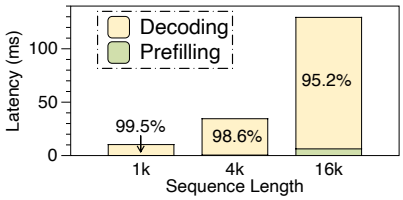

Figure 2: Latency comparison of prefilling and decoding for 256 tokens.

as streaming multiprocessor (SM). However, most existing systems treat thread blocks as independent execution units. Inter-block dependencies are resolved by materializing intermediate results to global memory, resulting in frequent off-chip transfers, redundant synchronization, and limited operator fusion scope [63, 53].

Recent GPU architectures offer new opportunities to address these limitations. NVIDIA Hopper introduces thread block clusters and distributed shared memory (DSMEM), which enable direct on-chip communication among blocks within the same cluster [33]. Despite this potential, two major challenges remain. First, current architectures expose only low-level data movement instructions without providing high-level structured communication abstractions. Second, as our analysis in Sec. 2.3 shows, DSMEM performance is highly sensitive to cluster configuration. These factors make it challenging to integrate DSMEM into real-world LLM systems.

To address these limitations, we propose two cluster-level collective primitives: `ClusterReduce` and `ClusterGather`, which abstract common communication patterns such as reduction and aggregation. These primitives enable structured intra-cluster coordination, allowing intermediate results to be shared and combined on-chip without global memory access. Built upon these primitives, our key insight is to treat each thread block cluster as a fundamental parallel unit, using cluster-level collective communication primitives to resolve inter-block dependencies efficiently. Guided by the key insight, we propose the cluster-centric dataflow and develop `ClusterFusion`, an execution framework that jointly schedules computation and communication to expand operator fusion scope.

Evaluation on NVIDIA H100 GPUs shows that `ClusterFusion` achieves $1.61\times$ speedup on average in end-to-end latency compared to state-of-the-art frameworks. These performance gains hold across diverse model architectures (e.g., Llama [51], DeepSeek [11]) and configurations, demonstrating the generality and effectiveness of our approach.

In summary, we have made the following contributions:

- We analyze the communication patterns and fusion scope in existing LLM decoding workflows, identifying fragmented kernel execution and off-chip synchronization as key barriers to efficient decoding fusion. We further profile the DSMEM mechanism on NVIDIA Hopper GPUs, revealing its potential to support low-latency inter-block communication and reduce off-chip memory dependency.
- We propose two cluster-level collective primitives, `ClusterReduce` and `ClusterGather`, to support structured inter-block collective communication. These primitives abstract reduction and aggregation over DSMEM and enable efficient coordination between thread blocks.
- We develop `ClusterFusion`, an execution framework that expands operator fusion via our proposed primitives. `ClusterFusion` integrates structured intra-cluster communication into the cluster-centric dataflow, fusing *QKV Projection*, *Attention* and *Output Projection*. This enables coordinated computation and communication without off-chip memory traffic and outperforms SOTA frameworks.

2 Background

2.1 LLM Inference Workflow and Bottlenecks

LLMs are commonly built on Transformer-based decoder-only architectures [52]. As shown in Fig. 1, each Transformer Block comprises *QKV Projection*, *Attention*, *Output Projection* and a feed-forward network (FFN). For an input sequence X, each *Attention* head computes projections $Q_i = XW_{Q_i},\ K_i = XW_{K_i},\ V_i = XW_{V_i}$, followed by scaled dot-product *Attention* and *Output*

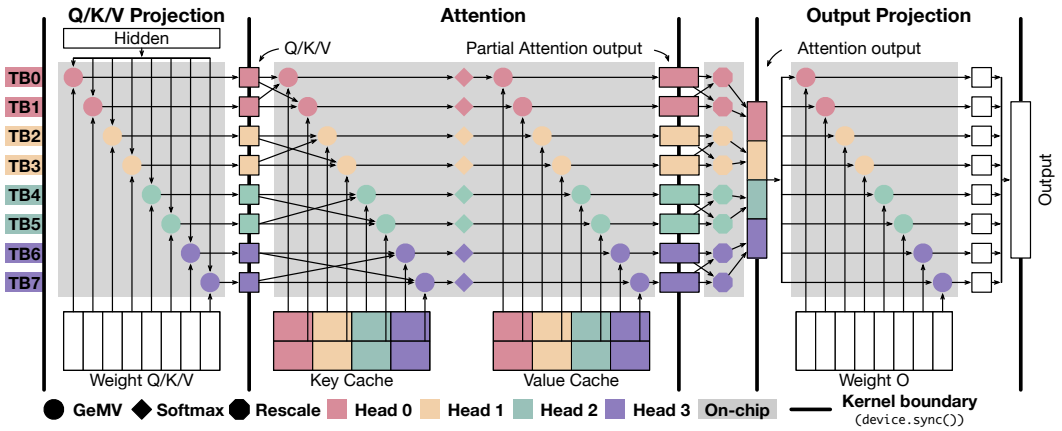

Figure 3: Existing *QKV Projection*, *Attention*, and *Output Projection* dataflow graph.

Projection:

$$Z = \text{Concat}\left(\text{Softmax}\left(\frac{Q_1 K_1^\top}{\sqrt{d_k}}\right) V_1, \ldots, \text{Softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}}\right) V_h\right) W_O \tag{1}$$

Here, d_k is the dimension of each head's query (Q) and key (K). The FFN module applies three linear layers with a non-linear activation in between, formulated as:

$$\text{FFN}(Z) = W_3 \left(\sigma(W_1 Z) \odot W_2 Z\right) \tag{2}$$

where σ is a non-linear activation (e.g., GELU), and $\odot$ denotes element-wise multiplication.

During inference, the model proceeds in two distinct stages: prefilling and decoding. In the decoding stage, the model generates tokens one at a time in an auto-regressive manner, reusing the KV cache while appending new entries. This sequential decoding pattern inherently limits parallelism and dominates inference latency at longer context lengths. In Fig. 2, the percentages represent the proportion of latency on decoding stage for different sequence lengths. Decoding accounting for over 95% of the total latency for a 256 tokens sequence generation, as measured using SGLang [58], a state-of-the-art LLM serving framework. This makes decoding the dominant computational bottleneck during inference and a natural target for system-level performance optimization.

2.2 Existing Communication Patterns and Fusion Scope

Numerous studies have proposed various techniques [9, 8, 44, 10, 19, 55, 57] to improve the Transformer performance. Among them, increasing attention has been given to the design of execution dataflow, which refers to the structured organization of computation stages and their associated data movement, including partitioning, scheduling, and communication, over the parallel execution model and memory hierarchy [34]. In decoding workloads, the dataflow plays a central role in determining end-to-end latency by governing operator scheduling and intermediate data exchange.

In existing GPU-based decoding dataflows, thread blocks commonly serve as the fundamental unit of parallel execution and scheduling. Fig. 3 illustrates the dataflow of Llama2-7B [51] decoding phase in SGLang [58], covering three stages: *QKV Projection*, *Attention*, and *Output Projection*. Within each kernel, thread blocks are assigned to individual *Attention* heads and operate on disjoint tiles of the hidden dimension and KV sequence. In the *QKV Projection* stage, each thread block performs a linear transformation on its assigned hidden states to produce local Q, K, and V vectors. These outputs are written to global memory. The *Attention* stage is implemented with FlashDecoding[10, 19, 55], where each block computes a partial *Attention* result using its corresponding Q and a segmented KV cache first. A separate rescaling kernel then aggregates these partial results across blocks. The final *Output Projection* is executed block-wise on the aggregated *Attention* output. Existing decoding dataflows exhibit two forms of inter-block communication: exchanging intermediate data (e.g., Q/K/V vectors) needed by multiple blocks, and reducing partial results across blocks (e.g., Attention output). These dependencies are resolved via off-chip memory and explicit kernel boundaries, leading to global synchronization barriers and hindering operator fusion [63, 53].

This block-isolated execution structure introduces launch overhead, off-chip memory round-trips, and global synchronization barriers between kernels. Due to the absence of structured communication across thread blocks, intermediate results must be materialized to global memory and reloaded by subsequent kernels, limiting opportunities for effective on-chip data reuse and broader fusion. To overcome these limitations, we require an execution mechanism that enables collective scheduling and communication across thread blocks.

2.3 Cluster-Level Opportunities and Challenges

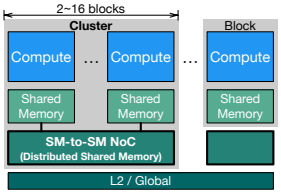

Figure 4: NVIDIA Hopper architecture.

Figure 5: SM-to-SM access latency (left), bandwidth (middle), and number of active SMs (right) for varying cluster size on H100 GPU.

The previous section shows that the mainstream methods resolve inter-block data dependencies via global memory synchronization. However, recent architectures, such as NVIDIA Hopper [33] shown in Fig. 4, can group a set of thread blocks into a cluster. Within each cluster, thread blocks can share data directly through a high-speed SM-to-SM Network-on-Chip (NOC), also referred to as DSMEM, thereby avoiding costly global memory accesses and enabling efficient intra-cluster communication.

To understand the opportunities and challenges of this mechanism, we profile DSMEM on an NVIDIA H100 GPU by varying the cluster size from 1 to 16, which is the maximum supported by Hopper hardware. As shown in Fig. 5 left, SM-to-SM access latency improves significantly with small cluster sizes. When the cluster size is 2, the average latency reaches 190 cycles, which is substantially lower than global memory latency (exceeding 470 cycles). This demonstrates the potential of DSMEM for low-latency on-chip communication.

However, this benefit comes with notable trade-offs. As the cluster size increases, the available communication bandwidth decreases, slightly lagging behind the global memory bandwidth when the cluster size reaches 16 (2.90 TB/s vs. 2.96 TB/s) due to crossbar architecture [23, 3]. Additionally, the number of active SMs is reduced due to hardware constraints. These effects limit the scalability of cluster-based execution and necessitate careful configuration to balance communication efficiency and overall parallelism.

Beyond hardware-level trade-offs, there are software-level challenges as well. NVIDIA currently exposes DSMEM and thread block cluster functionality only through low-level PTX instructions, which only support basic, peer-to-peer data movement between thread blocks [33]. This low-level interface presents significant challenges for expressing reusable synchronization and communication patterns in practical scenarios, which results in a steep programming barrier and leaves developers without clear guidance on how to effectively apply these capabilities.

3 ClusterFusion: Execution Framework with Cluster-Centric Dataflow

This section presents the design of `ClusterFusion`. We begin by introducing cluster-level collective primitives that enable structured data reduction and aggregation across thread blocks. These primitives form the foundation for the cluster-centric dataflow that fuses *QKV Projection*, *Attention* and *Output Projection*. `ClusterFusion` builds on this to achieve high-performance LLM decoding with expanded operator fusion scope.

3.1 Cluster-Level Collective Primitives

To explore hardware-level trade-offs and overcome software-level challenge, we introduce two cluster-level collective primitives that abstract common communication patterns. The key insight is to treat a thread block cluster as a fully connected logical network, where blocks participate in structured collective operations. Inspired by communication primitives in distributed systems [48, 4], we design

Algorithm 1 `ClusterReduce` over DSMEM - Thread Block view

Require: A cluster of $N = 2^k$ thread blocks ($k \leq 4$), each with rank $b \in [0, N-1]$ and a shared memory buffer $\mathbf{D}_b$ containing local data that needs to be reduced together with data from other thread blocks in the same cluster. A reduction operator $\oplus$ (e.g., sum or max).

Allocate shared memory buffer $\mathbf{B}_b$ with the same size of $\mathbf{D}_b$ ▷ Used as a temporary buffer.
$\texttt{stride} \leftarrow 1$.
while $\texttt{stride} < N$ **do**
    block rank $\texttt{send\_to} \leftarrow (b + \texttt{stride}) \bmod N$.
    block rank $\texttt{recv\_from} \leftarrow (b - \texttt{stride} + N) \bmod N$.
    Send $\mathbf{D}_b$ to $\mathbf{B}_{\texttt{send\_to}}$ of block `send_to` via DSMEM.
    Receive $\mathbf{D}_{\texttt{recv\_from}}$ from block `recv_from` into $\mathbf{B}_b$ via DSMEM.
    Wait for the arrival of $\mathbf{D}_{\texttt{recv\_from}}$.
    $\mathbf{D}_b \leftarrow \mathbf{D}_b \oplus \mathbf{B}_b$ ▷ Aggregate partial result using a reduction operator.
    $\texttt{stride} \leftarrow \texttt{stride} \times 2$ ▷ Exponential stride progression.
end while
Return $\mathbf{D}_b$.

Algorithm 2 `ClusterGather` over DSMEM - Thread Block view

Require: A cluster of $N = 2^k$ thread blocks ($k \leq 4$), each with rank $b \in [0, N-1]$ and a shared memory buffer $\mathbf{D}_b$ of size $N \times \texttt{size}$. The first segment $\mathbf{D}_b[0 : \texttt{size}]$ contains the local data that needs to be gathered together with data from other thread blocks in the same cluster.

$\texttt{stride} \leftarrow 1$.
while $\texttt{stride} < N$ **do**
    block rank $\texttt{send\_to} \leftarrow (b + \texttt{stride}) \bmod N$.
    block rank $\texttt{recv\_from} \leftarrow (b - \texttt{stride} + N) \bmod N$.
    Send $\mathbf{D}_b[0 : \texttt{size} \times \texttt{stride}]$ to $\mathbf{D}_{\texttt{send\_to}}[\texttt{stride} \times \texttt{size} : 2 \times \texttt{stride} \times \texttt{size}]$ of block `send_to` via DSMEM.
    Receive $\mathbf{D}_{\texttt{recv\_from}}[0 : \texttt{size} \times \texttt{stride}]$ from block `recv_from` into $\mathbf{D}_b[\texttt{stride} \times \texttt{size} : 2 \times \texttt{stride} \times \texttt{size}]$ via DSMEM.
    Wait for the arrival of $\mathbf{D}_{\texttt{recv\_from}}[0 : \texttt{size} \times \texttt{stride}]$.
    $\texttt{stride} \leftarrow \texttt{stride} \times 2$ ▷ Exponential stride progression
end while
Return $\mathbf{D}_b$.

two cluster-level primitives: `ClusterReduce`, which reduces data across blocks using associative operators such as sum or max, and `ClusterGather`, which replicates local data from each block to all others for data sharing as shown in Fig. 6.

As shown in Alg. 1 and Alg. 2, both `ClusterReduce` and `ClusterGather` adopt a binary-tree pattern across $log_2 N$ rounds, where N is the cluster size. In each round, the communication stride doubles, and each block exchanges data with a peer whose index is offset by the current stride. `ClusterReduce` performs element-wise reductions while keeping the message size constant, whereas `ClusterGather` progressively accumulates remote data by doubling the message size in each round. This shared structure facilitates uniform implementation and hardware tuning, enabling efficient cluster-level synchronization and data sharing.

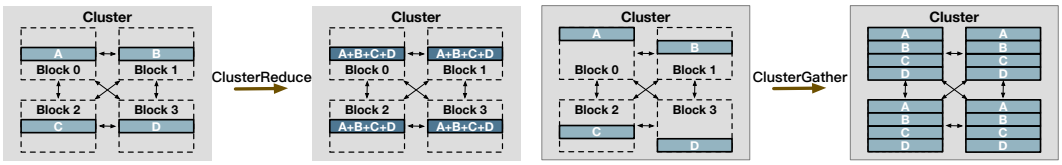

Figure 6: Illustration of cluster-level collective communication primitives: `ClusterReduce` and `ClusterGather`.

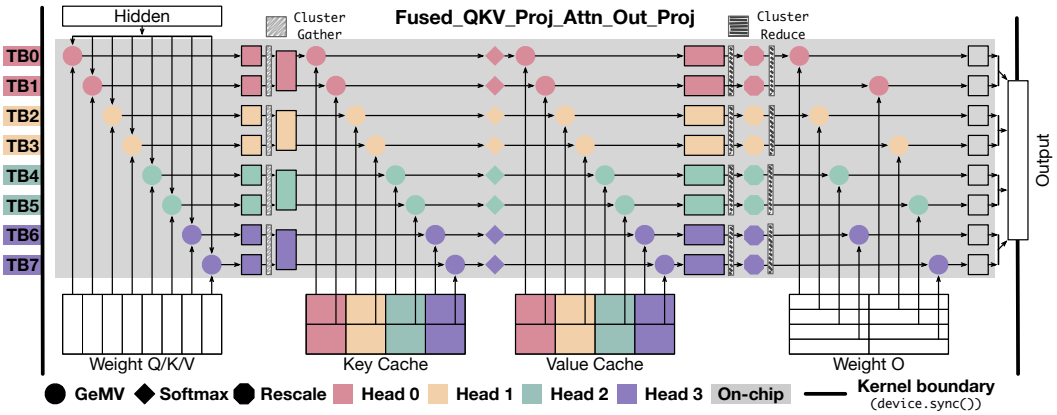

Figure 7: Cluster-centric fused *QKV Projection*, *Attention* and *Output Projection* dataflow graph.

3.2 Cluster-Centric Dataflow Design with Primitives

Building upon the proposed cluster-level primitives, we now illustrate how they are employed to construct a cluster-centric dataflow with expanded operator fusion scope. The core idea is to treat the **thread block cluster** as a cooperative execution and scheduling unit. Data-dependent dimensions are kept within each cluster to resolve inter-block data dependencies using cluster-level collective primitives which avoid off-chip data exchange, while data-independent dimensions are distributed across clusters. This organization aligns computation with memory locality and enables seamless kernel fusion by allowing intermediate results to be reused entirely within on-chip memory.

In the decoding stage, the dataflow for the Projection and *Attention* modules is parallelized over multiple dimensions. Specifically, the Projection is parallel along the number of heads and head dimension, while the *Attention* is parallel over the number of heads and the KV cache sequence length. Among these dimensions, thread blocks that compute different partitions of the head dimension and the KV cache sequence length exhibit inter-block data dependencies, since each block computes either a partial result that requires reduction or a segment result that needs to be gathered to form the final output. These thread blocks can be grouped into a cluster to resolve data dependencies on-chip by using cluster-level collective primitives. Since *Attention* heads are independent across all three modules, each cluster is accordingly mapped to a single head. As shown in Fig. 7, under this design, the intermediate results produced by the *QKV Projection* naturally remain on chip and are directly reused by the *Attention* module. Likewise, the output of the *Attention* module stays on chip and is immediately consumed by the *Output Projection*, enabling seamless data reuse across three modules.

By leveraging cluster-level communication primitives, we implement the fused *QKV Projection*, *Attention* and *Output Projection* dataflow. As illustrated in Alg. 5, the dataflow is parallel in the number of heads. Each head corresponds to a cluster of $N = 2^k$ thread blocks ($k \leq 4$), where each block is assigned a rank $b \in [0, N-1]$. Within each cluster, thread blocks respectively partition the head dimension in *QKV Projection*, the KV cache token dimension in *Attention*, and the output dimension in *Output Projection*. For the whole dataflow, each thread block processes the entire input hidden states and computes the corresponding output tile O_b after the *Output Projection*. In this algorithm, B denotes the batch size, D the input hidden dimension, H the total head dimension, and h, s and d represent the partitioned sizes of the head dimension, sequence length and the output dimension, per thread block, respectively.

According to our cluster-centric dataflow design principle, we also propose several dataflow variants, which are presented in the Appendix B. To evaluate these variants, we conduct a quantitative analysis of DSMEM traffic [47, 59]. We begin by analyzing the DSMEM memory traffic incurred by the `ClusterReduce` and `ClusterGather` primitives as follows:

$$Traffic_{Reduce}(size, N) = size \times \log_2 N \times N, \quad Traffic_{Gather}(size, N) = size \times \left(2^{\log_2 \frac{N}{2}+1} - 1\right) \times N$$

Here, $Traffic_{Reduce}$ and $Traffic_{Gather}$ denote the DSMEM traffic for `ClusterReduce` and `ClusterGather`, respectively. The variable `size` represents the size of the shared memory buffer D_b in Alg. 1, as well as the initial segment size in Alg. 2. Based on this analytical model, we estimate

Algorithm 3 Fused *QKV Projection*, *Attention* and *Output Projection* Dataflow - Thread Block View

Require: Input hidden states $\mathbf{H}_b \in \mathbb{R}^{1\times D}$, *QKV Projection* weight $\mathbf{W}_b^{QKV} \in \mathbb{R}^{D\times 3h}$, *Output Projection* weight $\mathbf{W}_b^{O} \in \mathbb{R}^{H\times d}$, and KV cache $\mathbf{K}_b^{\text{cache}}, \mathbf{V}_b^{\text{cache}} \in \mathbb{R}^{s\times H}$ in global memory.

1: Allocate shared memory buffers: $\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b \in \mathbb{R}^{1\times h}$, and $S_{\text{sum}}, S_{\text{max}}$ (softmax statistics).

2: Compute segment results of *QKV Projection*: $\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b \leftarrow \mathbf{H}_b \times \mathbf{W}_b^{QKV}$.

3: Obatin the complete QKV: $(\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b) \leftarrow \texttt{ClusterGather}(\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b)$.

4: Compute partial result of *Attention* similar to the FlashDecoding dataflow:
Compute $\mathbf{S}_b \leftarrow \exp(\mathbf{Q}_b \times (\mathbf{K}_b^{\text{cache}}, \mathbf{K}_b)^T)$, obtain local $S_{\text{sum}}, S_{\text{max}}$.
And store S_{max} in register Reg_{max}. ▷ softmax statistics.
Compute $\mathbf{A}_b \leftarrow \mathbf{S}_b \times (\mathbf{V}_b^{\text{cache}}, \mathbf{V}_b)$ ▷ $\mathbf{A}_b$ reuse the shared memory space of $\mathbf{Q}_b$

5: Obtain the complete softmax statistics S_{sum} and S_{max}:

$$S_{\text{sum}} \leftarrow \texttt{ClusterReduce}(S_{\text{sum}}, \text{sum}), \quad S_{\text{max}} \leftarrow \texttt{ClusterReduce}(S_{\text{max}}, \text{max})$$

6: Rescale *Attention* output according to online softmax:

$$\mathbf{A}_b \leftarrow \frac{\mathbf{A}_b \cdot \exp(Reg_{\text{max}} - S_{\text{max}})}{S_{\text{sum}}}$$

7: Obatin the complete *Attention* output: $\mathbf{A}_b \leftarrow \texttt{ClusterReduce}(\mathbf{A}_b, \text{sum})$.

8: Compute the result of the *Output Projection* and write it to global memory using `atomicAdd`:

$$\mathbf{O}_b \leftarrow \mathbf{A}_b \times \mathbf{W}_b^{O}$$

9: **return** $\mathbf{O}_b$.

the total DSMEM memory traffic of the dataflow used in Alg. 5, which includes one `ClusterGather` and two `ClusterReduce` operations:

$$Traffic_{Total} = Traffic_{Reduce}(3h, N) + Traffic_{Gather}(H, N)$$

We omit the traffic generated by the softmax statistics, as it involves only two floats and is negligible compared to tensor. According to the analysis shown in Appendix B, this dataflow yields the lowest DSMEM traffic and achieves better performance compared to other dataflow variants.

The proposed cluster-centric dataflow design principle can be naturally generalized to other operators, including DeepSeek MLA [11]. The corresponding algorithms are provided in the Appendix B. We implement an end-to-end execution framework, `ClusterFusion`, based on the cluster-centric dataflow, which incorporates the aforementioned fused *QKV Projection*, Attention, and *Output Projection* modules. For other components such as FFN and RMSNorm, we adopt optimized implementations consistent with those in existing frameworks such as CUTLASS [36] and Flashinfer [55].

4 Evaluation

Experimental Setup We evaluate `ClusterFusion` on an NVIDIA H100 SXM5 80GB GPU [33]. For the end-to-end evaluation, all inputs are in FP16 precision, and the context length varies from 1K to 16K. We set the batch size to 1; results of multi batch are presented in the Appendix C. All experiments are conducted using PyTorch 2.5.1 [40] and CUDA 12.4 [34].

Baselines We compare `ClusterFusion` with four state-of-the-art LLM inference frameworks: SGLang 0.4.3.post2 [58], vLLM 0.6.4.post1 [24], TensorRT-LLM 0.18.0 [39], and MLC-LLM 0.20.dev0 [32]. For all baselines, we use the recommended configurations from their official documentation, including backend kernels from libraries such as CUTLASS [36], FlashAttention [9, 8, 44, 57], and FlashInfer [55], or generated by Triton [50] and TVM [7]. All frameworks enable CUDA Graph [35] and Torch.compile [5].

Models `ClusterFusion` is evaluated on two representative LLMs: Llama2-7B [51] and DeepSeek-V2-Lite [11], both based on the Transformer architecture. Llama2-7B adopts standard Multi-Head Attention (MHA) mechanism, while DeepSeek-V2-Lite employs Multi-head Latent Attention (MLA) algorithm. These models differ in hidden dimensions, head dimensions, and number of heads.

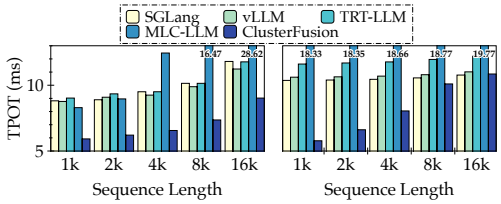

Figure 8: TPOT of Llama2-7B (left) and DeepSeek-V2-Lite (right) on H100.

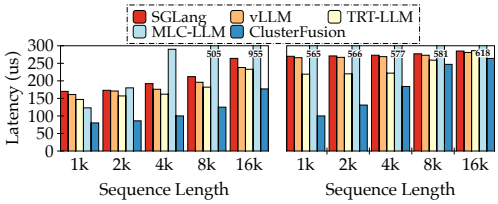

Figure 9: Latency of core modules of Llama2-7B (left) and DeepSeek-V2-Lite (right) on H100.

4.1 End-to-End and Core Module Evaluation

We use time per output token (TPOT) as the metric for end-to-end evaluation. The results are presented in Fig. 17. For baselines, the results include both CUDA graph launch overhead and kernel execution latency. On average, `ClusterFusion` achieves 1.41×, 1.39×, 1.43×, and 2.03× speedups over SGLang, vLLM, TensorRT-LLM, and MLC-LLM across various sequence lengths with a cluster size of 4 on Llama2-7B. For DeepSeek-V2-Lite, `ClusterFusion` delivers average speedups of 1.34×, 1.37×, 1.51×, and 2.39× under the same conditions.

As shown in Fig. 18, for the core *QKV Projection*, *Attention*, and *Output Projection* modules, `ClusterFusion` achieves average speedups of 1.85×, 1.73×, 1.61×, and 3.19× compared to SGLang, vLLM, TensorRT-LLM, and MLC-LLM, respectively on Llama2-7B. Similarly, for DeepSeek-V2-Lite, `ClusterFusion` delivers average speedups of 1.66×, 1.64×, 1.35×, and 3.5×. For the DeepSeek MLA, which is specifically designed to better leverage GPU hardware, the optimization space for operator fusion is relatively limited. Nevertheless, as shown in Fig. 10, sequence lengths in real datasets are predominantly under 8k. In this range, `ClusterFusion` achieves significant performance improvements, demonstrating its effectiveness in real-world scenarios.

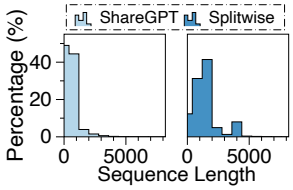

Figure 10: Sequence length distribution in ShareGPT [1] and Splitwise [41, 2] datasets.

We further evaluate the performance of the core modules in `ClusterFusion` under varying cluster sizes and numbers of *Attention* heads, with sequence lengths of 4K and 16K. The results are presented in Fig. 11. In our design, each *Attention* head is mapped to a cluster, so the cluster size determines the internal parallelism within each head. When the number of *Attention* heads is 32 and 64, a cluster size of 4 yields the best performance. However, when the number of heads increases to 128, a smaller cluster size of 2 becomes optimal. Conversely, cluster sizes of 8 and 16 lead to worse performance due to increased interconnect latency, bandwidth contention, and a reduced number of active SMs, which collectively limit overall core utilization, as illustrated in Fig. 5. Based on both theoretical analysis and empirical evidence, we conclude that the optimal cluster size varies across workloads. Therefore, cluster size should be tuned accordingly to maximize performance.

4.2 Speedup Analysis

We identify two primary factors contributing to the performance advantage of `ClusterFusion` over existing frameworks with CUDA Graph optimizations: minimized global memory transfer size and

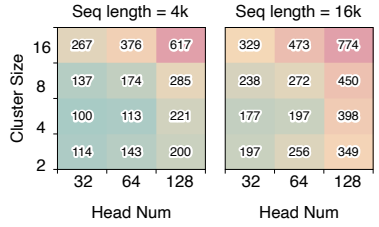

Figure 11: Latency of core module in ClusterFusion with varying settings.

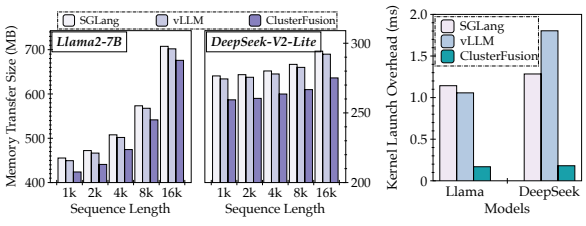

Figure 12: Comparison of global memory data transfer size (left) and GPU kernel launch overhead (right).

reduced GPU kernel launch overhead [63, 53]. To quantify these benefits, we leverage NVIDIA Nsight Systems [38] and Nsight Compute [37] to profile global memory transfer volume and GPU kernel launch overhead across different models and configurations. The results are illustrated in Fig. 19. The performance gain stems from the fact that `ClusterFusion` executes *QKV Projection*, *Attention*, and *Output Projection* entirely on-chip, significantly reducing intermediate memory traffic. Additionally, `ClusterFusion` reduces kernel launch overhead by nearly an order of magnitude in end-to-end scenarios, even when compared to baselines optimized with CUDA Graph.

4.3 Additional Analysis

Table 1: Latency comparison of on-chip ClusterReduce and ClusterGather with DSMEM versus off-chip implementations without DSMEM.

Operation	Data Size (KB)	Off-chip (μs)	On-chip (μs)	Speedup
ClusterReduce	32	8.03	6.77	1.18×
	64	9.01	6.61	1.36×
	128	14.95	7.42	2.01×
	256	22.44	9.17	2.44×
ClusterGather	32	6.26	3.90	1.60×
	64	6.27	4.12	1.52×
	128	6.31	4.39	1.44×
	256	6.61	4.15	1.59×

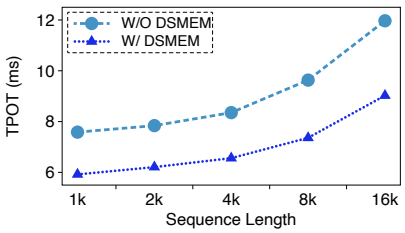

Figure 13: TPOT of `ClusterFusion` on Llama2-7B with and without DSMEM.

To demonstrate the importance of the on-chip interconnect leveraged in our dataflow design, we conduct a microbenchmark to evaluate the cluster-level collective communication primitives introduced in Sec. 3.1. As shown in Tbl. 1, the on-chip ClusterReduce and ClusterGather operations over DSMEM exhibit significantly lower latency across varying data transfer sizes compared to the off-chip implementations.

We further perform ablation studies to compare `ClusterFusion` with and without DSMEM enabled. The experimental results are presented in Fig. 13. Across different sequence lengths, disabling DSMEM increases the time per output token (TPOT) by up to 33%. These results highlight the effectiveness of DSMEM and the cluster-level collective primitives in enabling efficient on-chip reduction and aggregation, thereby improving end-to-end inference performance.

5 Discussion on Fusion Scope and Architectural Outlook

`ClusterFusion` builds on intra-cluster DSMEM communication, where each fused scope is bounded by a fixed cluster size (up to 16 thread blocks) [33]. This imposes constraints on fusion granularity and scheduling flexibility. Although most decoding operators in today's mainstream LLMs [51, 11], such as *Projection* and *Attention*, fit comfortably within this limit, future models with larger hidden dimensions or specialized operator variants may challenge this boundary. When fused operators exceed the cluster scope, the system must fall back to global memory communication, introducing additional latency and runtime fragmentation. This motivates reflection on hardware support for broader intra-chip collectives [20, 6, 27, 18].

Our findings suggest that enabling low-latency, topology-aware communication across a broader set of SMs would unlock more uniform and scalable fusion strategies. Such architectural support could extend structured coordination beyond current cluster boundaries. `ClusterFusion` highlights this co-design opportunity as a practical step toward supporting the growing complexity of architectures.

6 Conclusion

This paper presents `ClusterFusion`, an execution framework that schedules communication and computation jointly to expand operator fusion scope by composing operators during decoding stages such as *QKV Projection*, *Attention*, and *Output Projection* into a single fused kernels. By incorporating cluster-level collective communication primitives, `ClusterFusion` effectively reduces global memory traffic and kernel launch overhead, enabling efficient on-chip execution of key LLM decoding modules. Our comprehensive evaluation on NVIDIA H100 GPUs with representative

models Llama-2-7B and DeepSeek-V2-Lite demonstrates that `ClusterFusion` outperforms state-of-the-art LLM inference frameworks across different models and configurations.

7 Acknowledgement

This work was supported by the National Key R&D Program of China under Grant 2022YFB4501400, and the National Natural Science Foundation of China (NSFC) grant 62222210. This work was also supported by Shanghai Qi Zhi Institute Innovation Program SQZ202316. We sincerely thank Yilu Huang and Chiheng Jin for their assistance with kernel implementation and end-to-end integration. We would also like to thank the anonymous reviewers, as well as the Area Chairs and Program Chairs, for their constructive feedback and valuable comments that helped improve this work.

References

[1] Sharegpt. `https://sharegpt.com/`.

[2] Splitwise. `https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2023.md`.

[3] `https://patents.google.com/patent/US20230289189A1/en`.

[4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. Deepspeed- inference: Enabling efficient inference of transformer models at unprecedented scale. In Felix Wolf, Sameer Shende, Candace Culhane, Sadaf R. Alam, and Heike Jagode, editors, SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022, pages 46:1–46:15. IEEE, 2022.

[5] Jason Ansel, Edward Z. Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, C. K. Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Shunting Zhang, Michael Suo, Phil Tillet, Xu Zhao, Eikan Wang, Keren Zhou, Richard Zou, Xiaodong Wang, Ajit Mathews, William Wen, Gregory Chanan, Peng Wu, and Soumith Chintala. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024, pages 929–947. ACM, 2024.

[6] Cerebras. Cerebras wse-3 chip. `https://www.cerebras.ai/chip`.

[7] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. TVM: an automated end-to-end optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018, pages 578–594. USENIX Association, 2018.

[8] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024.

[9] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022.

[10] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. `https://crfm.stanford.edu/2023/10/12/flashdecoding.html`, 2023.

[11] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Hao Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, Tao Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, and Xiaowen Sun. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. CoRR, abs/2405.04434, 2024.

[12] 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, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, and Wangding Zeng. Deepseek-v3 technical report. CoRR, abs/2412.19437, 2024.

[13] Yue Guan, Changming Yu, Yangjie Zhou, Jingwen Leng, Chao Li, and Minyi Guo. Fractal: Joint multi-level sparse pattern tuning of accuracy and performance for DNN pruning. In Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir, editors, Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024, pages 416–430. ACM, 2024.

[14] Cong Guo, Bo Yang Hsueh, Jingwen Leng, Yuxian Qiu, Yue Guan, Zehuan Wang, Xiaoying Jia, Xipeng Li, Minyi Guo, and Yuhao Zhu. Accelerating sparse DNN models without hardware-support via tile-wise sparsity. In Christine Cuicchi, Irene Qualters, and William T. Kramer, editors, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020, page 16. IEEE/ACM, 2020.

[15] Cong Guo, Yuxian Qiu, Jingwen Leng, Xiaotian Gao, Chen Zhang, Yunxin Liu, Fan Yang, Yuhao Zhu, and Minyi Guo. Squant: On-the-fly data-free quantization via diagonal hessian approximation. In The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022. OpenReview.net, 2022.

[16] Cong Guo, Chen Zhang, Jingwen Leng, Zihan Liu, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. ANT: exploiting adaptive numerical data type for low-bit deep neural network quantization. In 55th IEEE/ACM International Symposium on Microarchitecture, MICRO 2022, Chicago, IL, USA, October 1-5, 2022, pages 1414–1433. IEEE, 2022.

[17] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. CoRR, abs/2401.14196, 2024.

[18] Congjie He, Yeqi Huang, Pei Mu, Ziming Miao, Jilong Xue, Lingxiao Ma, Fan Yang, and Luo Mai. Waferllm: A wafer-scale LLM inference system. CoRR, abs/2502.04563, 2025.

[19] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat GEMM optimization, and heuristics. In Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024. mlsys.org, 2024.

[20] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore IPU architecture via microbenchmarking. CoRR, abs/1912.03413, 2019.

[21] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention, 2024.

[22] Shengyu Liu Jiashi Li. Flashmla: Efficient mla decoding kernels. `https://github.com/deepseek-ai/FlashMLA`, 2025.

[23] Zhixian Jin, Christopher Rocca, Jiho Kim, Hans Kasan, Minsoo Rhu, Ali Bakhoda, Tor M. Aamodt, and John Kim. Uncovering real gpu noc characteristics: Implications on interconnect architecture. In 2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 885–898, 2024.

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023, pages 611–626. ACM, 2023.

[25] Christoph Leiter, Ran Zhang, Yanran Chen, Jonas Belouadi, Daniil Larionov, Vivian Fresen, and Steffen Eger. Chatgpt: A meta-analysis after 2.5 months. CoRR, abs/2302.13795, 2023.

[26] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. In Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024. mlsys.org, 2024.

[27] Yiqi Liu, Yuqi Xue, Yu Cheng, Lingxiao Ma, Ziming Miao, Jilong Xue, and Jian Huang. Scaling deep learning computation over the inter-core connected intelligence processor with T10. In Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024, pages 505–521. ACM, 2024.

[28] Zihan Liu, Jingwen Leng, Zhihui Zhang, Quan Chen, Chao Li, and Minyi Guo. VELTAIR: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling. In Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch, editors, ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022, pages 388–401. ACM, 2022.

[29] Zihan Liu, Xinhao Luo, Junxian Guo, Wentao Ni, Yangjie Zhou, Yue Guan, Cong Guo, Weihao Cui, Yu Feng, Minyi Guo, Yuhao Zhu, Minjia Zhang, Chen Jin, and Jingwen Leng. VQ-LLM: high-performance code generation for vector quantization augmented LLM inference. In IEEE International Symposium on High Performance Computer Architecture, HPCA 2025, Las Vegas, NV, USA, March 1-5, 2025, pages 1496–1509. IEEE, 2025.

[30] Zihan Liu, Wentao Ni, Jingwen Leng, Yu Feng, Cong Guo, Quan Chen, Chao Li, Minyi Guo, and Yuhao Zhu. JUNO: optimizing high-dimensional approximate nearest neighbour search with sparsity-aware algorithm and ray-tracing core mapping. In Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir, editors, Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024, pages 549–565. ACM, 2024.

[31] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Neo Y. Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. Moba: Mixture of block attention for long-context llms. CoRR, abs/2502.13189, 2025.

[32] MLC team. MLC-LLM, 2023-2025.

[33] NVIDA. Nvidia hopper architecture. https://www.nvidia.com/en-us/data-center/technologies/hopper-architecture/.

[34] NVIDIA. Cuda c++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[35] NVIDIA. Cudagraph. https://developer.nvidia.com/blog/cuda-graphs/.

[36] NVIDIA. Cutlass. https://github.com/NVIDIA/cutlass.

[37] NVIDIA. Nvidia nsight compute. https://developer.nvidia.com/nsight-compute.

[38] NVIDIA. Nvidia nsight system. https://developer.nvidia.com/nsight-systems.

[39] NVIDIA. Tensorrt-llm. https://github.com/NVIDIA/TensorRT-LLM.

[40] 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 Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 8024–8035, 2019.

[41] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative LLM inference using phase splitting. In 51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024, pages 118–132. IEEE, 2024.

[42] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. CoRR, abs/2308.12950, 2023.

[43] SGLang. Sglang deepseek model optimizations. https://github.com/sgl-project/sgl-learning-materials/blob/main/slides/sglang_deepseek_model_optimizations.pdf.

[44] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.

[45] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. CoRR, abs/2402.03300, 2024.

[46] Noam Shazeer. Fast transformer decoding: One write-head is all you need. CoRR, abs/1911.02150, 2019.

[47] Yining Shi, Zhi Yang, Jilong Xue, Lingxiao Ma, Yuqing Xia, Ziming Miao, Yuxiao Guo, Fan Yang, and Lidong Zhou. Welder: Scheduling deep learning memory access via tile-graph. In 17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023, pages 701–718. USENIX Association, 2023.

[48] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. CoRR, abs/1909.08053, 2019.

[49] Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. A contrastive framework for neural text generation. In Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022.

[50] Philippe Tillet, Hsiang-Tsung Kung, and David D. Cox. 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, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019, pages 10–19. ACM, 2019.

[51] 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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. CoRR, abs/2307.09288, 2023.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008, 2017.

[53] Chunwei Xia, Jiacheng Zhao, Qianqi Sun, Zheng Wang, Yuan Wen, Teng Yu, Xiaobing Feng, and Huimin Cui. Optimizing deep learning inference via global analysis and tensor expressions. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024, pages 286–301. ACM, 2024.

[54] An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. CoRR, abs/2409.12122, 2024.

[55] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for LLM inference serving. CoRR, abs/2501.01005, 2025.

[56] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention. CoRR, abs/2502.11089, 2025.

[57] Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang, Zizhong Chen, Xin Liu, and Yibo Zhu. Bytetransformer: A high-performance transformer boosted for variable-length inputs. In IEEE International Parallel and Distributed Processing Symposium, IPDPS 2023, St. Petersburg, FL, USA, May 15-19, 2023, pages 344–355. IEEE, 2023.

[58] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024, 2024.

[59] Size Zheng, Siyuan Chen, Siyuan Gao, Liancheng Jia, Guangyu Sun, Runsheng Wang, and Yun Liang. Tileflow: A framework for modeling fusion dataflow via tree-based analysis. In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2023, Toronto, ON, Canada, 28 October 2023 - 1 November 2023, pages 1271–1288. ACM, 2023.

[60] Yangjie Zhou, Jingwen Leng, Yaoxu Song, Shuwen Lu, Mian Wang, Chao Li, Minyi Guo, Wenting Shen, Yong Li, Wei Lin, Xiangwen Liu, and Hanqing Wu. ugrapher: High-performance graph operator computation via unified abstraction for graph neural networks. In Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift, editors, Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023, pages 878–891. ACM, 2023.

[61] Yangjie Zhou, Wenting Shen, Jingwen Leng, Shuwen Lu, Zihan Liu, Weihao Cui, Zhendong Zhang, Wencong Xiao, Baole Ai, Yong Li, Wei Lin, Deze Zeng, Yun Liang, Quan Chen, Ning Liu, and Minyi Guo. Voyager: Input-adaptive algebraic transformations for high-performance graph neural networks. In Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS '25, page 247–263, New York, NY, USA, 2025. Association for Computing Machinery.

[62] Yangjie Zhou, Honglin Zhu, Qian Qiu, Weihao Cui, Zihan Liu, Peng Chen, Mohamed Wahib, Cong Guo, Siyuan Feng, Jintao Meng, et al. A sample-free compilation framework for efficient dynamic tensor computation. 2025.

[63] Donglin Zhuang, Zhen Zheng, Haojun Xia, Xiafei Qiu, Junjie Bai, Wei Lin, and Shuaiwen Leon Song. Mononn: Enabling a new monolithic optimization space for neural network inference tasks on modern gpu-centric architectures. In 18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024, pages 989–1005. USENIX Association, 2024.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstraction and introduction reflect necessary contributions and experiments.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The potential limitations is discussed within the paper in a separate Sec. 5.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The necessary assumption and proof are include in the paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We have listed detail configuration of experiments in Sec. 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The code to reproduce the experimental results are provided in the supplementary material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: This is not the core contribution of this work. But still, we follow the standard method of prior work as we have mentioned in the Sec. 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: The experiments are conducted many times and report average results.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The specific configurations are included in the Sec. 4.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the potential positive societal impacts and negative societal impacts of the work in the paper.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The answer NA means that the paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the original paper that produced the code package or dataset.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: This paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLM is used only for writing and editing in this paper.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Related Works

Operator-Level Optimizations Numerous researchs have explored operator-level optimizaitons for LLM inference [60, 30, 28, 21, 61, 62]. FlashAttention [9, 8] fuses the entire Attention operation into a single memory-efficient kernel. Building on this, FlashDecoding [10] extends parallelism to the KV cache sequence dimension during decoding. FlashDecoding++ [19] proposes to determine the scaling factor based on statistics in advance and introduces FlatGEMM to represent the GEMM with a highly reduced dimension in decoding. FlashAttention-3 [44] demonstrated that warp specialization and new hardware features [33] such as asynchrony can have a significant impact on the Attention. FlashMLA [22] design an efficient decoding Attention kernel for DeepSeek MLA architecture inspired by FlashAttention-3. However, there optimizaitons adopt a block-isolated execution pattern. Due to the lack of structured communication across thread blocks, intermediate results are repeatedly written to and read from global memory, limiting opportunities for broader operator fusion and on-chip reuse. `ClusterFusion` explores a more general fusion space enabled by efficient cluster-scoped collective primitives.

Algorithm-Level Optimizations Several studies focus on improving LLM inference efficiency through algorithmic changes [26, 29, 56, 31]. Techniques such as quantization and sparsification aim to reduce computational and memory overhead. Quantization [26, 29] compresses model weights and activations by converting high-bitwidth representations into lower-bitwidth ones, reducing arithmetic and memory costs. Pruning [56, 31] increases the proportion of zero elements in weights or activations, enabling hardware to skip redundant computations. These techniques are orthogonal to our work. `ClusterFusion` does not modify the model workload but can complement these approaches by optimizing the underlying dataflow and kernel execution through structure and reusable primitives.

Systems on Inter-Core Connected Hardwares The on-chip inter-core interconnect has also been adopted in alternative hardware platforms such as Graphcore IPU [20] and Cerebras WSE [6]. Several prior works have explored leveraging this capabilities to optimize deep learning workloads. T10 [27] is an end-to-end deep learning compiler targeting inter-core connected intelligence processors, with an emphasis on fine-grained modeling of data movement between cores. WaferLLM [18], on the other hand, is the first system to propose a LLM parallelism solution tailored for wafer-scale accelerators. Moreover, these systems are closely tied to custom hardware architectures that offer full inter-core connectivity or integrates inter-core communication mechanisms into specific operators like GEMM and GEMV. As such, their design assumptions do not translate well to modern GPU architectures like NVIDIA Hopper which remains the dominant platform for LLM inference deployment. `ClusterFusion` introduces cluster-scoped communication primitives which abstract common aggregation and reduction patterns and can be flexibly reused across different workloads.

B Additional Dataflows

In this section, we first introduce the computation process of DeepSeek MLA and present our fused MLA dataflow, which leverages cluster-level collective communication primitives, as evaluated in the main paper. We then describe an alternative dataflow in which thread blocks partition the head dimension within the *Attention* module, in contrast to the dataflow in the main paper that partitions the KV cache along the token dimension, demonstrating the capability of our primitives in enabling different dataflows for the same computation. Finally, we evaluate these dataflow variants by analyzing their corresponding DSMEM traffic.

B.1 DeepSeek MLA

DeepSeek introduces the Multi-Head Latent Attention (MLA) mechanism, which has been adopted in its model series [11, 12]. During inference, a weight absorption optimization[43] is applied in the decoding stage of MLA to reduce overall computational cost. The detailed computation processes of both the original MLA and the optimized version in DeepSeek-V2-Lite[11] are illustrated in Fig. 14. In the original MLA computation, the input hidden state first undergoes a *Q Projection* to generate the multi-head $\mathbf{Q}$, and a *Down Projection* to obtain the compressed KV cache. The newly generated compressed KV is then concatenated with the cached KV and passed through an *Up Projection* to produce the multi-head $\mathbf{KV}$, which is used in the MHA module. In the weight absorption version of

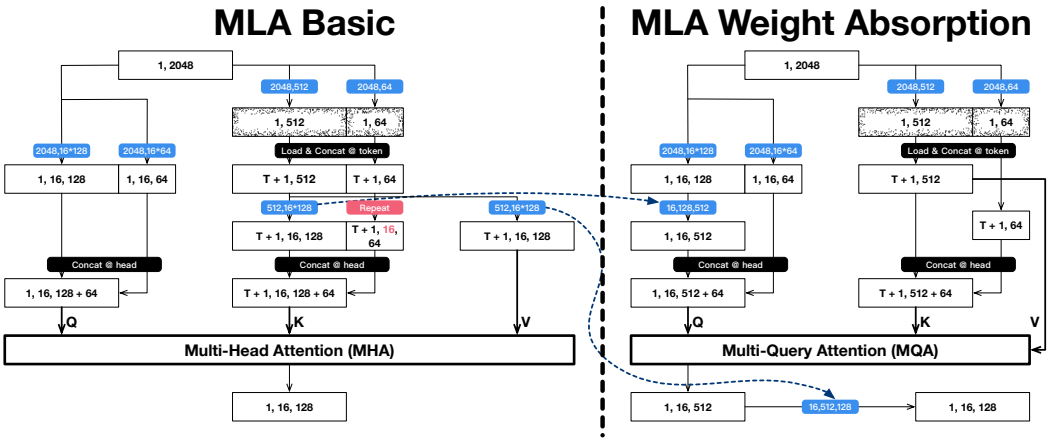

Figure 14: Overview of MLA computation: original (left) and with weight absorption optimization (right). The black stipple represents the newly generated latent presentation, which will be cached to calculate keys and values for *Attention* computation.

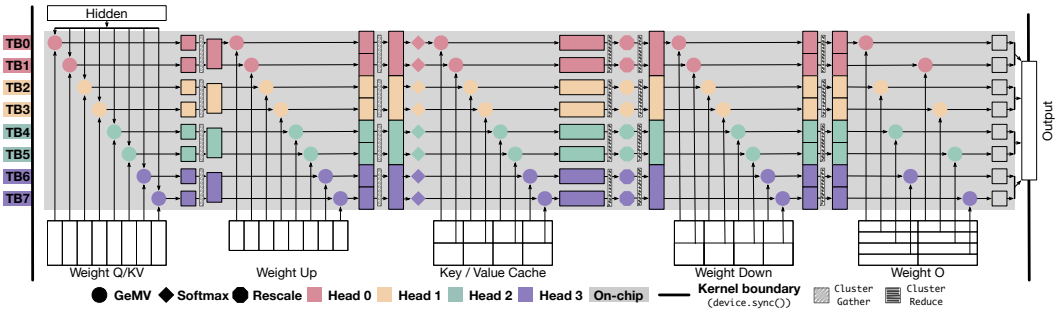

Figure 15: Cluster-centric fused MLA dataflow graph.

MLA, the process begins by computing $\mathbf{Q}$ and $\mathbf{K}$, where $\mathbf{V}$ is obtained by partially reusing $\mathbf{K}$:

$$Q = \text{Hidden} \times W_Q \times W_{\text{Up}}, \quad K = \text{Hidden} \times W_K, \quad V = K[:\texttt{kv\_lora\_rank}] \tag{3}$$

Next, the *Attention* module is computed as:

$$Z = \text{Concat}\left(\text{Softmax}\left(\frac{Q_1 K^\top}{\sqrt{d_k}}\right)V, \ldots, \text{Softmax}\left(\frac{Q_h K^\top}{\sqrt{d_k}}\right)V\right) \tag{4}$$

Finally, the result is passed through a *Down Projection* to produce the final *Attention* output:

$$\text{Output} = Z \times W_{\text{Down}} \tag{5}$$

The key difference between MLA and conventional MHA lies in the introduction of additional projection layers, specifically the *Up Projection* and *Down Projection*, which are designed to preserve mathematical consistency. Additionally, MLA employs the MQA mechanism [46], in which all $\mathbf{Q}$ heads share a single $\mathbf{KV}$ cache head. This approach increases computational intensity while reducing the memory access associated with the $\mathbf{KV}$ cache. Moreover, the head dimension in MLA corresponds to the value of `kv_lora_rank`, which is typically larger than that in other models. For example, in DeepSeek-V2-Lite, it is 512, whereas in Llama2-7B, the head dimension is 128.

According to our cluster-centric dataflow design principle, the fused MLA dataflow is illustrated in Fig. 15. This dataflow performs the entire MLA computation on-chip, eliminating any intermediate off-chip memory traffic. The detailed algorithm is presented in Alg. 4. This dataflow is parallelized across attention heads, with each head assigned to a cluster consisting of $N = 2^k$ thread blocks ($k \leq 4$). Each thread block within a cluster is assigned a rank $b \in [0, N-1]$. Within each cluster, thread blocks collaboratively partition the head dimension for the *QKV Projection*; the

Algorithm 4 Fused MLA Dataflow - Thread Block View

Require: Input hidden states $\mathbf{H}_b \in \mathbb{R}^{1\times D}$, *Q Projection* weight $\mathbf{W}_b^Q \in \mathbb{R}^{D\times h}$, *KV Projection* weight $\mathbf{W}_b^{KV} \in \mathbb{R}^{D\times l}$, *Up Projection* weight $\mathbf{W}_b^{Up} \in \mathbb{R}^{H\times l}$, *Down Projection* weight $\mathbf{W}_b^{Down} \in \mathbb{R}^{l\times H}$, *Output Projection* weight $\mathbf{W}_b^O \in \mathbb{R}^{H\times d}$, and KV cache $\mathbf{KV}_b^{\text{cache}} \in \mathbb{R}^{s\times L}$ in global memory.

1: Allocate shared memory buffers: $\mathbf{Q}_b, \mathbf{KV}_b \in \mathbb{R}^{1\times l}$, and $S_{\text{sum}}, S_{\text{max}}$ (softmax statistics).
2: Compute segment results of *Q Projection*: $\mathbf{Q}_b \leftarrow \mathbf{H}_b \times \mathbf{W}_b^Q$.
3: Compute segment results of *KV Projection*: $\mathbf{KV}_b \leftarrow \mathbf{H}_b \times \mathbf{W}_b^{KV}$.
4: Obtain the complete QKV: $\mathbf{Q}_b \leftarrow \texttt{ClusterGather}(\mathbf{Q}_b)$, $\mathbf{KV}_b \leftarrow \texttt{ClusterGather}(\mathbf{KV}_b)$.
5: Compute segment results of *Up Projection* by batch matmul: $\mathbf{Q}_b \leftarrow \mathbf{Q}_b \times \mathbf{W}_b^{Up}$.
6: Obtain the complete Q: $\mathbf{Q}_b \leftarrow \texttt{ClusterGather}(\mathbf{Q}_b)$.
7: Compute partial result of *Attention* similar to the FlashDecoding dataflow:
Compute $\mathbf{S}_b \leftarrow \exp(\mathbf{Q}_b \times (\mathbf{KV}_b^{\text{cache}}, \mathbf{KV}_b)^T)$, obtain local $S_{\text{sum}}, S_{\text{max}}$.
And store S_{max} in register Reg_{max}. ▷ softmax statistics.
Compute $\mathbf{A}_b \leftarrow \mathbf{S}_b \times (\mathbf{KV}_b^{\text{cache}}, \mathbf{KV}_b)$ ▷ $\mathbf{A}_b$ reuse the shared memory space of $\mathbf{Q}_b$
8: Obtain the complete softmax statistics S_{sum} and S_{max}:

$$S_{\text{sum}} \leftarrow \texttt{ClusterReduce}(S_{\text{sum}}, \text{sum}), \quad S_{\text{max}} \leftarrow \texttt{ClusterReduce}(S_{\text{max}}, \text{max})$$

9: Rescale *Attention* output according to online softmax:

$$\mathbf{A}_b \leftarrow \frac{\mathbf{A}_b \cdot \exp(Reg_{\text{max}} - S_{\text{max}})}{S_{\text{sum}}}$$

10: Obtain the complete *Attention* output: $\mathbf{A}_b \leftarrow \texttt{ClusterReduce}(\mathbf{A}_b, \text{sum})$.
11: Compute partial results of *Down Projection* by batch matmul: $\mathbf{A}_b \leftarrow \mathbf{A}_b \times \mathbf{W}_b^{Down}$.
12: Obtain the complete *Down Projection* output: $\mathbf{A}_b \leftarrow \texttt{ClusterReduce}(\mathbf{A}_b, \text{sum})$.
13: Compute the result of the *Output Projection* and write it to global memory using `atomicAdd`:

$$\mathbf{O}_b \leftarrow \mathbf{A}_b \times \mathbf{W}_b^O$$

14: **return** $\mathbf{O}_b$.

`kv_lora_rank` dimension for the *Up Projection* and *Down Projection* modules; the token dimension of the KV cache for the *Attention* module; and the output dimension for the *Output Projection*. And each thread block processes the full input hidden states and computes its corresponding output tile O_b after the *Output Projection*. In this algorithm, B denotes the batch size, D the input hidden dimension, and H the total head dimension. The variables h, l, s, and d refer to the partitioned sizes of the head dimension, `kv_lora_rank`, sequence length, and output dimension per thread block, respectively. For simplicity, we omit the `rope_dim` shown in Fig. 14.

We also estimate the total DSMEM traffic incurred by the dataflow used in Alg. 4, which involves three `ClusterGather` operations and three `ClusterReduce` operations. The DSMEM traffic for the `ClusterGather` operations is given by:

$$Traffic_{Gather}(h, N) + 2 \times Traffic_{Gather}(l, N)$$

and the traffic for the `ClusterReduce` operations is:

$$Traffic_{Reduce}(l, N) + Traffic_{Reduce}(H, N)$$

We omit the traffic introduced by the softmax statistics, as it involves only two floating-point values and is negligible compared to the tensor-level data movement.

B.2 SplitHead Dataflow

In line with our cluster-centric dataflow design principles and by leveraging cluster-level communication primitives, we implement the fused *QKV Projection*, *Attention*, and *Output Projection* dataflow described in the main paper, where intermediate data is stored in on-chip shared memory. In addition, we design an alternative dataflow called SplitHead dataflow that stores the intermediate data in faster

Algorithm 5 SplieHead Dataflow - Thread Block View

Require: Input hidden states $\mathbf{H}_b \in \mathbb{R}^{B\times D}$, *QKV Projection* weight $\mathbf{W}_b^{QKV} \in \mathbb{R}^{D\times 3h}$, *Output Projection* weight $\mathbf{W}_b^{O} \in \mathbb{R}^{h\times D}$, and KV cache $\mathbf{K}_b^{\text{cache}}, \mathbf{V}_b^{\text{cache}} \in \mathbb{R}^{S\times h}$ in global memory.

1: Allocate register memory buffers: $\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b \in \mathbb{R}^{B\times h}$, shared memory buffer: $\mathbf{S}_b \in \mathbb{R}^{S\times B}$.

2: Compute segment results of *QKV Projection*: $\mathbf{Q}_b, \mathbf{K}_b, \mathbf{V}_b \leftarrow \mathbf{H}_b \times \mathbf{W}_b^{QKV}$.

3: Compute segment result of *Attention*:

Compute $\mathbf{S}_b \leftarrow \mathbf{Q}_b \times (\mathbf{K}_b^{\text{cache}}, \mathbf{K}_b)^T$.

Obtain the complete result of $\mathbf{Q} \times \mathbf{K}^T$: $\mathbf{S}_b \leftarrow \texttt{ClusterReduce}(\mathbf{S}_b, \text{sum})$.

Compute $Softmax$: $\mathbf{S}_b \leftarrow softmax(\mathbf{S}_b)$.

Compute *Attention* output: $\mathbf{A}_b \leftarrow \mathbf{S}_b \times (\mathbf{V}_b^{\text{cache}}, \mathbf{V}_b)$

4: Compute the partial one head result of the *Output Projection*: $\mathbf{O}_b \leftarrow \mathbf{A}_b \times \mathbf{W}_b^{O}$.

5: Obtain the complete one head result of the *Output Projection*:

$$\mathbf{O} \leftarrow \texttt{ClusterReduce}(\mathbf{O}_b, sum)$$

6: Write the complete result of the *Output Projection* to global memory using `atomicAdd`:

$$\mathbf{O} \leftarrow \texttt{atomicAdd}(\mathbf{O})$$

7: **return O**.

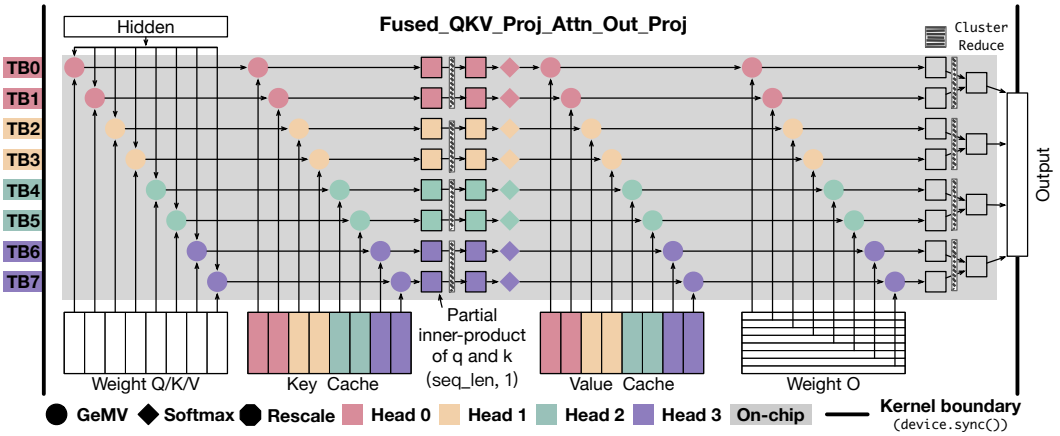

Figure 16: Cluster-centric fused *QKV Projection*, *Attention* and *Output Projection* dataflow graph.

on-chip register memory. As illustrated in Alg. 5, this dataflow is parallel in the number of heads. Each head corresponds to a cluster of $N = 2^k$ thread blocks ($k \leq 4$), where each block is assigned a rank $b \in [0, N-1]$. Within each cluster, thread blocks just partition the head dimension in *QKV Projection*, *Attention*, and *Output Projection*. For the whole dataflow, each thread block processes the entire input hidden states and computes the corresponding partial output O_b after the *Output Projection*. In this algorithm, B denotes the batch size, D the input hidden dimension, H the total head dimension, S the KV cache sequence length, and h, d represent the partitioned sizes of the head dimension and the output dimension, per thread block, respectively. In this design, we need to reduce the result of $\mathbf{Q} \times \mathbf{K}^T$ which has a shape of $Sequence\ Length \times Batch\ Size$. Each thread block holds the segment *Attention* output and computes the partial *Output Projection*, which must then be reduced and written to global memory using `atomicAdd`.

As shown in Fig. 16, this dataflow enables intermediate results such as $\mathbf{Q}_b$, $\mathbf{K}_b$, and $\mathbf{V}_b$ to be stored in faster register memory, instead of shared memory. However, the total DSMEM traffic incurred by this dataflow is

$$Traffic_{Total} = Traffic_{Reduce}(S, N) + Traffic_{Reduce}(D, N)$$

which is higher than the dataflow that partitions the KV cache along the token dimension in the *Attention* module, as proposed in the main paper and Sec. B.1. The total DSMEM traffic of the

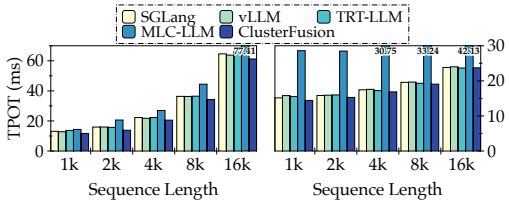

Figure 17: TPOT of Llama2-7B (left) and DeepSeek-V2-Lite (right) on H100.

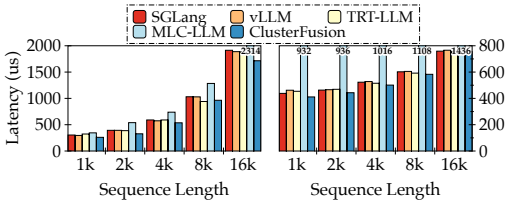

Figure 18: Latency of core modules of Llama2-7B (left) and DeepSeek-V2-Lite (right) on H100.

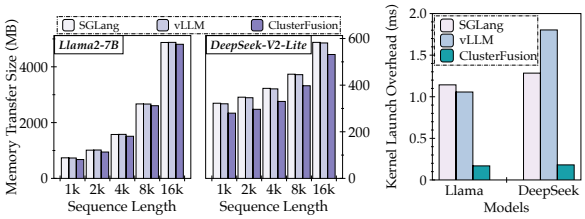

Figure 19: Comparison of global memory data transfer size (left) and GPU kernel launch overhead (right).

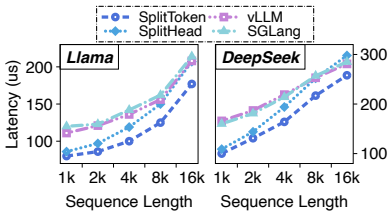

Figure 20: Latency comparision of SplitToken and SplitHead dataflows.

SplitToken dataflow is

$$Traffic_{Total} = Traffic_{Reduce}(H, N) + Traffic_{Gather}(3h, N)$$

which is significantly lower, as it mainly depends on the head dimension H or the `kv_lora_rank` l in the fused MLA dataflow, both of which are much smaller than the sequence length S that dominates the DSMEM traffic in the SplitHead dataflow. The increased communication overhead outweighs the benefits of using register memory. As demonstrated by our experimental results presented later, the SplitHead dataflow yields higher latency.

C Additional Experiments

C.1 Multi-Batch Evaluation and Speedup Analysis

We conduct additional experiments using a batch size of 16, while keeping other experimental settings consistent with those described in the main paper. The TPOT and latency results for core modules are shown in Fig. 17 and Fig. 18, respectively. On average, `ClusterFusion` achieves speedups of 1.11×, 1.09×, 1.12×, and 1.32× over SGLang, vLLM, TensorRT-LLM, and MLC-LLM, respectively, across various sequence lengths on Llama2-7B with a cluster size of 4. For DeepSeek-V2-Lite, `ClusterFusion` delivers average speedups of 1.15×, 1.14×, 1.07×, and 1.84× under the same conditions. In terms of the core modules, including fused *QKV Projection*, *Attention*, and *Output Projection*, `ClusterFusion` achieves average speedups of 1.14×, 1.12×, 1.2×, and 1.41× over SGLang, vLLM, TensorRT-LLM, and MLC-LLM, respectively, on Llama2-7B. Similarly, for DeepSeek-V2-Lite, the corresponding speedups are 1.19×, 1.18×, 1.14×, and 2.04×.

We identify two primary factors contributing to the performance advantage of `ClusterFusion` over existing frameworks with CUDA Graph optimizations: reduced global memory transfer volume and significantly lower GPU kernel launch overhead [63, 53]. To quantify these benefits, we use NVIDIA Nsight Systems [38] and Nsight Compute [37] to profile memory traffic and kernel launch overhead under a batch size of 16. The results are presented in Fig. 19. The observed performance gains are primarily attributed to `ClusterFusion` executing *QKV Projection*, *Attention*, and *Output Projection* entirely on-chip, thereby mi nimizing intermediate memory traffic. Furthermore, `ClusterFusion` reduces kernel launch overhead by nearly an order of magnitude in end-to-end scenarios, even compared to baselines already optimized with CUDA Graph. However, the reduction in global memory traffic has limited impact in the multi-batch scenario, as the KV cache and model weights still dominate memory usage, while the intermediate memory footprint remains small. Moreover, the overall computation intensity increases significantly with larger batch sizes, leading to a reduced speedup compared to the single-batch results presented in the main paper.

C.2 SplitHead Dataflow Evaluation

We also implement the SplitHead dataflow described in Sec. B.2 and present the performance comparison between the SplitToken, SplitHead dataflows and two representive baselines in Fig. 20. When the sequence length is short, the latency difference is minimal because intermediate data can be stored in register memory, which improves efficiency compared to the SplitToken, and the gap in DSMEM traffic compared to the SplitToken dataflow remains small. However, as the sequence length increases, the DSMEM traffic of the SplitHead dataflow grows significantly larger than that of SplitToken, resulting in increased latency. From the perspective of operator fusion efficiency, the SplitHead dataflow enables fusion with intermediate data stored in high-speed register memory, as long as the data size is small enough to fit entirely within the registers. However, despite this benefit, the SplitHead dataflow incurs significantly higher DSMEM traffic, especially as the sequence length increases. This increased traffic becomes a major bottleneck and leads to worse overall performance. Therefore, our cluster-centric dataflow design takes into account not only fusion efficiency but also the memory traffic of different dataflows, aiming to achieve better overall performance.