

Efficient and Economic Large Language Model Inference with Attention Offloading

Shaoyuan Chen
Tsinghua University

Yutong Lin
Tsinghua University

Mingxing Zhang
Tsinghua University

Yongwei Wu
Tsinghua University

Abstract

Transformer-based large language models (LLMs) exhibit impressive performance in generative tasks but introduce significant challenges in real-world serving due to inefficient use of the expensive, computation-optimized accelerators. This mismatch arises from the autoregressive nature of LLMs, where the generation phase comprises operators with varying resource demands. Specifically, the **attention operator is memory-intensive**, exhibiting a memory access pattern that clashes with the strengths of modern accelerators, especially as context length increases.

To enhance the efficiency and cost-effectiveness of LLM serving, we introduce the concept of **attention offloading**. This approach leverages a collection of **cheap, memory-optimized devices for the attention operator** while still utilizing high-end accelerators for other parts of the model. This heterogeneous setup ensures that each component is tailored to its specific workload, maximizing overall performance and cost efficiency. Our comprehensive analysis and experiments confirm the viability of splitting the attention computation over multiple devices. Also, the communication bandwidth required between heterogeneous devices proves to be manageable with prevalent networking technologies. To further validate our theory, we develop Lamina, an LLM inference system that incorporates attention offloading. Experimental results indicate that Lamina can provide $1.48\times$ – $12.1\times$ higher estimated throughput per dollar than homogeneous solutions.

1 Introduction

The emergence of transformer-based large language models (LLMs) has been attracting widespread attention due to their remarkable performance in generative tasks, including chatbots [26, 28] and coding assistants [13, 37]. The intensive computation required for these models has propelled the development of dedicated accelerators (e.g., H100, TPU v5e). These accelerators, while costly, stand out due to their unprecedented computational power measured in teraflops.

In particular, these specialized accelerators often integrate high-performance computational units and high-bandwidth memories (HBM) within a single package, delivering optimal performance for large-scale model processing. In conjunction with advanced distributed training frameworks, these devices can achieve optimized model FLOPS utilization (MFU) in LLM training.

However, the ever-growing deployment of LLMs has introduced another critical challenge: the high inference costs. This challenge primarily stems from the **reduced MFU during inference phases**, which is a result of the unique computation pattern inherent to LLM inferences. This pattern does not align well with the capabilities of modern accelerators, leading to a significant underutilization of hardware resources.

The process of LLM inference involves two phases. The first, known as the *prefill* phase, processes all input tokens from the prompt in parallel. This phase resembles the forward pass in model training and displays a high computation efficiency. However, the subsequent *generation* phase, where tokens are generated sequentially one after another, requires much higher memory bandwidth. Even worse, conventional methods for enhancing GPU utilization, such as increasing batch sizes, are less effective in this phase due to the limited capacity of GPU memory.

A detailed examination reveals that the generation phase mainly comprises two types of operators, each facing distinct resource bottlenecks. Linear transformations, including QKV projections and feedforward networks, are implemented with GEMM (GEneralized Matrix-Matrix multiplication) operations. Since all requests multiply with the same parameter matrices in these operators, processing multiple requests in batch can avoid repeated parameter loads from memory, making these operators primarily computation-bound. Also, the GEMMs can be efficiently handled by specialized circuits (e.g., tensor cores) in cutting-edge accelerators. However, the self-attention operator is different. This pivotal operator requires each request to access its own, distinct data (KV cache) for computation, resulting in a batched GEMV (GEneralized Matrix-Vector multiplication) pattern. Increasing batch

sizes does not reduce the bandwidth requirement but places additional pressure on the already limited memory capacity. This memory-bound workload disagrees with the inherent strengths of modern accelerators, resulting in a scenario where the memory controllers are overwhelmed while the powerful computation cores remain idle. Furthermore, as the sequence length gets longer, the size of KV caches grows proportionally, which further deteriorates the overall accelerator utilization.

Our contributions. Addressing the challenges in LLM inference has led to a plethora of solutions, ranging from model quantization and pruning to enhance performance, to system-level optimizations like continuous batching [43] and paged-attention [18] to improve hardware utilization. Hitherto, these methods have predominantly focused on homogeneous architectures, employing only high-end flagship accelerators for inference. However, our research suggests that the unique characteristics of the LLM generation phase call for a heterogeneous architecture for better efficiency and lower cost.

We surveyed a diverse spectrum of accelerators, each distinguished by specific features ideally suited for particular aspects of LLM inference. For instance, as highlighted in Table 1, consumer-grade GPUs exhibit exceptional cost-effectiveness in memory-related characteristics, providing more than $3\times$ more memory capacity and bandwidth per dollar compared to their high-end counterparts. These alternative accelerators, while falling short in raw computation power and scalability, introduce economic options for efficient handling of the attention operator. Furthermore, we anticipate that the evolution of Processing-in-Memory (PIM) devices [5, 14, 16, 19] will demonstrate even greater cost advantages alongside their larger capacity and higher bandwidth. Compared to conventional accelerators, PIMs provide less computation power and lack uniform access performance to the whole memory from all computational units, but these attributes are simply not required for efficient handling of the attention operator.

Table 1: Memory specifications and cost efficiency of enterprise- and consumer-grade accelerators.

Accelerator	Price/\$	Memory Spec.		Cost Efficiency	
		Cap. (GB)	Bw. (GB/s)	\$/GB	\$(/GB/s)
A100-80G	17500	80	1935	219	9.04
H100	36500	80	3350	456	10.90
RTX 3090	1600	24	936.2	67	1.71
RX 7900 XTX	1000	24	960	42	1.04
Arc A770	350	16	560	22	0.63

In light of these findings, we propose an innovative concept of **attention offloading**. This approach involves creating two pools of accelerators — one optimized for computational power and the other for memory bandwidth efficiency. Adopting this heterogeneous architecture allows us to design a

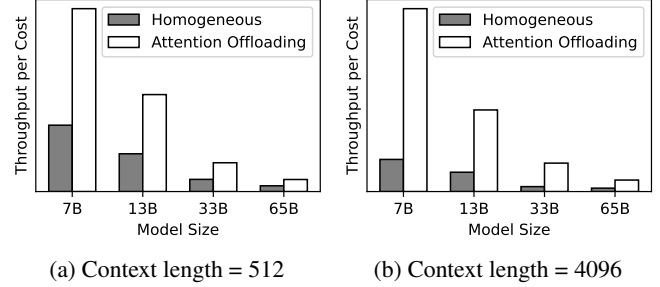


Figure 1: Relative estimated throughput per cost when serving different models, using optimal hardware configurations.

serving system that flexibly delivers the three essential components (i.e., computational power, memory capacity, and bandwidth) for high-performance LLM inference in a cost-efficient manner. To demonstrate the effectiveness of our approach, Figure 1 presents a comparison of estimated throughput per dollar between a traditional setup using homogeneous A100 clusters and our proposed architecture, which pairs each A100 with several memory-optimized devices. As we can see, this novel approach effectively aligns the resource demands of LLM inference with the strengths of a diverse range of hardware platforms.

More importantly, our approach not only separates accelerators into two distinct resource pools but also leads to an extremely elastic architecture. The linear transformation operators processed in the computation pool are essentially stateless operators, which facilitates natural scalability to match fluctuating workloads. This adaptability is ideal for maximizing the utilization of these precious resources.

Like other disaggregated architectures, the feasibility of our proposed attention-offloading architecture hinges on its communication demands within realistic settings. Specifically, our research aims to address two critical questions. (1) What is the minimum bandwidth requirement for interconnecting these two pools of accelerators? It would be impractical if only exceptionally high interconnect bandwidths could meet these needs. (2) While the benefits in terms of throughput are apparent in our architecture, what impact does it have on inference latency? Excessive latency could restrict the range of applicable scenarios.

Fortunately, our studies indicate that these concerns are manageable in the context of LLM inference, making it a suitable scenario for applying the disaggregated architecture. Firstly, we provide detailed profiling and a theoretical model to calculate the ratio needed for balancing between computational power and memory in varying workloads. This model also assists in determining the minimum bandwidth threshold between different accelerator pools. Our findings reveal that not only conventional system buses such as PCIe 4.0 could meet our needs, networking technologies like 200Gb Infini-band or even Ethernet, already widely deployed in current

AI-oriented data centers nowadays, also suffice. The feasibility of inter-node communications for attention offloading also enables more flexible designs of disaggregated LLM inference systems.

Attention offloading may introduce additional latency due to the added overhead of scheduling and networking. To mitigate this, we have employed various techniques, such as GPUDirect RDMA and device-side busy polling, which have proven effective in reducing data transfer times. Furthermore, we have implemented communication-attention overlapping, a strategy that conceals networking time by overlapping it with attention computation. Experimental results demonstrate that our attention offloading approach can achieve latency comparable to that of existing solutions, ensuring the user experience of LLM services.

With attention offloading, the inference process with a single batch results in underutilization of resources, as the memory device remains idle when the computation device is active, and vice versa. To address this inefficiency and enhance cost-effectiveness, we introduce staggered pipelining, an advanced technique that maximizes resource utilization without compromising inference latency. Through staggered pipelining, we run multiple batches concurrently and optimize the workflow to ensure that both the computation and memory devices are engaged simultaneously, minimizing resource waste and maximizing system performance.

To validate our analysis, we develop and evaluate Lamina, a distributed heterogeneous LLM inference system with attention offloading. We also conduct extensive experiments to reveal the advantages of the attention offloading architecture. Experimental results on 13B and 33B models show that our system can achieve up to $1.48\times$ – $12.1\times$ higher throughput per cost than existing solutions.

2 Preliminaries

Generative AI services, like coding assistants and chatbots, rely on large language models (LLMs) to produce meaningful results. Given an input text (prompt) represented as a sequence of *tokens* (w_1, w_2, \dots, w_n) , an LLM predicts the probability distribution of the next token $P(w_{n+1}|w_1, w_2, \dots, w_n)$. To generate a continuous stream of text, this process is repeated iteratively. At each step, the LLM predicts the probabilities for the subsequent token, samples one based on these probabilities, and appends it to the growing text sequence. This approach, known as *autoregressive generation*, continues until an end-of-sequence (EOS) token is generated or the generated text reaches a predefined length limit.

Transformer-based LLMs. Most contemporary LLMs are based on the *transformers* [40] architecture. A transformer-based LLM first maps each input token to a word embedding of dimension d_{embd} . The word embeddings then go through

a sequence of transformer blocks. Finally, the outputs of the last transformer block are multiplied by a sampling matrix to obtain the predicted likelihoods of the next tokens.

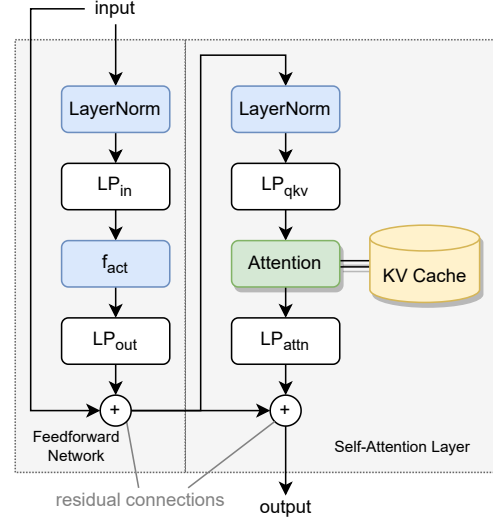


Figure 2: Computation graph of one LLM block.

Each transformer block in LLM (Figure 2) contains a feed-forward network and a self-attention layer. The feedforward network projects the input embeddings to an intermediate vector space by multiplying a matrix $W_{\text{fc}} \in \mathbb{R}^{d_{\text{interm}} \times d_{\text{embd}}}$. After applying an activation function, the intermediate vectors are then multiplied by another matrix $W_{\text{proj}} \in \mathbb{R}^{d_{\text{embd}} \times d_{\text{interm}}}$ to obtain the transformed embeddings:

$$x'_i = W_{\text{proj}} \cdot f_{\text{act}}(W_{\text{fc}} \cdot y_i). \quad (1)$$

The self-attention layer first projects the embeddings into query, key, and value vectors:

$$q_i = W_q x_i, \quad k_i = W_k x_i, \quad v_i = W_v x_i \quad (2)$$

Then, the *attention operator* is applied to the query, key, and value vectors of a sequence to obtain the attentions:

$$a_i = \sum_{j=1}^i \text{softmax}(q_i^\top k_j / \sqrt{d}) v_j \quad (3)$$

The attentions are then multiplied by a matrix W_{out} to obtain the output embeddings y_i of the attention layer:

$$y_i = W_{\text{out}} a_i. \quad (4)$$

In practice, multi-head attention is often used to enhance model quality, where the query, key, and value vectors are subdivided into independent *heads*. Each head undergoes the computation described in Equation 3. The results of the heads are then concatenated and multiplied by W_{out} .

KV cache. Recall that LLM runs in an autoregressive fashion. After each iteration, we may cache the values of k_i and v_i in all layers. Thus, in later iterations, we may only run the model on the latest token and use the cached value to compute the attentions in Equation 3. This optimization greatly reduces the computation complexity of LLM inference at the expense of larger memory requirements.

With the adoption of the KV cache, the LLM inference procedure can be divided into two distinct phases. The *prefill phase* processes the whole input prompt, computing the intermediate embeddings of each token in the prompt, as well as the first token to generate. The *generate phase* iteratively computes the embeddings of the last token and generates the next token.

Request Batching. In machine learning training and inference, the batching technique is widely used to increase GPU utilization [8, 12, 35]. By processing multiple inputs simultaneously, the model parameters loaded from GPU memory can be reused for different inputs, making the entire workload more computation-intensive. However, in the context of LLM serving, the effectiveness of batching is limited by several factors. First, LLM serving requests may come with different context lengths. An implementation that blindly pads all inputs to the longest length can lead to considerable waste of computation and memory resources. More sophisticated batching techniques [18, 43] have been proposed to address this problem.

Most importantly, with the introduction of KV cache, the attention operator remains bottlenecked at memory bandwidth, as each request must access its own specific KV cache. Furthermore, the size of the batches is severely limited by the memory capacity of GPUs, which can only accommodate a small number of KV caches.

Model parallelism. Due to the limited capability of a single accelerator, model parallelism has been developed to divide the model across multiple accelerators. Model parallelism is necessary for managing models that are excessively large to be stored in a single accelerator. Two popular forms of model parallelism are pipeline parallelism and tensor parallelism.

Pipeline Parallelism (PP) involves splitting the model’s layers into different stages and assigning each stage to a separate device. This allows for the concurrent processing of different parts of the model, with the output of one stage being fed as input to the next. Pipeline parallelism does not reduce the end-to-end computation time.

Tensor Parallelism (TP) divides individual parameter tensors within a model’s layers across several devices, enabling operators such as matrix multiplications to be computed jointly by multiple accelerators. While this significantly reduces computation time, tensor parallelism has inferior scalability compared to pipeline parallelism. This is due to the extensive data exchange required between accelerators for

every single layer. As a result, tensor parallelism is often limited to be deployed within a node, where accelerators are connected by high-speed links like NVLink.

3 Case Study: LLaMA-13B Serving

To comprehensively understand the constraints present in current homogeneous clusters, this section will analyze the LLaMA-13B model as a representative architecture to evaluate its performance characteristics. By integrating quantitative performance models with empirical evaluations, our goal is to illustrate the substantial benefits of deploying the model across two heterogeneous accelerator pools. This approach has the potential to significantly enhance throughput per dollar while maintaining latency at comparable levels. The specific notations used in this analysis are explained in Table 2.

Table 2: Notations used in the performance analysis. The values for LLaMA-13B are also presented.

Parameter	Description	Typical Value
N	Number of parameters in LLM.	13 billion
d	Dimension of the embeddings.	5120
L	Layers of the LLM.	40
e	Bytes per element.	2
B	Batch size.	$1 \sim 512$
l	Context length.	$128 \sim 4096$

3.1 Serving with a Single A100

In our earlier discussions, we highlighted that during the decoding phase, when a batch of B queries is processed, the computational tasks within each transformer block can be classified into two distinct operator types, each with significantly different arithmetic intensities. For linear transformation tasks, such as the multiplication of QKV, Out, and FFN weight matrices, the computation involves $2NB$ floating-point operations and also requires loading $e(N + Bd)$ data from global memory. This results in an arithmetic intensity of $\frac{2NB}{e(N + Bd)}$, which substantially increases with larger batch sizes. Figure 3 displays the measured results of these linear transformations executed on one A100 GPU at varying batch sizes, demonstrating a rapid increase in MFU up to a batch size of 2048, thereby highlighting the benefits of increasing batch size.

In contrast, the arithmetic intensity for the attention operator remains a relatively small constant. This is due to the attention operator’s execution is essentially a group of GEMV (General Matrix-Vector Multiplication) operations, with each query independently processing its own KV cache. Despite advancements in optimizations allowing state-of-the-art kernels to achieve high (>80%) model bandwidth utilization (MBU) for this operator, its MFU is generally limited to less than 5% in most scenarios.

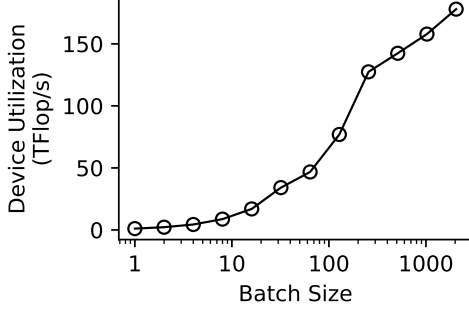


Figure 3: Computation utilization of A100 when serving LLaMA-13B with different batch size. The attention is excluded.

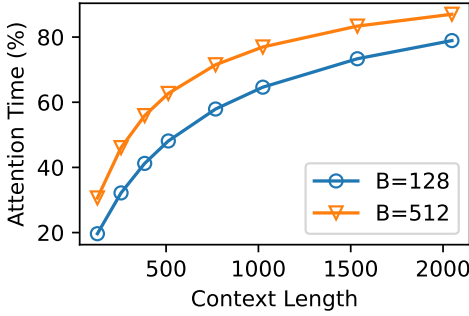


Figure 4: Percentage of time consumed by attention operators.

Furthermore, as the context length increases, the proportion of time spent on attention operations also rises, leading to a decrease in overall MFU during the decoding phase. Figure 4 illustrates the proportion of time consumed by attention operators across various context lengths, using a batch size of 128 and 512. It becomes evident that attention operations dominate (exceeding 80% of the time) when the context length surpasses 1500, a common scenario in real-world applications. Additionally, the memory capacity required for KV cache also increases linearly with the context length. Consequently, managing a large batch size becomes impractical in real-world scenarios due to memory shortage, further diminishing the overall MFU in practical applications.

3.2 Using Memory-Optimized Accelerators for Attention

While numerous cost-effective accelerators offer impressive memory bandwidth and capacity, solely relying on these consumer-grade GPUs for handling large models can be insufficient. The reason lies in their limited computing power and scalability, which restricts their ability to handle the computationally intensive tasks associated with inference. For instance, a single RTX 4080 GPU is capable of delivering merely 31.2% of the TFLOPs offered by A100. Also, as shown in Figure 5, using tensor parallelism with four RTX 4080 GPUs to serve

a 7-billion model can achieve only $1.41\times$ speedup against using only one GPU. For even larger models, the computation latency becomes a significant challenge, struggling to meet the QoS requirement even if more devices are used.

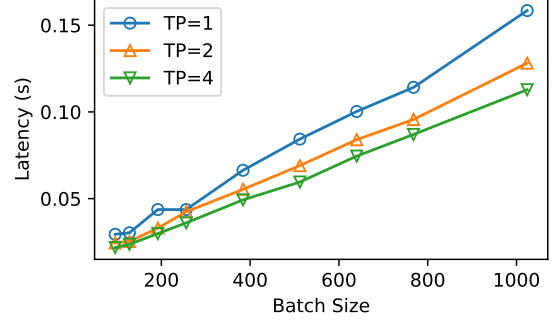


Figure 5: Scalability of RTX 4080 GPUs for LLaMA-7B model serving. The attention is excluded.

Nevertheless, the cost-effective memory in these devices presents a feasible approach to efficiently handle the attention operator. The attention computation is highly parallelizable, as each request and each head can be processed independently. As illustrated in Figure 3, running the attention operator with a small batch size can already saturate the entire device. Hence, it is possible to divide attention computation into smaller subtasks and distribute them across multiple cheap, memory-optimized devices without sacrificing bandwidth utilization.

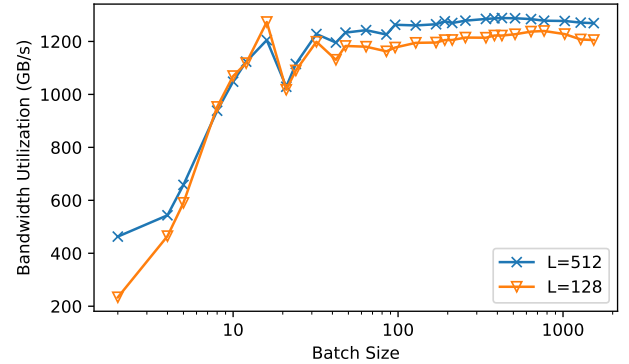


Figure 6: Bandwidth utilization of the attention operator with different batch sizes.

4 The Attention Offloading Architecture

To leverage the strengths of cheap memory-optimized accelerators, we propose an attention offloading architecture to effectively address the distinct characteristics of the two operators in LLM inference. This novel approach separates the processing of the attention operator from the overall model evaluation. Such an architectural division is strategically designed to optimize performance by selecting the most suitable

hardware tailored to the requirements of each operator. In particular, we employ memory-optimized devices for computing the attention operator and computation-optimized devices for the rest of the model.

4.1 Cost Efficiency

The disparity in arithmetic intensity between different types of operators within LLMs is a crucial factor influencing the overall efficiency of the model inference, and it forms the basis for our proposed approach in a heterogeneous accelerator environment. To analyze the cost efficiency of attention offloading, we propose pairing the compute-optimized accelerators (e.g., A100) with memory-optimized accelerators that offer $\alpha \times$ superior memory cost efficiency. Specifically, the hardware cost of a single A100 GPU could instead be used for a collection of memory-optimized accelerators, collectively achieving an aggregated memory capacity and bandwidth that is α times that of a single A100. As per Table 1, the value of α lies in $3.27 \times - 17.3 \times$ in real-world scenarios. Due to the additional cost of ancillary facilities and management, the actual cost efficiency for heterogeneous offloading is slightly higher than α . Hence, we conservatively take the minimum value of $\alpha = 3.27$ for analysis.

With the above definition, we calculate the optimal throughput per cost by enumerating the ratios of the two types of devices. The results presented in Figure 1 indicate a $2.1 \times - 2.8 \times$ theoretical cost efficiency advantage of our attention offloading architecture than homogeneous architectures when the context length is 512. This margin is even wider for longer context lengths due to the increasing proportion of the attention in the overall computation.

4.2 Surmounting the Communication Barriers

One potential obstacle in implementing attention offloading lies in the necessity of data transmission between heterogeneous accelerators, which could encounter the communication wall problem.

We conduct a quantitative analysis to determine the required interconnect bandwidth for such transfers. Say we run one iteration with batch size B . The minimum interconnect bandwidth required without slowing down the computation devices can be calculated as

$$\begin{aligned} \text{minimum bandwidth} &= \frac{\text{size of data to transmit}}{\text{computation time}} \\ &= \frac{4edBL}{2NB/\text{MFU}(B)} = \frac{2edL \cdot \text{MFU}(B)}{N} \end{aligned}$$

where $\text{MFU}(B)$ is the MFU at batch size B . The estimated minimum bandwidths required for different model inference configurations are calculated and presented in Table 3.

As evident from the data presented, the required interconnect bandwidth does not exceed 20GB/s, even when dealing

Table 3: Minimum interconnect bandwidth required in GB/s.

Batch Size	128	256	512	1024
LLaMA-13B @A100	4.90	8.12	9.08	10.06
LLaMA-33B @2×A100	5.94	9.31	11.56	12.53
LLaMA-65B @4×A100	7.55	11.78	13.92	15.90

with large models with batch sizes as high as 1024. This bandwidth demand can be easily met only by standard bus technologies, such as PCIe Gen 4, but also by networking solutions like 200G Ethernet. Indeed, contemporary data centers already fulfill this requirement, where each GPU is typically equipped with an exclusive 200Gbps NIC to provide sufficient networking bandwidth for training.

For memory devices, the identical interconnection bandwidth is also necessary to communicate with computational devices. Since we employ a collection of more economical yet less powerful memory devices to collaboratively compute attention, the communication bandwidth needed for each individual device is significantly smaller. Consequently, we can choose to either equip each device with a less powerful NIC or install a single shared 200Gbps NIC to serve multiple memory devices.

5 The Lamina System

To validate our theory, we craft Lamina, a distributed heterogeneous LLM inference system with attention offloading. Lamina is developed in Python with a few lines of C/C++ code. The LLM models are implemented in PyTorch [29], with the attention operator implemented by custom CUDA kernels. We use Ray [1] to manage the cluster and facilitate physical resource allocation and worker placement. Lamina supports both pipeline parallelism and model parallelism. Figure 7a depicts the overall architecture of Lamina.

Lamina employs two kinds of acceleration devices: memory devices are used for storing KV cache and computing the attention operator, and computation devices are used for storing model parameters and computing other parts of the model. These devices can either be co-located within the same physical machine or distributed across a cluster of nodes.

5.1 Execution Pipelining

Due to the serial nature of transformer-based models, if there is only one batch under processing, the memory device is idle when the computation device is working, and vice versa. To address this resource underutilization problem and increase system throughput, we may run multiple batches concurrently in a pipelined fashion. With properly designed pipelining, better hardware utilization can be achieved without sacrificing latency. We propose two pipelining schemes for this purpose.

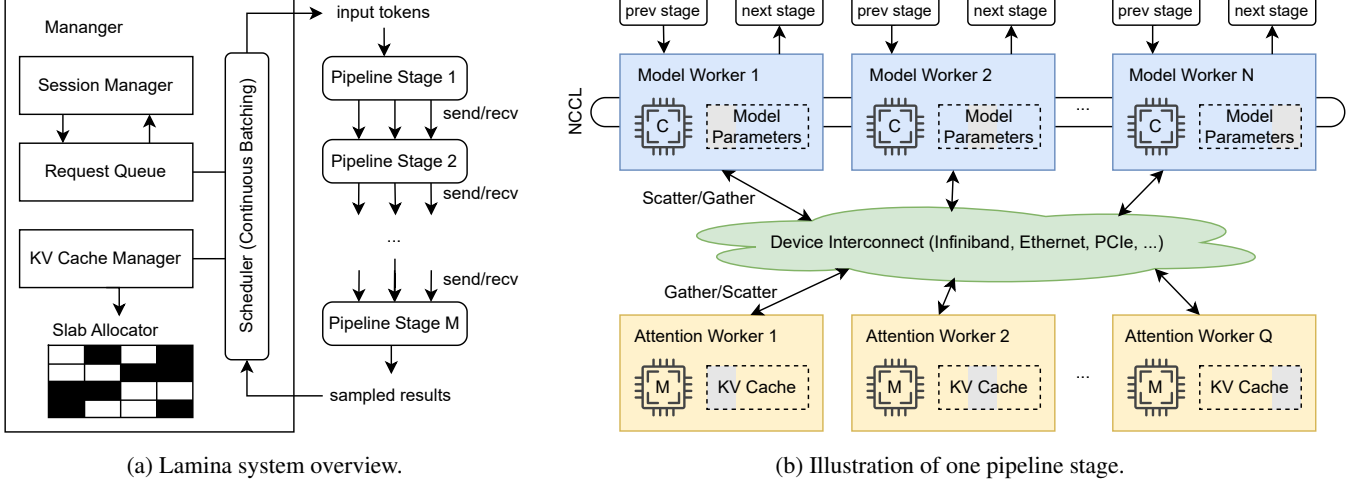


Figure 7: System architecture of Lamina.

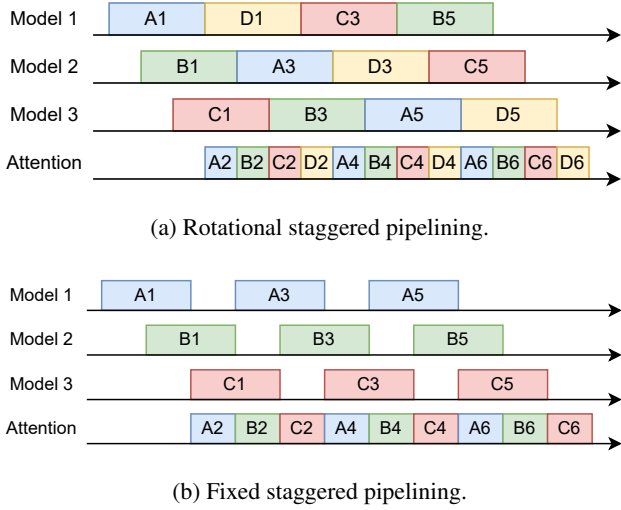


Figure 8: Execution pipelining schemes.

5.1.1 Rotational Staggered Pipelining

Assume that we execute n batches concurrently. Let t_m, t_a represent the time required for executing one model slice and one attention operator, respectively. As illustrated in Figure 8a, we deploy $n - 1$ model replicas, with each replica starting its tasks at a time of $\frac{t_m}{n-1}$ later than the previous one. All batches share a common set of memory devices to maximize aggregated memory bandwidth and improve memory utilization. For every batch, the KV cache is evenly partitioned across these devices. All memory devices jointly compute the attention operator for a single batch. The number of memory devices is selected to make $t_a = \frac{t_m}{n-1}$. After the attention operator, each batch transitions to the next model replica according to a rotational schedule; that is, the k th model slice of the j th batch is executed on replica $(j + k) \bmod (n - 1) + 1$.

This rotational task scheduling, combined with the staggered execution intervals, guarantees seamless task transitions for each batch and ensures a conflict- and bubble-free workflow on each device. Furthermore, by increasing the number of concurrent batches, the overall inference latency can be reduced due to the decreased attention computation time. However, the rotational scheduling requires migrating batch execution contexts between computation devices. Fortunately, these contexts are typically small enough to allow transmission during attention processing. Note that when $n = 2$, the context migration is unnecessary because both batches are executed within a single model replica.

5.1.2 Fixed Staggered Pipelining

We introduce an alternative pipelining approach with simpler task scheduling but slightly worse performance. Like rotational staggered pipelining, each model replica executes model slices with staggered execution intervals, and memory devices are also shared by all batches. However, as depicted in Figure 8b, each replica is assigned a fixed batch of requests and waits for the memory devices during attention computation. This fixed configuration simplifies execution control and allows better scalability at the expense of reduced throughput due to the emergence of bubbles in model devices.

5.2 Attention Parallelism

Due to the limited capability of a single device, we need to use multiple memory devices to jointly compute the attention operators. As depicted in Figure 9, the attention operators within one stage can also be parallelized among memory devices in various ways. One method is to distribute different requests across different devices; an alternative strategy is to partition and distribute the attention heads, which can

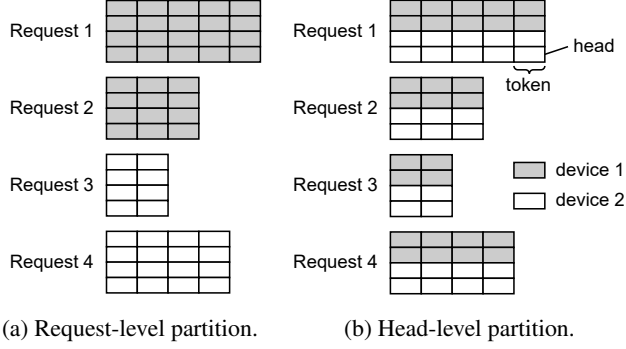


Figure 9: Work partition methods of the attention operator.

also be computed independently, to different devices. The head-level partitioning approach ensures a balanced workload distribution, whereas the request-level partitioning may result in load imbalance due to the differences in sequence lengths and therefore the KV cache sizes among requests. However, head-level partitioning has limited flexibility, as it requires the number of memory devices to be divisible by the number of attention heads. We opt for request-level partitioning in Lamina, which offers greater adaptability in accommodating various computational scenarios despite the potential for unequal workload distribution.

In the attention offloading architecture, achieving efficient data transmission between different devices poses a significant challenge. This challenge is primarily attributed to two factors. Firstly, the size of data that needs to be transmitted is enormous, which demands robust and high-bandwidth communication channels. Secondly, the stringent QoS requirements for interactive serving impose critical constraints on networking delay, often expecting microsecond-level latencies. We implement several key measures to optimize the networking, as described below.

5.2.1 Communication-Attention Overlapping

The overall inference latency can be reduced by simultaneously executing the communication and computation phases of the attention operator. A critical insight here is that the attention operator is *streamable*; that is, we may perform partial computation based on partial inputs, without waiting for complete input data. Given that the attention operator functions independently for each request and each head, we may stream the attention along either requests or heads.

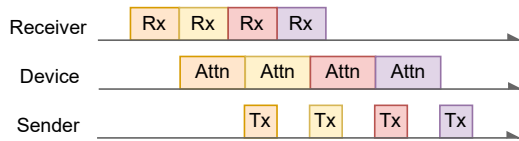


Figure 10: Communication-attention overlapping.

Figure 10 illustrates the process of streaming the attention operator within a single device. In this method, a batch for a device is divided into several mini-batches. These mini-batches are pipelined to hide the communication latency behind ongoing computations.

5.2.2 Reducing Networking Latency

With communication-attention overlapping, the network transmission time can be largely hidden behind the computation. However, the end-to-end round-trip latency of an ordinary networking implementation is still significant and requires further optimizations. This latency is influenced by various factors, including (1) the kernel networking stack processing, (2) the data transfer between host and device memory, and (3) the notification of message reception completion.

In Lamina, we use RDMA technology to transmit data by-passing the kernel networking stack. We further adopt GPUDirect RDMA to allow NICs to directly read or write GPU device memory without host memory transit. For completion notification, we observe a significant delay in starting subsequent computations in GPUs after the CPU receives work completions from NICs. This is primarily due to the kernel launch overhead, which involves copying kernel parameters to devices and notifying GPUs to start execution. To reduce this latency, we spawn the subsequent kernels in advance, making GPUs to be ready to execute the next task immediately. We use GPU-side busy polling to check for reception completion, saving additional synchronizations from the CPU side and further reducing the latency. By combining these optimizations, Lamina achieves a significant reduction in overall data transmission latency for attention offloading tasks.

5.3 Model Splitting

In the attention offloading architecture, different operators of the LLM might be executed on different hardware; hence, we need to partition the model into slices, which is achieved by cutting at the attention operators. It often involves significant modifications to the existing codebase, primarily because the desired cutting points do not align with the LLM’s inherent modular structure. This misalignment complicates the partitioning process and increases the risk of errors and inconsistencies within the attention offloading system.

To facilitate model partitioning, we develop an automated model splitter capable of transforming the LLM into individually invokable slices, illustrated in Figure 11. Given the LLM source code, the splitter uses symbolic execution to generate a weighted computation graph. The weight of each edge denotes the size of the data passed between the operators, which is derived from the model’s shape specification.

Due to the presence of residual connections and other intricate model constructs, directly removing the attention operator does not always result in a disconnected computation

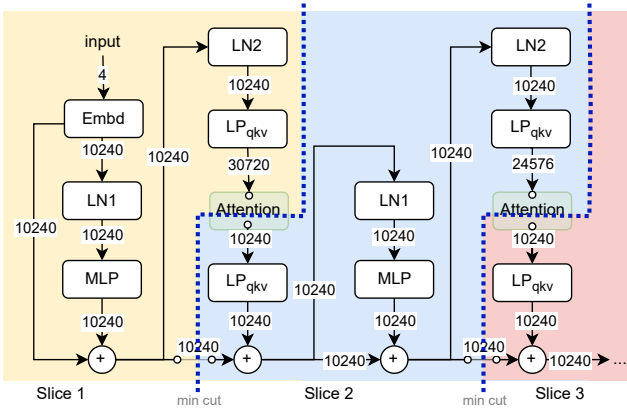


Figure 11: The partitioned computation graph of an LLM.

graph. Therefore, we compute the *minimum weighted cut* of the remaining graph, from the input to the output of the attention operator. The edges comprising this minimum cut, representing the context that must be saved across invocations, are removed from the computation graphs. This process is iteratively applied to each attention operator, ultimately yielding $n + 1$ model slices, where n denotes the number of the attention operators in the original computation graph.

5.4 KV Cache Management

The KV cache management of Lamina is similar to vLLM [18], except that Lamina allocates KV cache in token granularity. This helps further reduce the internal memory fragmentation compared to a page-granular allocator. By reference-counting the allocated tokens, Lamina can also support advanced decoding algorithms and scenarios, including beam search, parallel decoding, and shared prompt.

The Lamina manager is in charge of allocating and deallocating the KV cache. It employs a slab allocator and maintains the slab indices for the KV cache of each request. The slab allocator maintains the free list in LIFO order to improve cache utilization.

In contrast to vLLM which explicitly migrates KV cache between device memory and host memory, Lamina overcommits the KV cache space and leverages CUDA unified memory to transparently handle page swapping when the allocated KV cache size exceeds the device memory capacity.

5.5 Task Scheduling

As illustrated in Figure 7a, a centralized manager oversees the task scheduling and execution of Lamina. All incoming requests are first staged in the request queue. The scheduler employs *continuous batching* [43], where the requests are re-batched after each iteration. This allows new requests to join computation after waiting only a single iteration. Also,

padding are no longer required with proper kernel implementations.

By default, Lamina uses a FIFO scheduler. In each iteration, the scheduler picks the earliest requests until the batch size reaches a predetermined threshold. Completed requests (the EOS token is generated or the sequence length reaches the limit) are removed from the request queue after each iteration. Lamina also provides the flexibility to implement custom scheduling policies, enabling users to tailor the system to specific QoS requirements.

5.6 Handling the Prefill Phase

In Lamina, the prefill phase of the LLM inference is always carried out in computation devices and is not offloaded; only the KV cache is transferred to memory devices. It is observed that running requests in both the prefill and decode phases may lead to suboptimal overall performance. This is primarily due to the intensive computation involved in prefilling, which may create bubbles for decoding requests. To alleviate this problem, some research [2] suggests segmenting the prompt and running with generate phase in a piggybacking fashion. Another viable approach is to run the prefill phase on a separate system and copy the KV cache to the memory devices for prefilling [30].

6 Evaluation

Benchmark environment. Deploying Lamina in a true heterogeneous environment requires both high-end accelerators and many cheap memory-optimized devices with high-speed networking between them, which is not readily available in data centers. Hence, we simulate a heterogeneous environment with homogeneous NVIDIA A100-40G GPUs, where each GPU is equipped with a 200Gbps Infiniband NIC. We partition these GPUs into computation devices and memory devices. The memory devices are only used to store the KV cache and run the memory-intensive attention operator; all other computations are run on the computation devices. For the purpose of cost estimation, we take the cost efficiency of RTX 3090 GPUs for memory devices.

Models and workloads. Lamina supports a wide variety of LLM architectures, including OPT [44], LLaMA [38], and LLaMA2 [39]. All these architectures have similar outlines and workload characteristics and only have minor differences irrelevant to system designs. Hence, we choose two typical models, LLaMA-13B and LLaMA-65B, for evaluations. Both model parameters and KV caches are stored in BF16 format.

Baseline system. We compare with vLLM [18], a state-of-the-art LLM serving system optimized for high throughput.

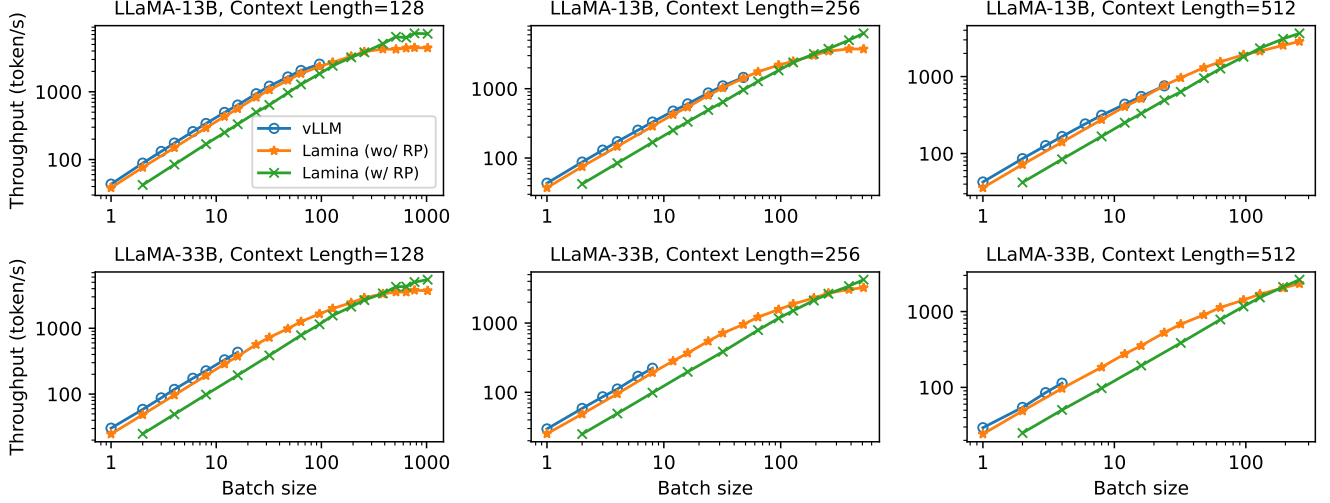


Figure 12: Token generation throughput on LLaMA-13B and LLaMA-33B.

Table 4: Large language models used for evaluation.

Model	Parameters	Layers	Hidden Size
LLaMA-13B	25.7 GB	40	5120
LLaMA-33B	64.6 GB	60	6656

vLLM also integrates optimizations from other LLM inference systems, such as continuous batching from Orca [43].

6.1 Throughput

We evaluate the system throughput using two distinct models and different context lengths. The result is presented in Figure 12. For smaller batch sizes, the throughput increases almost linearly as the batch size increases, which suggests a constant token generation latency by Little’s theorem. This latency is mainly attributed to the model parameter loading time in computation devices, which is irrelevant to batch sizes. For larger batches, the throughput increases slower because both computation and memory devices have to spend more time processing the incoming requests.

With rotational pipelining (RP) disabled, Lamina demonstrates comparable throughput to vLLM with the same batch sizes. This suggests that the additional overhead of separating the attention operator does not significantly impact system performance. The strength of Lamina becomes evident for larger batch sizes. By offloading the KV cache storage and attention computation to memory devices, Lamina can handle $10.7\times$ – $64.0\times$ larger batches than vLLM. This capability helps Lamina achieve better utilization in expensive computation-optimized accelerators. With maximum batch sizes, Lamina can offer $1.75\times$ – $20.5\times$ larger throughput than vLLM.

With rotational pipelining enabled, Lamina displays an ad-

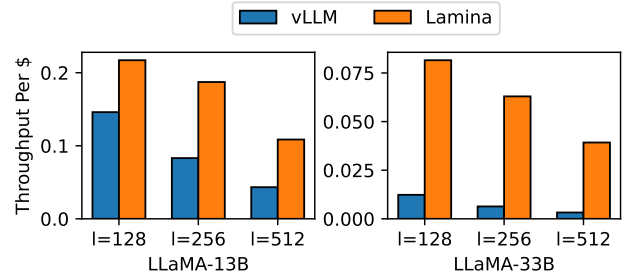


Figure 13: Throughput per hardware cost.

ditional $1.13\times$ – $1.63\times$ boost in throughput when the batch size is large (exceeding 128). By alternating the execution of multiple batches on computation and memory devices, we ensure that both kinds of devices are active, thereby minimizing resource wastage. For smaller batch sizes, rotational pipelining has much higher (nearly $2\times$) latency because the attention computation time is small compared with the model parameter loading time. Hence, we recommend enabling rotational pipelining only for sufficiently large batches to achieve throughput gains without suffering from latency penalties.

Figure 13 illustrates the cost efficiency of various models across different context lengths, measured by the maximum achievable throughput relative to hardware cost. Due to the capability to handle larger batches, Lamina demonstrates a significant economic advantage over vLLM, achieving an estimated $1.48\times$ – $12.1\times$ higher throughput per cost. This superiority of Lamina becomes particularly evident in longer contexts, where attention operators constitute a larger proportion of the total inference computation.

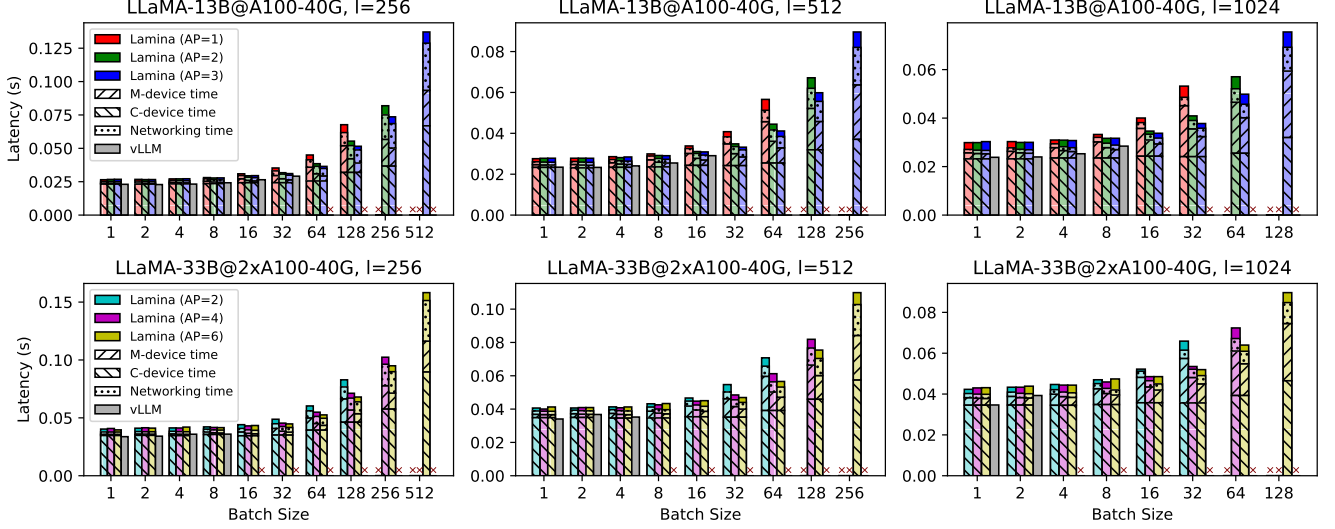


Figure 14: Token generation latency breakdown.

6.2 Latency

Latency is a crucial indicator of the service quality offered by LLM applications. In our evaluation of Lamina, we measure token generation latency across various system configurations and provide a detailed time breakdown illustrated in Figure 14. We disable rotational pipelining to better reveal the time breakdown.

For smaller batch sizes, it is observed that Lamina exhibits slightly higher latency compared to vLLM. This increase can be attributed to the additional data transfer and synchronization overhead between computation and memory devices. For larger batch sizes where vLLM runs out of memory, Lamina spends more time on networking. The attention computation time also increases with larger batch sizes as well as longer context lengths. However, the attention time can be decreased by adding more computation devices.

Despite the increase in token generation latency, it is important to note that Lamina offers significant advantages in terms of efficient resource utilization and cost-effectiveness by enabling larger batch sizes, making it a viable and attractive alternative to traditional LLM inference systems.

6.3 Networking Optimizations

Due to the notorious communication barriers, the efficient implementation and optimization of data transfer between computational and memory devices are paramount to the overall performance of the attention offloading system.

In Figure 15, we assess the impact of various measures aimed at enhancing networking latency between a single computation device and a memory device, starting from a vanilla RDMA implementation. For smaller batch sizes, the network-

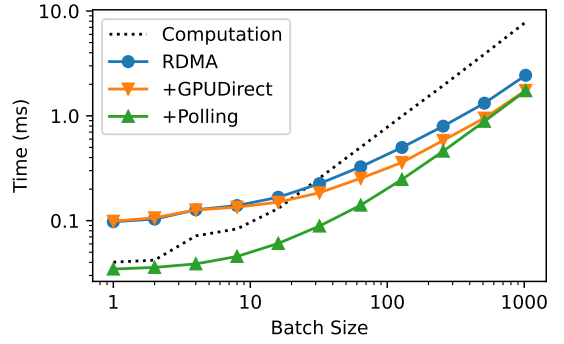


Figure 15: Networking overhead when serving LLaMA-13B. The attention computation time is also plotted for reference.

ing overhead is predominantly latency-bound. Without device-side polling, the end-to-end networking latency can soar to 100 μ s, accounting for over 70% of the total execution time of the attention operator. With device-side polling enabled, this latency is significantly reduced to 33 μ s, aligning with the system’s hardware limitations. As batch sizes increase, the communication time becomes constrained by network bandwidth. In such scenarios, using GPUDirect RDMA proves effective by elevating the networking bandwidth from 17.2 GB/s to 24.2 GB/s, achieving a 96.8% link speed utilization.

To evaluate the effectiveness of communication-attention overlapping, we measure the model execution time of LLaMA-13B configured with one computation device and three memory devices. The findings are presented in Figure 16. For smaller batch sizes, the benefits of communication-attention overlapping are not immediately apparent due to the predominance of round-trip latency in networking time. Nevertheless, as the batch size increases, this optimization

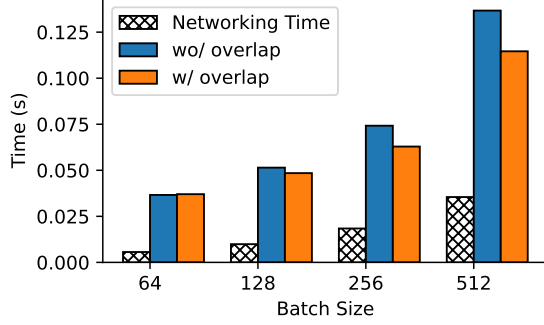


Figure 16: Impact of communication-attention overlapping.

successfully masks up to 62.2% of the networking time behind attention computation, leading to a reduction of 16.2% in the overall end-to-end latency.

6.4 Real-World Workload

We also evaluate the performance of Lamina in real-world serving scenarios with the ShareGPT [36] dataset on LLaMA-33B model. The latency-throughput relation, as well as the cost efficiency, is plotted in Figure 17. In our settings, all APs have similar cost efficiency; however, as the price of memory devices continues to drop, we expect that using memory devices will provide better cost-performance ratio in the future.

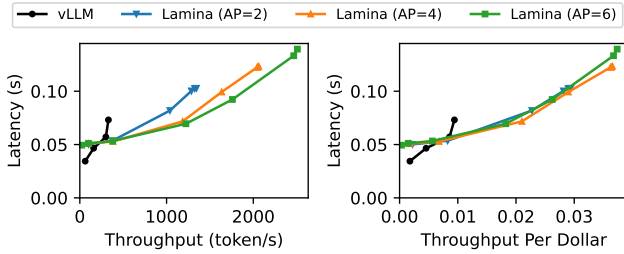


Figure 17: Latency-throughput relation and cost efficiency in real-world workloads.

7 Related Work

System optimizations for LLM Inference. Orca [43] proposes *continuous batching*, that batches incoming requests in iteration granularity. Compared with whole-request batching, continuous batching greatly reduces resource waste caused by early termination during the decode phase. PagedAttention [18] focuses on memory management optimizations, using fine-grained KV cache management to reduce memory waste. PagedAttention can also be used to optimize various decoding scenarios, like beam search and shared prefixes. These optimizations can all be used in our system. FlexGen [33] is a heterogeneous LLM inference system employing layer-

and token-level task partitioning and scheduling. However, it does not account for the varying characteristics of different operators within a layer. LLM-tailored inference systems, like DeepSeed [4], Megatron-LM [34], and TensorRT-LLM [27], use optimizations of various aspects including kernel optimization [9, 15], advanced scheduling [2, 10, 21, 30, 41], and efficient memory management [10].

Speculative Decoding The speculative decoding technology [20, 23, 25] enables parallel generation of multiple tokens for a single request during the decoding phase. This is done by *guessing* the next few tokens using a smaller auxiliary model. These predicted tokens are then validated by the primary LLM. This validation phase, similar to the prefill phase, can be executed in parallel, thereby enhancing the arithmetic intensity. Such parallel processing contributes to making the inference process less memory-bound and helps in reducing inference latency. However, speculative decoding can lead to a trade-off in throughput due to the auxiliary model’s overhead and the potential need for re-execution in case of misprediction.

Variations of the Attention Operator. Researchers have developed many variations of the attention operator for large language models to mitigate the memory bottleneck. Model quantization uses reduced-precision formats (e.g., FP8) to store KV caches. The grouped-query attention [3], used in LLaMA2-70B [39], shares the key-value pairs across multiple heads, effectively reducing the KV cache size. Additionally, various sparse attention mechanisms [6, 7, 17, 22, 24, 31, 32, 42] have been adopted, focusing on a subset of all history key-value pairs during attention computation. This approach not only conserves memory but also scales more effectively for longer contexts. H₂O [45] dynamically identifies and evicts less significant key-value pairs to limit the KV cache size. AttMemo [11] uses selective memoization and vector database to reduce the bandwidth requirement during LLM inference. All these modifications to the attention operator, however, might compromise the model quality.

8 Conclusion

In this paper, we introduce attention offloading, an innovative architectural approach to improve the cost efficiency of LLM inference. This approach is motivated by the observation that LLM inference computation can be divided into computation-intensive parts and memory-intensive parts (i.e., the attention operators). Thanks to the divisibility of the attention operator and the manageable bandwidth requirement, we successfully move the attention computation from the high-end accelerators to a pool of cheaper memory devices, improving the resource utilization of the costly high-end accelerators, and providing $1.48\times$ – $12.1\times$ higher estimated throughput per dollar than heterogeneous solutions.

References

- [1] Ray. <https://www.ray.io/>.
- [2] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills, 2023.
- [3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023.
- [4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale, 2022.
- [5] Kazi Asifuzzaman, Narasinga Rao Miniskar, Aaron R Young, Frank Liu, and Jeffrey S Vetter. A survey on processing-in-memory techniques: Advances and challenges. *Memories-Materials, Devices, Circuits and Systems*, 4:100022, 2023.
- [6] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [8] Y. Choi, Y. Kim, and M. Rhu. Lazy batching: An sla-aware batching system for cloud machine learning inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 493–506, Los Alamitos, CA, USA, mar 2021. IEEE Computer Society.
- [9] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. <https://crfm.stanford.edu/2023/10/12/flashdecoding.html>.
- [10] Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. Turbotransformers: An efficient gpu serving system for transformer models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP '21*, page 389–402, New York, NY, USA, 2021. Association for Computing Machinery.
- [11] Yuan Feng, Hyeran Jeon, Filip Blagojevic, Cyril Guyot, Qing Li, and Dong Li. Attmemo : Accelerating transformers with memoization on big memory systems, 2023.
- [12] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. Low latency rnn inference with cellular batching. In *Proceedings of the Thirteenth EuroSys Conference, EuroSys '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [13] GitHub. GitHub Copilot. <https://github.com/features/copilot>.
- [14] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. Newton: A dram-maker’s accelerator-in-memory (aim) architecture for machine learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 372–385, 2020.
- [15] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Hanyu Dong, and Yu Wang. Flashdecoding++: Faster large language model inference on gpus, 2023.
- [16] Jin Hyun Kim, Shin-Haeng Kang, Sukhan Lee, Hyeonsu Kim, Yuhwan Ro, Seungwon Lee, David Wang, Jihyun Choi, Jinin So, YeonGon Cho, JoonHo Song, Jeonghyeon Cho, Kyomin Sohn, and Nam Sung Kim. Aquabolt-xl hbm2-pim, lpddr5-pim with in-memory processing, and axdim with acceleration buffer. *IEEE Micro*, 42(3):20–30, 2022.
- [17] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
- [18] 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 page-dattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSPP '23*, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.
- [19] Yongkee Kwon, Kornijuk Vladimir, Nahsung Kim, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Guhyun Kim, Byeongju An, Jeongbin Kim, Jaewook Lee, Ilkon Kim, Jaehan Park, Chanwook Park, Yosub Song, Byeongsu Yang, Hyungdeok Lee, Seho Kim, Daehan Kwon, Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Myeongjun Lee, Minyoung Shin, Minhwan Shin, Jaekyung Cha, Changson Jung, Kijoon Chang, Chunseok Jeong, Euicheol Lim, Il Park, Junhyun Chun, and Sk Hynix. System architecture and software stack for gddr6-aim. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–25, 2022.

- [20] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.
- [21] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, 2023.
- [22] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023.
- [23] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding, 2023.
- [24] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.
- [25] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative large language model serving with speculative inference and token tree verification, 2023.
- [26] Microsoft. Bing ai. <https://chat.bing.com/>.
- [27] NVIDIA. Tensorrt-llm: A tensorrt toolbox for optimized large language model inference. <https://github.com/NVIDIA/TensorRT-LLM>.
- [28] OpenAI. Chatgpt. <https://chat.openai.com/>.
- [29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [30] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Aashaka Shah, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting, 2023.
- [31] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise self-attention for long document understanding. *arXiv preprint arXiv:1911.02972*, 2019.
- [32] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.
- [33] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning, ICML’23*. JMLR.org, 2023.
- [34] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism, 2020.
- [35] Franyell Silfa, Jose Maria Arnau, and Antonio González. E-batch: Energy-efficient and high-throughput rnn batching. *ACM Trans. Archit. Code Optim.*, 19(1), jan 2022.
- [36] ShareGPT Team. Sharegpt: Share your wildest chatgpt conversations with one click. <https://sharegpt.com/>.
- [37] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bis-syandé. Is chatgpt the ultimate programming assistant – how far is it?, 2023.
- [38] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.
- [39] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton,

Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

- [40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.
- [41] Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models, 2023.
- [42] Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. Bp-transformer: Modelling long-range context via binary partitioning. *arXiv preprint arXiv:1911.04070*, 2019.
- [43] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.
- [44] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.
- [45] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *arXiv preprint arXiv:2306.14048*, 2023.