ShadowServe: Interference-Free KV Cache Fetching for Distributed Prefix Caching

Xingyu Xiang[#]   Raj Joshi[#]   Yuhan Liu[†]   Jiayi Yao[†]   Chenxingyu Zhao[‡]

Junchen Jiang[†]   Yang Zhou[¶]   Eddie Kohler[#]   Minlan Yu[#]

[#]*Harvard University*   [†]*University of Chicago*   [‡]*University of Washington*   [¶]*UC Davis*

Abstract

Distributed prefix caching accelerates long-context LLM serving by reusing KV cache entries for common context prefixes. However, KV cache fetches can become a bottleneck when network bandwidth is limited. Compression mitigates the bandwidth issue, but can degrade overall performance when decompression interferes with model computation.

We present ShadowServe, the first SmartNIC-accelerated, *interference-free* prefix caching system for LLM serving. ShadowServe separates a control plane on the host and a data plane fully offloaded to the SmartNIC, which eliminates interference to both host GPU and CPU. To overcome the SmartNIC's limited compute and memory resources, we design a *chunked pipeline* that parallelizes data plane operations across the SmartNIC's compute resources, and a *minimal-copy memory management* scheme that reduces memory pressure on the SmartNIC. Compared to state-of-the-art solutions, ShadowServe achieves up to 2.2× lower loaded time-per-output-token (TPOT), and reduces time-to-first-token (TTFT) by up to 1.38× in low-bandwidth scenarios ($\leq 20\,\text{Gbps}$), translating to up to 1.35× higher throughput.

1 Introduction

Large Language Model (LLM) serving increasingly relies on longer contexts [10, 11, 21, 91, 100, 107] to improve generation quality and coherence, both in the form of longer input prompts [40] and multi-turn interactions [54]. These long contexts must be processed by the LLMs into intermediate states called the *KV cache* before output generation begins. A key technique to reduce the high computational cost of this pre-processing step is *prefix caching*, which can reuse precomputed KV cache data when different requests share prefixes in their contexts (e.g., system prompts [1]).

Prefix caches now commonly scale to distributed storage systems [18, 22, 62, 89, 106]. There are three primary reasons for this shift. First, with production services generating petabytes of data daily, reusable KV cache entries far exceed the capacity of local GPU or CPU memory [89]. Second, the lifespan of prefix cache entries can be long and unpredictable [101], so high cache hit rates require large storage [62]. Third, a distributed system allows KV cache entries to be shared across an entire cluster, allowing requests with
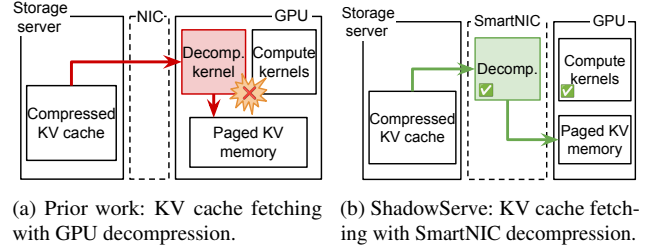

(a) Prior work: KV cache fetching with GPU decompression.

(b) ShadowServe: KV cache fetching with SmartNIC decompression.

Figure 1: Solutions for distributed prefix caching.

identical prefixes to be routed to different nodes for load balancing [37, 89, 96].

Distributed prefix caching is only beneficial when fetching the KV cache is faster than recomputing it. However, many LLM serving settings operate in relatively low-bandwidth environments, such as public cloud instances with capped network speeds [43–45, 62, 67, 70], low-end GPUs that are typically paired with lower bandwidth [14, 32, 43, 44, 69], and network-attached storage systems [3, 79]. In these environments, the network can become a bottleneck.

Recent work [62] observes that transferring *compressed*, rather than raw, KV caches can mitigate this bottleneck and improve the performance of prefix caching. KV caches are stored and transmitted in compressed format; serving nodes must use their GPUs to decompress KV caches before using them (Figure 1a). Unfortunately, this introduces a new problem: *interference with model computation*. We find that concurrently running decompression and LLM serving on the same GPU causes both tasks to slow down, often by $\geq 30\%$ (§2.2). Offloading decompression to the host CPUs seems like a feasible workaround, but host CPUs in serving clusters are already burdened with other tasks like vector search and data preprocessing [31, 42, 58, 66, 72, 120], and they are inefficient at a variety of decompression algorithms (§2.3).

In this work, we present a new approach. ShadowServe (Figure 1b) uses SmartNICs, rather than GPUs or CPUs, to fetch and decompress KV caches. SmartNICs are system-on-chip (SoC) devices with computation cores isolated from the host, integrating hardware accelerators for decompression. ShadowServe separates the prefix caching control plane from a SmartNIC-only data plane. The control plane, integrated into the LLM serving scheduler, initiates background fetches of KV caches as required. Meanwhile, the data plane handles

all data-intensive operations, including decompression; since it runs on the SmartNIC, the host GPU is free to focus entirely on model computation.

However, using SmartNICs is not straightforward, as they are usually equipped with wimpy general-purpose on-chip cores and constrained memory subsystems. Naively offloading the entire data plane can create a new performance bottleneck. We overcome these limitations with a *chunked pipeline* for the SmartNIC data plane. Fetching is divided into four distinct stages: network fetching, lossless decompression, dequantization, and DMA to the GPU. Instead of processing an entire KV cache sequentially, we split it into fixed-size chunks that flow through these stages in a pipelined manner. This maximizes hardware utilization and improves operation throughput. To avoid resource contention, we use performance measurements to optimally partition the SmartNIC's compute resources, including the on-chip cores and dedicated accelerators, among these tasks. In addition, we design a *minimal-copy memory management* mechanism that eliminates redundant data copies and reduces memory access stalls; it works by preallocating and pinning all required memory buffers both on the SmartNIC and in GPU memory. Contention is avoided by giving each concurrent chunk in the pipeline its own dedicated memory region.

We implement a prototype of ShadowServe with the NVIDIA BlueField-3 DPU, a state-of-the-art SmartNIC, and evaluate it across a wide range of network bandwidth and output length settings. Compared to state-of-the-art GPU-decompression solutions, ShadowServe consistently achieves up to $2.2\times$ lower loaded time-per-output-token (TPOT) across all settings, and reduces time-to-first-token (TTFT) by up to $1.38\times$ in low-bandwidth scenarios (≤ 20 Gbps). These advantages combined translate to up to $1.35\times$ higher throughput. Our analysis also reveals that ShadowServe's performance is limited in high-bandwidth settings due to the SmartNIC's memory subsystem, identifying a key area for future hardware improvements.

We note that ShadowServe does not propose a new KV cache compression algorithm, but offloads existing ones from the host GPU/CPU to the SmartNIC. Therefore, we do not change any numerical property (e.g., accuracy or compression ratio) of the compression algorithms. ShadowServe's design is general to any compression algorithm that can be efficiently mapped to the hardware resources on the SmartNIC.

In summary, we make the following contributions.
- We identify and characterize severe bidirectional interference between KV cache decompression and LLM model computation on the GPU.
- We present ShadowServe, the first prefix caching system to offload KV cache fetching and decompression to SmartNICs, with a clean control/data plane separation that eliminates host-side interference.
- We design a chunked pipeline and a minimal-copy memory management scheme to overcome the compute and memory
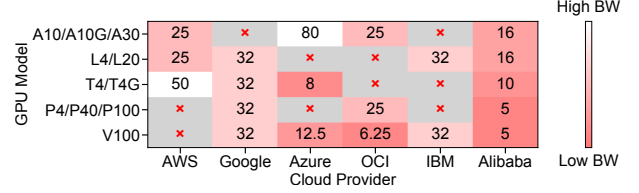
Figure 2: Maximum per-GPU network bandwidth (Gbps) for low-end GPU instances in major clouds. × = unsupported GPU model.

GPU Model	AWS	Google	Azure	OCI	IBM	Alibaba
A10/A10G/A30	25	×	80	25	×	16
L4/L20	25	32	×	×	32	16
T4/T4G	50	32	8	×	×	10
P4/P40/P100	×	32	×	25	×	5
V100	×	32	12.5	6.25	32	5

limitations of current SmartNICs and maximize data plane throughput.
- We implement a prototype of ShadowServe, demonstrate that it outperforms state-of-the-art systems in terms of both latency and throughput in most settings, and analyze ShadowServe's bottleneck for the settings where it lags behind.
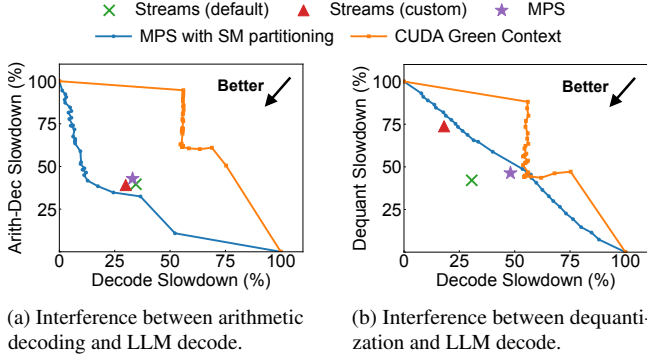
2 Background and Motivation

2.1 Prefix Caching

LLM serving requests go through two phases, *prefill* and *decode*. In the prefill phase, the input prompt is processed to produce the intermediate states called the KV cache. Then, the decode phase utilizes the KV cache to autoregressively generate output tokens. Prefills are both resource- and time-consuming with complexity quadratic to the prompt length. Prefix caching [6, 23, 28, 84, 123] helps requests skip the expensive prefill phase by reusing previously computed KV caches for repeated prompt prefixes.

Many existing systems [28, 51, 52, 114, 116, 122, 123] cache KV data in local GPU/CPU memory, and some extend to local disk [16, 27, 46, 47, 112], but on production-level services, prefix caching is expanding to distributed storage [18, 22, 62, 79, 89]. The expanded scale of remote storage supports the larger caches common on today's models—e.g., Llama-34B generates 19 GB KV cache for a single 80K-token document [62], and production systems can generate PB-level KV caches [89]—and allows KV caches to be held for longer intervals [62]. Distributed storage also enables cross-node cache sharing, allowing requests with the same prefix to be routed to different machines for load balancing [37, 89, 96].

Effective distributed prefix caching might seem to require high bandwidth. For example, Mooncake [89] reports a sharp increase in TTFT and evident network congestion when inter-node bandwidth falls below 100 Gbps; faster GPUs would push this breakeven bandwidth even higher. Nevertheless, in some real-world scenarios, KV caches need to be fetched over limited bandwidth. As serving extends to public cloud environments [43–45, 67, 70], KV cache must be transmitted over cloud networks. Unlike specialized datacenter interconnects, cloud platforms typically cap inter-node bandwidth [4, 29, 39, 85]. Low-end GPUs, nowadays a trendy choice for LLM serving [14, 32, 43, 44, 69], are provided with even lower bandwidth (Figure 2). Some works even transmit KV cache via the Internet over single-digit Gbps [18, 62]. In

(a) Interference between arithmetic decoding and LLM decode.

(b) Interference between dequantization and LLM decode.

Figure 3: Decompression and LLM decode interfere on GPU. No interference would correspond to 0% slowdown for either task; we observe significant slowdowns across all GPU multitasking mechanisms. For CUDA streams, we launch LLM decode either in the default stream, or in a custom stream like decompression. MPS [80] and Green Context [76] enable SM partitioning, and the two curves are plotted by assigning different numbers of SMs to the two tasks. Green Context performs poorly as it is an experimental feature and fails to partition memory bandwidth effectively [109].

Algorithm	CPU (single core)	BF3 accelerator
Deflate	2.5	276.5
LZ4	18.6	246.3

Table 1: Decompression output throughput (Gbps) for Deflate and LZ4, on 1 host CPU core and NVIDIA BlueField-3 DPU's decompression accelerator. We use the same KV cache data used in the final evaluation as input.

addition, fetching KV cache from remote storage [3, 79] further limits bandwidth due to disk access. In the AWS cloud, a GPU instance's access to network-attached storage like AWS Elastic Block Store can be limited to as low as 19 Gbps [4].

2.2 KV Cache Compression

CacheGen [62] (now part of LMCache [65]) shows that transmitting *compressed* KV caches can significantly reduce bandwidth requirements, thus making distributed prefix caching valuable at lower network bandwidths. The KV cache is stored and fetched in a compressed form; it is only decompressed at the serving node where it is needed.

We note that this *transmission-oriented* compression approach is distinct from *runtime* KV cache compression, which aims to reduce the GPU memory footprint. Transmission-oriented KV cache compression allows for more aggressive compression algorithms because it need not maintain a tensor format that the GPU can use directly. Therefore, powerful lossless compression like arithmetic coding [62, 103] is used to further reduce the data transfer size beyond runtime methods like pruning [41, 63, 119] and quantization [36, 48, 64].

CacheGen delegates decompression to the GPU (Figure 1a). The compressed KV cache data is fetched to the GPU, decompressed using customized GPU kernels, and then stored
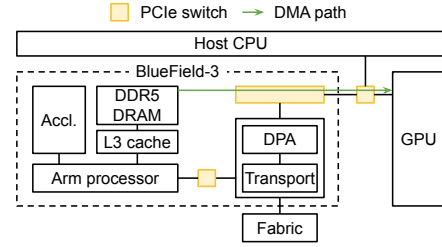
Figure 4: Architecture of the NVIDIA BlueField-3 SmartNIC. It allows peer-to-peer DMA between the on-chip DRAM and GPU memory without host intervention.

into the GPU's KV cache memory. Unfortunately, when decompression runs concurrently with model computation on the GPU, we observe severe bidirectional interference: both model serving performance and decompression throughput degrade substantially.

To better characterize this interference, we run experiments that collocate decompression and LLM decode on an NVIDIA L40S GPU with different GPU multitasking mechanisms, and measure their respective slowdown (the throughput drop compared with running each operation on the entire GPU alone). We test both lossy and lossless compression mechanisms as used in CacheGen [62]. The LLM decode has a context length of 30K; a minimal batch size (1 token) is used as it yields the least possible interference in long-context serving.

Figure 3 shows the results. Minimizing the slowdown of one task always leads to a drastic performance degradation of the other, regardless of the multitasking mechanism. For arithmetic decoding (Figure 3a), it is not possible to limit both tasks' slowdown below 30% at the same time. For dequantization (Figure 3b), the best mechanism still results in more than 25% slowdown for both tasks. This fundamental bottleneck motivates moving decompression off the GPU.

2.3 Opportunities with SmartNICs

Our key idea is to offload KV cache fetching and decompression from GPU to a programmable I/O device, specifically a SmartNIC (Figure 1b). This cleanly separates KV cache preparation and model computation by putting them on different hardware, thereby eliminating their interference. For example, Figure 4 shows the architecture of the NVIDIA BlueField-3 DPU [74] used in our system. It has a 16-core Arm Cortex-A78AE processor and 32GB DDR5 memory. It supports peer-to-peer DMA between the on-chip DRAM and the GPU memory, and it has a lossless decompression accelerator for both Deflate [25] and LZ4 [113] algorithms directly accessible by the Arm processor.

The SmartNIC offers several advantages for fetching and decompression. It avoids using the GPU, which should spend as much time as possible on model serving. It avoids using the CPU, which is often burdened with auxiliary tasks (e.g., vector search in Retrieval-Augmented Generation [42, 58, 120] or data preprocessing for multimodal models [31, 66, 72]), and has low decompression throughput per core (Table 1).
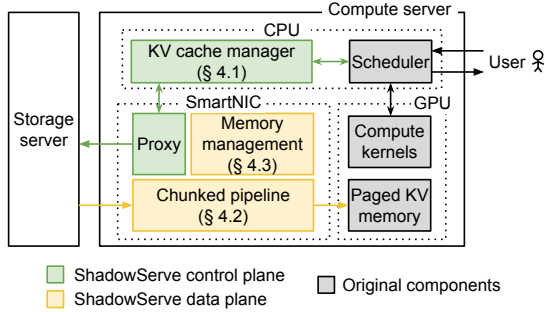
Figure 5: ShadowServe overview.

Data transfer overheads can be localized: on-chip DMA engines that facilitate peer-to-peer DMA between the SmartNIC and the GPU enable fetching directly into the GPU memory transparent to the host. Finally, SmartNICs yield superior KV cache fetching and decompression performance. SmartNICs are usually equipped with on-chip general-purpose CPU cores and dedicated lossless decompression accelerators that enable high-throughput processing on the KV cache data.

3 ShadowServe Overview

We present ShadowServe, a SmartNIC-accelerated prefix caching system that supports *interference-free* KV cache fetching. At the core of ShadowServe is a clean separation between the *control plane* and the *data plane* of prefix caching. The control plane runs on the host CPU and schedules KV cache fetches, and the data plane, fully offloaded to the SmartNIC, handles individual network I/O and decompression operations. By offloading only the data operations to the SmartNIC, ShadowServe eliminates host-side interference while keeping the SmartNIC stack simple to implement.

Fully unleashing the benefits of SmartNIC offloading requires addressing two challenges on the SmartNIC: limited compute and memory resources. On the compute side, SmartNICs integrate general-purpose on-chip cores that are significantly less powerful than standard server CPUs, and performing data-intensive operations naively on these cores can create a new performance bottleneck. On the memory side, the SmartNIC's memory subsystem is also restricted. For example, the BlueField-3 has a small cache hierarchy of 1 MB, 8 MB, and 16 MB L1, L2, and L3 caches [74], which must serve all pipelined operations. Although the on-chip DDR5 DRAM provides a high bandwidth of > 80 GB/s, there are only two memory channels. For comparison, a typical host memory system provides eight memory channels per CPU socket. Meanwhile, most of the offloaded data operations are memory intensive.

To address the compute challenge and maximize throughput on the SmartNIC, we design the data plane as a *chunked pipeline* (§4.2). This design breaks the end-to-end fetching process into distinct stages, and assigns them to different SmartNIC resources (e.g., CPU cores and hardware accelerators), to enable maximum parallelism and resource utilization.

To overcome the memory challenge, we design a *minimal-copy memory management* scheme (§4.3) for the data plane. This mechanism carefully manages on-chip memory buffers to ensure a smooth flow of data through the pipeline, avoiding expensive memory registration and data copies between stages.

Figure 5 provides an overview of the ShadowServe architecture, built upon the separation of control and data planes. ShadowServe integrates well with existing serving framework components, including a scheduler that accepts user requests arriving at the system and dispatches compute kernels to the GPU, and a paged KV memory region in the GPU memory. The core prefix caching functionality is implemented in our two planes.

Control plane. The control plane manages the orchestration of KV cache fetches. It enables fully *asynchronous* fetching that moves KV cache fetching off the critical path (§4.1) and is composed of two major components.
KV cache manager. The KV cache manager runs on the host CPU. It communicates with the scheduler each iteration to identify requests eligible for KV cache fetching. It temporarily takes ownership of these requests and sends their metadata to the SmartNIC proxy for fetching. Once the data plane has fetched the KV cache, the manager is notified, and it seamlessly submits the ready requests back to the scheduler to begin token generation. This coordination occurs in the background, transparent to the serving scheduler (§4.1).
SmartNIC proxy. The proxy runs on the SmartNIC on-chip cores and communicates with the KV cache manager via a communication channel. Upon receiving fetch requests, it communicates request metadata with the remote storage server and instructs the data plane to begin the fetching process. It oversees the entire operation on the SmartNIC and notifies the KV cache manager upon successful completion, i.e., when the KV cache is ready in the GPU memory.

Data plane. The data plane resides entirely on the SmartNIC and is responsible for efficiently retrieving, decompressing, and transferring KV cache data from the remote storage server to GPU memory. The data plane's key components are the *chunked pipeline* (§4.2), which maximizes resource utilization, and our *minimal-copy memory management* mechanism (§4.3), which alleviates the on-chip memory bottleneck. By offloading the entire data path from the host, ShadowServe eliminates GPU interference while achieving high throughput.

While our discussion focuses on a single GPU-SmartNIC pair, ShadowServe's design applies to multi-GPU serving environments that leverage tensor and pipeline parallelism [56, 73]. This extension requires pairing each GPU with a dedicated SmartNIC, where each SmartNIC's data plane independently fetches the KV cache portion for its associated GPU. We believe this one-to-one architecture is practical, as datacenter machines often equip each GPU with a dedicated high-performance NIC already [12, 30, 75, 99].
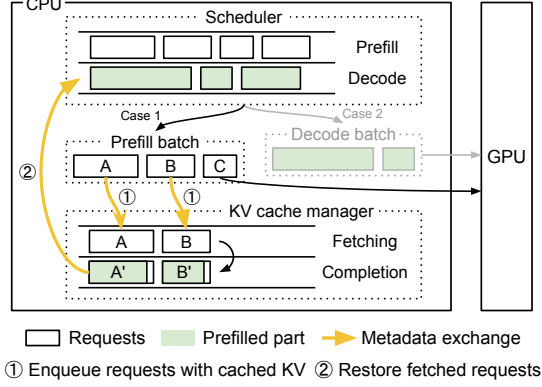
Figure 6: ShadowServe's asynchronous fetching.

4 ShadowServe Design

4.1 Asynchronous Fetching

Asynchronous fetching is essential for realizing the full benefits of prefix caching in a system with remote storage. Fetching a request's KV cache from a storage server introduces I/O latency that, if handled synchronously, would stall the GPU and severely limit system throughput. The primary goal of our asynchronous design is to decouple this I/O-bound fetching process from the main execution flow. By fetching KV caches for some requests in the background while the GPU processes others, we can hide the I/O latency, maximize hardware utilization, and maintain high serving throughput.

To achieve asynchronous fetching, we design a *KV cache manager* running in parallel with the serving scheduler in the host CPU. The manager interacts with the scheduler, transparently moving eligible requests out of the execution flow and fetching their KV cache in the background.

The KV cache manager keeps track of requests undergoing KV cache fetching by maintaining two internal queues, *fetching* and *completion*, shown in Figure 6. The *fetching* queue contains requests eligible for fetching, and the *completion* queue holds requests that have completed the fetching process and are ready to resume execution. In our current design, we use simple FIFO queues; the manager processes fetch jobs serially. However, because the number of tokens in each fetch job is known, the fetching times can be estimated, opening the door to more advanced scheduling (e.g., Shortest Job First), which we leave as future work.

Batch interception. The KV cache manager identifies and moves eligible requests to the fetching queue without interrupting the scheduler. It achieves this with a non-intrusive *batch interception* mechanism that operates in lockstep with the scheduler. In each iteration that the scheduler produces a prefill batch, the manager performs a two-way exchange. First, it intercepts the batch and strips out any requests eligible for remote KV cache fetching, moving them to its fetching queue to be processed in the background (① in Figure 6). Note that the KV cache manager only intercepts prefill batches,

as decode requests already have their KV cache computed. Simultaneously, it checks its completion queue for any requests that have finished fetching from a previous iteration and restores them to the scheduler for execution (② in Figure 6). These two operations are atomic from the scheduler's perspective. The scheduler then proceeds with its (now modified) batch, unblocked by the KV cache fetches happening asynchronously.

For example, in Figure 6, the scheduler either produces a prefill batch (case 1) or a decode batch (case 2) for the current iteration. For case 1, the KV cache manager examines the prefill batch, consisting of requests A, B, and C, to see if any of them are eligible for KV cache reuse. Here, request C's KV cache is not stored in the storage server, so its metadata is parsed and sent to GPU as usual. For requests A and B, their KV cache is stored, so they are temporarily removed from the current batch and the scheduler, and added to the manager's fetching queue. Meanwhile, any requests in the completion queue are restored to the scheduler.

A natural question is why not begin fetching KV cache earlier, e.g., when the requests *arrive*, instead of when they *are scheduled*. The reason is that for every new request, its entire prompt's KV cache space in the GPU is allocated on demand when it is scheduled for the first time as a prefill request, and we can only begin fetching after this GPU memory allocation. This lazy allocation strategy aims to lower GPU memory consumption. Without it, a burst of requests could easily exhaust GPU memory.

Background fetching. After identifying eligible requests and moving them to the fetching queue, the KV cache manager needs to tell the data plane to fetch the KV cache for these requests in the background. It does so by running a *background fetching loop*. Each time, it retrieves a request from the fetching queue, fetches the request's KV cache into the paged memory via the data plane, and puts the request into the completion queue. In Figure 6, requests A and B are submitted to the data plane, and put to the completion queue after fetching. Again, this process is transparent to the scheduler, which continues forming and dispatching batches without interruption.

When a request is put into the completion queue, a strawman solution would be to update it as fully prefilled. However, this will cause a bug because only populating the KV cache is not equivalent to a full prefill. The reason is that, a full prefill additionally produces the *first output token* by sampling from the *hidden states*, i.e. the computation result of the model, while the KV cache is just the *intermediate state* of this computation. This problem is also present in previous prefix caching systems [62], and some propose storing both the KV cache and the hidden states in the storage server. In our system, we adopt a similar workaround to CacheGen [62] by marking the last token of the request as not yet prefilled, illustrated as A' and B' in Figure 6. When these last-token-
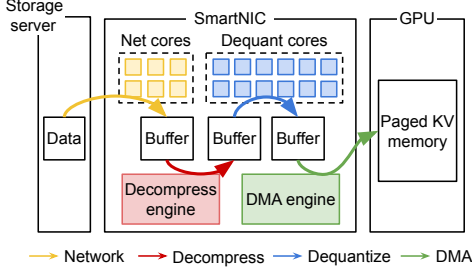
Figure 7: ShadowServe's chunked pipeline.

prefill jobs are restored to the scheduler, they are piggybacked immediately in the next batch.

Limitations. Currently, our design does not support chunked prefill. We also do not support partial hits, so a user request will either have its entire KV cache stored in the storage server, or have no KV cache stored at all. Nevertheless, we believe these features to be orthogonal to our contribution, and easy to integrate with our design. We discuss how to support them in §7.

4.2 Chunked Pipeline

To address the SmartNIC compute bottleneck, we divide the data operations into four stages and explicitly allocate Smart-NIC compute resources across them, including CPU cores and accelerators. To maximize SmartNIC resource utilization, we further pipeline these stages to make them run in parallel.

We perform four data operations in total on the SmartNIC, shown in Figure 7. First, we need to fetch the compressed data from the remote storage server via the network. Then, we need to perform lossless decompression on the data, followed by dequantization, to get the original KV cache tensor. Finally, we need to copy the original KV data from the on-chip memory to the GPU via DMA. The reason we need both decompression and dequantization is that the two steps combined ensure maximum compression ratio over the high-entropy KV cache data, following prior work [62].

Resource partitioning. To ease the load on the wimpy Arm cores, we first offload the lossless decompression and DMA operations to the hardware accelerators on the SmartNIC (see Figure 7). Decompression accelerators and DMA engines are present in many SmartNIC SoCs [5, 38, 50, 74]. For example, BlueField-3 supports hardware acceleration for lossless decompression with high throughput [78], and it has an on-chip DMA engine capable of copying data from the on-chip DRAM directly into host GPU via peer-to-peer DMA over PCIe (§2.2). Both engines work fully asynchronously to the Arm cores and incur negligible CPU load.

For the network and dequantization operations, we partition the on-chip Arm cores into two parts, assigning one part to each operation (see Figure 7). We further parallelize network transmission and dequantization to their assigned cores. For network, we split the transmitted data into equal-sized slices and receive each slice with one core. For dequantization, we
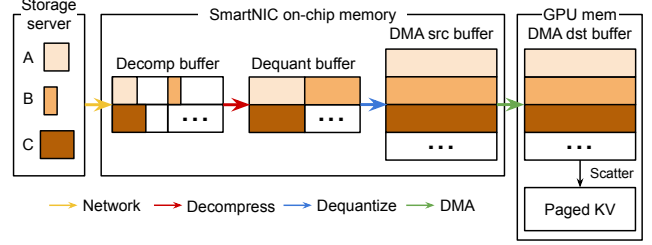
Figure 8: ShadowServe's memory management. It is also possible to perform DMA directly into paged KV memory, but this results in scattered data copies which significantly degrade DMA throughput.

use vector-wise data binning, and we split the KV cache tensor into 1D vectors, distributing them equally among the assigned cores. The number of cores assigned to each part is determined by the network and dequantization throughput per core to ensure balanced throughput.

To save context switching overhead, we start and pin all threads to the assigned core during SmartNIC program initialization, and use a lightweight thread-safe FIFO queue to pass tasks from the main program to these worker threads.

Four-stage chunked pipeline. To minimize fetch latency and maximize resource utilization, we pipeline all the four operations on the SmartNIC. Instead of processing each request's entire KV cache sequentially through the four stages, we split each KV cache tensor into fixed-size chunks, allowing each chunk to flow independently through the pipeline. This overlaps network transmission, lossless decompression, dequantization, and DMA transfer on the SmartNIC. Since we partitioned the compute resources, the operations have minimal compute interference with each other. This chunked pipeline enables continuous full utilization of all resources on the SmartNIC, and reduces end-to-end latency to the slowest of the four stages.

Prior systems also have similar pipelining techniques. For example, CacheGen [62] pipelines network transmission and GPU decompression. However, both CacheGen's lossless decompression and dequantization utilize the entire GPU, so they cannot be pipelined. Meanwhile, ShadowServe assigns different resources to all operations, enabling more fine-grained pipelining and parallelism.

4.3 Memory Management

To address the SmartNIC memory bottleneck, we design a *minimal-copy* memory management mechanism that eliminates redundant copies (Figure 8). It integrates seamlessly with the SmartNIC programming model and the chunked pipeline.

Memory pinning. Instead of creating memory buffers during runtime, we pre-allocate and pin all buffers during program initialization. They include (1) the decompression buffer, the dequantization buffer, and the DMA source buffer in the SmartNIC on-chip memory; (2) the DMA destination buffer in the GPU memory. Only these memory regions are accessed

by the operations. For example, the DMA destination buffer serves as the sole endpoint for DMA transfers into the GPU.

This technique has two benefits. First, it minimizes memory copies on the SmartNIC, as a subsequent operation directly reads from the previous one's output buffer. For example, in Figure 8, dequantization can be directly carried out on the output of decompression, i.e., data in the dequantization buffer, without any data copy. Second, it reduces memory registration overhead. All memory regions accessed by hardware accelerators should be registered with the SmartNIC, and memory registration during runtime usually causes significant delay (e.g., up to $3\times$ fetching latency on BlueField-3). Similar memory pinning strategies are present in other systems like NCCL [82].

A straw-man solution here is to reserve larger buffers so that each request's entire KV cache can fit in the buffers. However, this solution results in high GPU memory consumption caused by the DMA destination buffer, which leads to memory contention with model serving and even out-of-memory errors on the GPU. Therefore, we only allocate a medium-sized buffer to ensure a bounded GPU memory footprint. If one request's KV cache data is larger than any of the buffers, we perform the KV cache transfer for such a request in multiple *rounds*, each time fetching a subset of chunks that fit within the available buffer.

Buffer partitioning. With the chunked pipeline design (§4.2), each chunk goes through the pipeline independently, and operations on different chunks need to access the same memory buffer at the same time. For example, in Figure 8, we fetch 3 compressed KV cache chunks—A, B, and C—in the current round. When A is being dequantized, B is being decompressed in parallel, and both operations need to access the dequantization buffer. To avoid memory contention, we need to partition the buffers across chunks.

The challenge is to decide the memory size allocated to each chunk in each buffer, which we call *occupancy*. Luckily, for a given chunk, its occupancy in most of the buffers can be neatly determined. First, both in the DMA source buffer and the DMA destination buffer, a chunk's occupancy is its original KV cache tensor size, which is determined by the number of tokens in the chunk and the model dimensions (e.g., number of layers and channels). This size can be calculated locally during runtime. Since our quantization algorithm exactly halves the original data size, the occupancy in the dequantization buffer is half of the occupancy in the DMA buffers.

The last difficulty is to determine the occupancy in the decompression buffer, as we do not know the compressed data size for each chunk. A straw-man solution is to query the storage server, but it adds querying overhead for each round. Our solution leverages the fact that compressed size is always smaller than the original, and directly uses the same occupancy in the dequantization buffer as the decompression buffer. Since compressed size is smaller, we create some unused fragments in the decompression buffer, illustrated in Figure 8. With this strategy, each chunk's occupancy in both the decompression and dequantization buffers will always be half of that in the DMA buffers. Based on this observation, we make the decompression and dequantization buffers also half the size of the DMA buffers, so that for each round, the buffers are able to fit the same set of chunks.

Occupancies are computed with negligible overhead for every chunk in a round before the actual fetching begins. After that, the buffers are logically partitioned according to the computed occupancies, so that each chunk only utilizes its assigned part in all buffers. For example, in Figure 8, assuming one row in each buffer corresponds to 4 MB of data, each chunk's occupancy is 2 MB in the decompression and dequantization buffers, and 4 MB in the DMA buffers. When, for instance, chunk B is doing dequantization, it reads from its own allocated region in the dequantization buffer (2 MB), and writes to its allocated region in the DMA source buffer (4 MB).

Scattering. Every round after fetching a group of chunks, we launch a lightweight kernel to *scatter* the contiguous KV cache tensor in the DMA destination buffer into the paged KV memory. As both regions are in the GPU, this memory copy kernel causes negligible overhead, which is further mitigated by the fact that we launch this kernel every *round* after fetching multiple chunks, while previous works launch a decompression kernel for every *chunk*.

5 Implementation

We implement a system prototype of ShadowServe with 7K lines of code in Python and C++, and integrate it with vLLM, a state-of-the-art LLM serving system. ShadowServe can be integrated with any serving framework, and we choose vLLM because it is the most widely used. Our implementation leverages the NVIDIA BlueField-3 DPU, but ShadowServe can be implemented on any SmartNIC with decompression accelerators and P2P DMA capability, present on most current SmartNICs like Intel IPU [38] and AMD Pensando [5].

Host side. The KV cache manager runs as a thread in the vLLM process, communicating with the SmartNIC proxy via DOCA Comch [77], an event-driven message-based channel over PCIe. All SmartNIC-related functionalities are exposed to the manager as a C++ pybind function. This function communicates with the SmartNIC proxy, and blocks until it receives completion notification from the proxy. We release the Python Global Interpreter Lock (GIL) inside the pybind function, so that the vLLM scheduler thread is not blocked.

SmartNIC side. We use TCP as the transport between the SmartNIC and the storage server, and utilize NVIDIA Accelerated IO (XLIO) library, a user-space TCP/IP stack compatible with POSIX socket APIs, to improve network performance.

We disable Nagle's algorithm and TCP delayed acknowledgment to avoid stalls caused by small metadata exchanges between the SmartNIC proxy and the storage server.

For the chunked pipeline, we set the chunk size to 256 tokens, following prior work [62]. We allocate 2 of the 16 Arm cores on BlueField-3 to network (XLIO TCP), and the remaining 14 cores to dequantization. We further use SIMD (Arm Neon [7]) to boost the dequantization performance on the Arm cores. Both the decompression and DMA accelerators on BlueField-3 require polling from the Arm cores, and we set the polling interval as $10\mu s$, which ensures responsiveness while incurring negligible load on the Arm cores.

For memory management, we allocate 0.5 GiB for both the DMA source and destination buffers, and 0.25 GiB for the decompression and dequantization buffers. In addition to the DMA destination buffer, we reserve an additional 0.5 GiB in the GPU for tensor reshaping and scattering, capping our GPU memory footprint at 1 GiB. This size can be further reduced at the cost of some additional overhead, caused by launching the scattering kernel for more fetching rounds (§4.3). For scattering, we use the `reshape_and_cache` CUDA kernel implemented by vLLM. To handle BlueField-3's 2 MiB limit for accelerator operations (e.g., decompression and DMA), we pre-slice data into compatible blocks during the initial compression stage to avoid splitting already-compressed data.

Compression algorithm. We use vector-wise data binning for quantization, following previous work [62]. For each vector in the KV cache tensor, this method finds the maximum absolute value and uses it to scale all elements into a predefined set of bins. The resulting quantized data is losslessly compressed using the Deflate algorithm. We chose Deflate because it is hardware-accelerated by BlueField-3 and achieves a superior compression ratio on the quantized KV cache data compared to LZ4.

Storage server. The remote storage server is organized as a key-value store, where each entry stores the compressed KV cache for a given chunk. The key is the prefix hash of the request's prompt up to the chunk in question. The KV cache manager checks whether a request's KV cache is stored in the storage server by querying whether the last chunk's prefix hash exists as a key in the server.

6 Evaluation

This section aims to answer the following questions:

1. What is the throughput and latency of ShadowServe under different settings compared to existing approaches, and what is the trade-off space (§6.2)?
2. How is the SmartNIC data plane performing, and what is the bottleneck on the SmartNIC (§6.3)?
3. What are the effects of asynchronous fetching, chunked pipeline, and memory management on ShadowServe's performance (§6.4)?

6.1 Experiment Setup

Testbed. Our testbed consists of a single machine with two 16-core Intel Xeon Gold 6526Y CPUs at 3.50 GHz and 504 GB memory. It has an NVIDIA L40S GPU and an NVIDIA BlueField-3 DPU under NUMA node 0, and a dual-port Mellanox ConnectX-7 400 Gbps NIC under NUMA node 1, all connected via PCIe 4.0 ×16. The host machine runs Ubuntu 24.04 (kernel v6.8.0), and the BlueField-3 runs Ubuntu 22.04 (kernel v5.15.0) and DOCA v2.9.0. The BlueField-3 features a 16-core Cortex-A78AE Arm CPU with 32 GB of on-chip memory, and integrates a dual-port Mellanox ConnectX-7 400 Gbps NIC. One port of the BlueField-3 and the CX7 NIC under NUMA node 1 are connected via a loopback link. The CX7 NIC is rate limited with Mellanox OFED QoS [83]. In our experiments, the LLM serving engine runs on NUMA node 0, and the KV cache storage server runs in a separate network namespace on NUMA node 1, accessible through the physical loopback link between the two NICs.

**Baselines.** We compare ShadowServe to two baseline LLM serving systems.
- **vLLM [51].** It is a popular LLM inference system. We use vLLM v0.8.1, the latest version when starting the project.
- **CacheGen-Async.** It is a state-of-the-art prefix caching system with asynchronous KV cache fetching. Because the original implementation of CacheGen [62] does not support asynchronous fetching, we enable it with our design (§4.1). It bypasses the BlueField-3, using it as a normal NIC, and utilizes GPU for KV cache decompression. Like ShadowServe, it uses TCP as transport with the same optimizations discussed in §5, and implements a chunked pipeline of network transmission and decompression.

For CacheGen-Async, we use one custom CUDA stream for model computation, and another for KV cache decompression. ShadowServe also needs to run a lightweight scattering kernel in the GPU (§4.3), and we put it in a custom stream as well. We also evaluate the effect of using the default stream for model computation in §A.

Methodology. To measure the pure remote fetching performance of ShadowServe and CacheGen-Async, we configure the experiment to ensure every request triggers a fetch from the remote storage server, similar to CacheGen [62]. This is achieved by two means: First, we disable their local prefix caching (in GPU/CPU memory). Second, we ensure a 100% cache hit rate on the remote server, so that requests never need to be recomputed. To facilitate this, all necessary KV cache data is pre-loaded into the storage server's memory before each experiment. Conversely, vLLM serves as our recomputation baseline. We force it to recompute the KV cache for every request by also disabling its local prefix caching.

Workloads. We evaluate ShadowServe on two models of different sizes, Llama-8B and Mistral-7B. Both models are fine-tuned to take long contexts (32K for Mistral-7B and 128K for Llama-8B). We use two different datasets, TriviaQA and
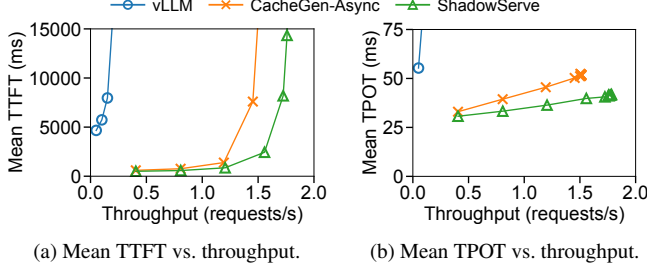
Figure 9: Load-latency curves for output length = 32 and network bandwidth = 20 Gbps. The TPOT curves do not show turning points, because TPOT does not include the queueing delay, and therefore does not increase further after the systems become fully loaded.

NarrativeQA. Both datasets are part of the LongBench [8] benchmark suite, designed to test the reading comprehension ability of LLMs by giving the LLMs a story or script, provided as a long document, and asking questions on it. TriviaQA has a medium length of 9.3K tokens, and NarrativeQA has a medium length of 14K. Both datasets have P95 lengths of 15K. For each experiment, we randomly sample 200 requests from one dataset, and the request arrival timestamps follow the Poisson process with different average rates.

Metrics. We measure two primary latency metrics for LLM serving: TTFT (time-to-first-token) and TPOT (time-per-output-token). For a given request, TTFT measures the latency of generating the first output token from the moment a request arrives in the system. For systems without prefix caching (vLLM), unloaded TTFT indicates the latency of prefill computation, while for prefix caching systems (CacheGen-Async and ShadowServe), it indicates the latency of fetching and decompressing the KV cache. TPOT measures the interval between the generation of consecutive output tokens of a request. For asynchronous fetching systems, a higher TPOT indicates stronger interference between KV cache decompression and model computation.

6.2 End-to-End Performance

6.2.1 Overall Performance

We first evaluate the end-to-end performance with the Llama-8B model and the NarrativeQA dataset. Figure 9 shows how the mean TTFT and TPOT of different systems change with request throughput, when all requests have a fixed output length of 32 and the network bandwidth is limited to 20 Gbps.

Both ShadowServe and CacheGen-Async significantly outperform vLLM, which suffers from high latency due to the lack of prefix caching. ShadowServe achieves 16% lower unloaded TTFT than CacheGen-Async (502.2ms vs. 600.5ms) for two reasons. First, with its data plane design (§4.2 and §4.3), ShadowServe's fetching and decompression throughput is higher than CacheGen-Async, resulting in lower fetching latency. Second, our use of Deflate as the lossless compression algorithm provides a better compression ratio than CacheGen-Async's arithmetic coding, thus reducing the data
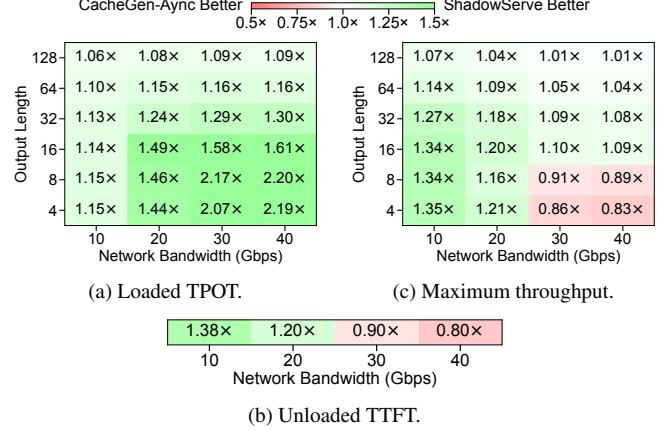
Figure 10: Performance comparison between ShadowServe and CacheGen-Async across different output lengths and network bandwidths. Unloaded TTFT is not affected by output length as it measures the latency before the first output token.
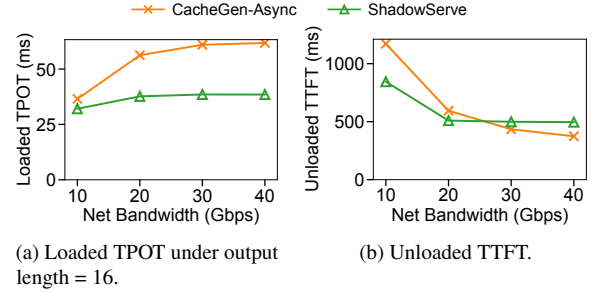
Figure 11: Absolute metrics for ShadowServe and CacheGen-Async.

transferred over the low-bandwidth network. ShadowServe also attains 20% lower loaded TPOT compared to CacheGen-Async (41.8ms vs. 52.0ms). This gap comes again from decompression offloading, as the GPU is now freed to focus solely on model computation (decode), preventing the interference seen in CacheGen-Async. These advantages combined allow ShadowServe to achieve 18% higher maximum throughput than CacheGen-Async (1.78 req/s vs. 1.51 req/s).

6.2.2 More Bandwidth and Output Length Settings

We extend our evaluation to more network bandwidth and output length configurations to show the entire trade-off space. Figure 10 shows the relative improvement of ShadowServe over CacheGen-Async on maximum throughput, loaded TPOT, and unloaded TTFT, respectively, using Llama-8B and NarrativeQA across a wide range of settings.

Loaded TPOT. ShadowServe consistently surpasses CacheGen-Async with 1.06–2.19× lower loaded TPOT across all output lengths and network bandwidths, demonstrating its ability to eliminate interference between KV cache decompression and model computation. ShadowServe is able to achieve much less interference because of its much less frequent and less heavyweight kernel launches in the

GPU (the per-round scattering kernel discussed in §4.3) compared with the per-chunk GPU decompression needed in CacheGen-Async.

Zooming in, we show the absolute values for loaded TPOT under 16 output tokens in Figure 11a. CacheGen-Async's loaded TPOT increases dramatically from 36.5ms to 61.8ms (1.7×) as the bandwidth scales up from 10 Gbps to 40 Gbps, while ShadowServe's loaded TPOT only increases slightly from 32.1ms to 38.5ms (1.2×), which leads to ShadowServe's increasing gain (1.14–1.61×) over CacheGen-Async as network scales. The increase in loaded TPOT in both systems stems from more frequent GPU kernel launches in their pipeline as the network stage becomes shorter for each chunk, leading to more interference with model computation.

ShadowServe yields less gain in loaded TPOT compared with CacheGen-Async for longer outputs. For example, it only achieves 1.06–1.09× better for 128 output tokens, compared with up to 2.19× for 4 output tokens. This occurs because with longer outputs, both systems' performance bottleneck shifts from data plane operations to GPU memory capacity. A request with a long output sequence occupies its KV cache memory region in the GPU for a prolonged decoding phase, stalling new fetches until it completes. During the periods when fetching is stalled, no decompression kernels run on the GPU, which temporarily eliminates the interference that typically penalizes CacheGen-Async. As a result, its TPOT during these compute-only phases becomes nearly optimal, improving its average loaded TPOT and narrowing the performance gap with ShadowServe.

Unloaded TTFT. ShadowServe achieves 1.20–1.38× lower unloaded TTFT than CacheGen-Async at bandwidths below 20 Gbps thanks to the better decompression ratio of Deflate, but is 11–24% higher above 20 Gbps. Looking more closely at the absolute values shown in Figure 11b, as network bandwidth scales from 10 Gbps to 40 Gbps, CacheGen-Async's fetching latency continues to decrease until 40 Gbps (e.g., an 11% decrease from 30 Gbps to 40 Gbps), while ShadowServe's fetching latency stops decreasing from 20 Gbps on, staying at around 500ms. This is caused by the premature network bottleneck in ShadowServe, as the Bluefield SoC could only achieve 20.6 Gbps bandwidth because of inter-stage contention in the SmartNIC pipeline, further analyzed in §6.3.

Nevertheless, these unloaded results assume no interference. Under load, CacheGen-Async will perform worse as its decompression becomes interfered by model computation in the GPU. In addition, based on our measurement, the decompression in CacheGen-Async has a maximum throughput of ∼ 32 Gbps under interference, so CacheGen-Async is already bottlenecked by decompression when the network bandwidth reaches 40 Gbps. Therefore, above 40 Gbps, CacheGen-Async's performance stops improving as well.

Maximum throughput. For shorter output lengths (4 and 8),
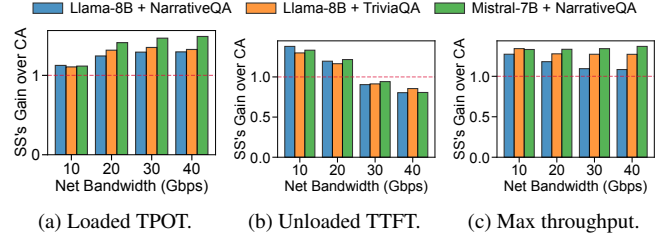
Figure 12: ShadowServe (SS)'s gain over CacheGen-Async (CA) under more models and datasets, output length = 32.

ShadowServe achieves 1.16–1.35× higher throughput in 10 and 20 Gbps, but becomes 9–17% worse for 30 and 40 Gbps, following roughly the same pattern observed in unloaded TTFT. This is because KV cache fetching (TTFT) dominates end-to-end performance for shorter outputs. In addition, shorter outputs result in less model computation (decode), resulting in less GPU interference for CacheGen-Async. Conversely, ShadowServe beats CacheGen-Async across all bandwidths for longer outputs (≥ 16) due to its lower TPOT, achieving 1.01–1.34× higher throughput. However, contrary to the pattern observed in loaded TPOT, the advantage of ShadowServe diminishes as network scales up, due to worse fetching performance (longer TTFT).

Summary of trade-off space. ShadowServe consistently excels in TPOT across all configurations due to its interference-free decompression offload. Conversely, CacheGen-Async achieves better TTFT at network bandwidths above 20 Gbps, as ShadowServe's fetching speed is bottlenecked by its SmartNIC pipeline. This trade-off dictates the overall throughput: for longer outputs (≥ 16 tokens) or lower bandwidth (≤ 20 Gbps), ShadowServe delivers higher throughput, while CacheGen-Async pulls ahead when both the output is short and the available bandwidth is high.

Looking beyond our tested settings, we project that for extremely long outputs (> 128 tokens), the performance of both systems would converge. This is because KV cache fetching will be constantly stalled as the GPU is busy doing many iterations of decode, a setting that neither system optimizes for. Similarly, at bandwidths beyond 40 Gbps, as both systems are already bottlenecked (CacheGen-Async by GPU decompression, ShadowServe by the SmartNIC), their relative performance would likely mirror the 40 Gbps results. At an even higher bandwidth threshold, a baseline without any decompression would likely outperform both, as the latency of decompression itself would outweigh the benefits of reduced data transfer. Ultimately, ShadowServe is the superior choice for most low-bandwidth prefix caching scenarios especially for TPOT-sensitive workloads, while its main limitation is its raw fetching performance in high-bandwidth networks.

6.2.3 Different Models and Datasets

We evaluate ShadowServe and CacheGen-Async's performance for other models and datasets. Figure 12 shows Shad-
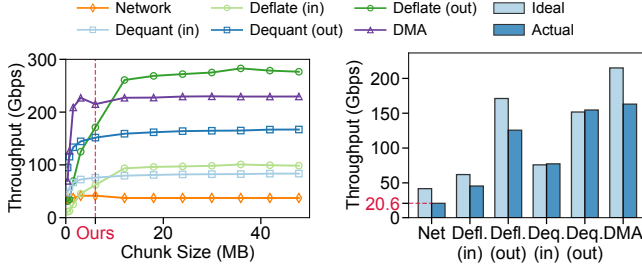
(a) Standalone (microbenchmark) performance for each pipeline stage under different chunk sizes.

(b) Standalone vs. actual performance for each pipeline stage using our chunk size (256 tokens).

Figure 13: SmartNIC microbenchmark and actual performance. There are two throughput numbers for Deflate and dequantization because they have different input and output data sizes. The results show that the pipeline is currently bottlenecked by the network stage.

owServe's gain over CacheGen-Async for loaded TPOT, unloaded TTFT, and maximum throughput, with two additional configurations: (Llama-8B + TriviaQA) and (Mistral-7B + NarrativeQA). The output length is fixed at 32 tokens.

ShadowServe's benefit stays similar across other models and datasets, indicating a very similar trade-off space as discussed in §6.2.2, demonstrating the generality of ShadowServe's design. For loaded TPOT, ShadowServe is still better than CacheGen-Async across all settings, with 1.11–1.33× improvement on (Llama-8B + TriviaQA), and 1.12–1.49× on (Mistral-7B + NarrativeQA). Like what we observed in §6.2.2, ShadowServe's gain increases as network bandwidth grows. For unloaded TTFT, ShadowServe is 1.17–1.38× better for 10 and 20 Gbps, and 6–20% worse for 30 and 40 Gbps, across all models and datasets. For maximum throughput, ShadowServe stays ahead by 1.08–1.37× across all settings.

6.3 SmartNIC Performance

We characterize the data plane performance on the SmartNIC. Specifically, we examine stages in the chunked pipeline (§4.2) and measure their standalone performance with a set of microbenchmarks, as well as their actual performance in the end-to-end system. We then provide an analysis of the performance bottleneck.

Standalone throughput of pipeline stages. Figure 13a shows how the throughput of each stage in the data plane's chunked pipeline varies with input chunk size. Standalone means that each stage runs alone on the SmartNIC using its assigned resources in the end-to-end system. For example, we allocate 2 cores to user-space TCP in our implementation (§5), so the network curve in Figure 13a shows the performance of user-space TCP on the two assigned cores without anything else running on the SmartNIC.

As Figure 13a shows, each operation's throughput increases with chunk size before saturating. The BlueField-3's Deflate accelerator is capable of reaching ~ 280 Gbps output throughput, and its DMA engine achieves a maximum of 230 Gbps, which interestingly does not saturate the PCIe 4.0×16 line

rate. Dequantization has a lower, yet still substantial, throughput of 83.5/167 Gbps input/output. In the ideal case, these results show that the pipeline is bottlenecked by the network stage, which saturates at 37.3 Gbps. This bottleneck dictates our choice of chunk size. The optimal size must be large enough for all other stages to operate above the network's throughput, yet small enough to enable fine-grained pipelining. Therefore, we select a chunk size of 6 MB (256 tokens).

Actual throughput. Figure 13b illustrates the performance degradation when all pipeline stages run together on the SmartNIC. While dequantization maintains its standalone performance, the Deflate and DMA stages experience throughput reductions of 27% and 24%, respectively. The most significant impact is on the network stage; already the bottleneck in isolation, its throughput drops by an additional 59% to 20.6 Gbps. This degradation is the cause of the suboptimal TTFT observed in ShadowServe (§6.2.2).

SmartNIC bottleneck analysis. A common question would be why not increase the number of cores assigned to network to improve its throughput. This is because currently the network operation is not CPU-bound. Indeed, increasing the number of cores assigned to network results in negligible throughput improvement without severely degrading dequantization. We believe BlueField-3's memory subsystem is the cause of the inter-stage contention, because compute resources are already partitioned to each stage (§4.2). To validate this, we run the same test as in Figure 13b but remove all memory accesses in dequantization. In this scenario, per-chunk dequantization latency becomes near-zero, and the network throughput recovers to ~ 30 Gbps.

We identify two potential sources for this memory contention. The first is the cache hierarchy. The BlueField-3 has 1 MB, 8 MB, and 16 MB L1, L2, L3 caches, respectively [74], much smaller than the host, and its L3 cache is smaller than the per-chunk working set of the dequantization stage. Our experiments confirm that more than 90% of L3 cache accesses miss, 40% of which are caused by dequantization. A smaller chunk size can mitigate this issue, but results in lower operation throughput (Figure 13a).

However, since the on-chip Deflate and DMA engines bypass the CPU caches, this is unlikely to be the sole factor. Another cause could be memory bandwidth. Although the BlueField-3 is equipped with DDR5 memory offering a high theoretical bandwidth (> 80 GB/s), it has only two memory channels [74], which can lead to head-of-line blocking at the memory controller, leading to suboptimal memory bandwidth. Despite these hardware limitations, our design remains promising. Future SmartNICs can be architected with more memory channels and higher bandwidth. Adding more memory channels is feasible, as Intel Mount Evans IPU includes three channels [13], and Marvell Octeon DPU supports up to six channels [68]. Moreover, improving the memory subsystem of the SmartNIC is more cost-effective as Arm SoCs are

(a) Loaded TPOT. (b) Unloaded TTFT. (c) Max throughput.
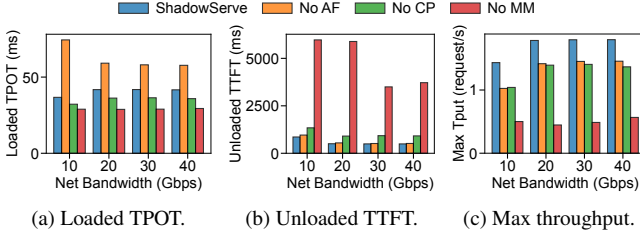
Figure 14: Impact of asynchronous fetching (AF), chunked pipeline (CP), and memory management (MM). Output length = 32.

less expensive than server-class x86 CPUs [50].

6.4 Ablation Studies

We evaluate the effectiveness of each component in ShadowServe (§4) by comparing the performance metrics for the following artifacts, each keeping all other components the same.

- **ShadowServe.** With all the techniques enabled.
- **No AF.** It turns off asynchronous fetching (§4.1) in ShadowServe. KV cache fetches block model execution, instead of happening in the background.
- **No CP.** It disables the chunked pipeline (§4.2) in the data plane on the SmartNIC. Chunks are processed one by one through the four stages, and each stage uses all resources on the SmartNIC.
- **No MM.** It disables memory management (§4.3) on the SmartNIC. Instead of pre-allocation and memory pinning, memory buffers are allocated and registered for each chunk on demand during runtime.

Figure 14 shows several metrics for the evaluated systems for 32 output tokens. Overall, ShadowServe achieves 1.2–4.5× higher maximum throughput, demonstrating the contribution of each technique in ShadowServe's design. Disabling asynchronous fetching (No AF) does not affect unloaded TTFT, as the fetching itself has roughly the same latency whether it happens in the foreground or background. However, it results in 1.39–2.03× higher TPOT, as blocking fetches constantly interrupt decodes and cause prolonged stalls between consecutive generated tokens, while ShadowServe is able to eliminate such stalls by moving KV cache fetching to the background. The gap is larger for lower network bandwidth because fetching takes even longer.

Disabling either the chunked pipeline (No CP) or memory management (No MM) increases unloaded TTFT, as both are critical for data plane performance and low-latency fetching. The impact is particularly severe for No MM, which exhibits a 6.96–11.73× higher TTFT compared to No CP (1.56–1.86×), due to the high overhead of on-demand memory registration (§4.3). It might be surprising that disabling the two techniques leads to an improved loaded TPOT (Figure 14a). This is because, as fetching performance deteriorates for No CP and No MM, the scattering kernel in ShadowServe's data plane (§4.3) is launched less often in the GPU, leading to slightly less interference and more efficient model computation. As No

MM's data plane performance degrades more severely than No CP's, this effect is more pronounced, improving TPOT by 21–31% for No MM versus 12–14% for No CP.

7 Discussion and Future Work

Prefill-decode disaggregation. Prefill-decode disaggregation [15,87,124] assigns prefill and decode to different GPUs, creating significant overhead from transmitting KV caches between them [20,35,55,124]. ShadowServe can benefit this setting by using its SmartNIC data plane to transparently compress the KV cache on the prefill side and decompress it on the decode side. The network latency can be further hidden with asynchronous sending and receiving (§4.1). Even if only one side is equipped with SmartNICs, single-sided offloading is still beneficial for low-bandwidth settings.

Chunked prefill and partial hits. Chunked prefill [2] breaks large prefill jobs into smaller chunks, and puts them in finer-grained mixed batches alongside decodes. ShadowServe's asynchronous fetching (§4.1) can support this feature. The KV cache manager would modify its batch interception mechanism to inspect the prefill chunks within each mixed batch. When a chunk is eligible for a cache hit, the manager intercepts it and begins the background fetch, using the same process as for full requests. Chunked prefill also enables partial hits. For a partially cached prefill request, the manager fetches the available prefix, and the scheduler computes the remainder as a chunked prefill.

Wish list for SmartNICs. We highlight the need for more powerful SmartNIC memory subsystems, as ShadowServe's performance is bottlenecked by the limited cache size and memory bandwidth of current devices (§6.3). Enhancing these would directly boost the data plane throughput. Additionally, emerging on-path cores (e.g., DPAs on BlueField-3 [17,121]) could be a promising alternative to the off-path cores utilized in ShadowServe, as they can process traffic directly on the data path, bypassing on-chip memory.

8 Related Work

LLM serving. Efficient LLM serving requires combining various optimizations. For a single node, fine-grained scheduling [2, 115], paged memory management [51], and operator pipelining [125] are proposed. For multiple nodes, PD-disaggregation [87, 124] separates prefill and decode phases to different GPUs with elastic parallelism scaling [104]. Recently, some propose PD-multiplexing on the same device [20, 35, 59] and Attention-FFN disaggregation [97]. All of these systems can leverage prefix caching to avoid recomputation, and thus can benefit from ShadowServe's design.

KV cache compression. KV cache compression has been well studied. Lossy algorithms include pruning [41, 63, 119], low-rank approximation [24, 111], quantization [36, 48, 64],

and many more. Transmission-oriented KV cache compression can be further augmented with lossless algorithms like arithmetic coding [62]. We emphasize that ShadowServe does not propose a new KV cache compression algorithm, but offloads existing ones from the host GPU/CPU to the SmartNIC. Therefore, ShadowServe's design is complementary to new compression algorithms.

SmartNIC offloading. SmartNICs present a promising approach to reducing host processing overheads [92]. Many applications have been offloaded to SmartNICs, including networking stacks [93, 95, 121], file systems and storage [33, 49, 71], distributed transactions [92, 102], content delivery [50], network functions [26, 53, 88, 90], and many more [19, 60, 61, 118]. ShadowServe shares the same vision and targets a new realm of applications, i.e., LLM serving, and offloads KV cache compression to the SmartNIC to achieve transparent KV cache fetching.

Asynchronous KV cache fetching. Recent works also explore asynchronous KV cache fetching. CachedAttention [27] overlaps layer-wise pre-loading with computation. However, it needs a dedicated, potentially unbounded pre-loading buffer in the GPU, while ShadowServe has a bounded GPU memory usage. KVFlow [86] also employs a proactive prefetching mechanism, but it is designed for agentic workflows with known prefix patterns. In contrast, ShadowServe supports general serving without this prior knowledge.

GPU multitasking. GPU multitasking mechanisms like MPS and CUDA Green Context are actively used in LLM serving [20, 35, 59, 125], but their performance is nondeterministic and workload-dependent as the GPU scheduler is closed-source. ShadowServe demonstrates their limitations when multiplexing decompression with model computation. Better ways to multiplex tasks on the GPU have been explored, but they either target limited workloads [9, 57, 94, 108, 117] or provide imperfect performance guarantees [34, 98, 105, 109]. In addition, even if GPUs achieve perfect multitasking, offloading is still beneficial as it saves GPU resources for more efficient model serving.

Other hardware for tensor compression. The latest GPUs are integrating on-chip hardware accelerators for decompression [81], but they are only present in high-end GPUs like NVIDIA Blackwell, and out of scope for low-cost serving. GPU video codecs also yield good compression ratio on KV cache tensors with little loss in accuracy [110]. However, these codecs have very limited throughput [110], and are only present in limited GPU types not suitable for LLM workloads.

9 Conclusion

We present ShadowServe, the first SmartNIC-accelerated prefix caching system that achieves *interference-free* KV cache fetching by offloading the entire data plane to the SmartNIC.

ShadowServe leverages a *chunked pipeline* and a *minimal-copy memory management* scheme to overcome on-device resource limitations and maximize throughput. ShadowServe achieves up to 2.2× lower loaded time-per-output-token and up to 1.35× higher throughput compared to state-of-the-art systems. Our work shows that SmartNICs are a promising and underutilized compute tier for LLM serving infrastructure.

References

[1] System Prompts in Large Language Models. `https://promptengineering.org/system-prompts-in-large-language-models`, 2024.

[2] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 117–134, Santa Clara, CA, July 2024. USENIX Association.

[3] ai-dynamo/NIXL Development Team. NVIDIA Inference Xfer Library (NIXL). `https://github.com/ai-dynamo/nixl`, 2025.

[4] Amazon Web Services. *Amazon EC2 Instance Types*. Amazon Web Services, 2025.

[5] AMD. AMD Pensando DSC3-400 Distributed Services Card: Product Brief. Technical report, AMD, 2024.

[6] Anthropic. Prompt caching with Claude. `https://www.anthropic.com/news/prompt-caching`, August 2024.

[7] Arm Limited. *Arm NEON Programmer's Guide*, Mar 2020. Document ID: DEN 0018D. Available at `https://developer.arm.com/documentation/den0018/d/`.

[8] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3119–3137, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[9] Zhihao Bai, Zhen Zhang, Yibo Zhu, and Xin Jin. PipeSwitch: Fast pipelined context switching for deep learning applications. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI*

20), pages 499–514. USENIX Association, November 2020.

[10] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020.

[11] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R. Gormley. Unlimiformer: long-range transformers with unlimited length input. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[12] Matt Bowman and Jeremy Baumgartner. Grand teton systems overview. `https://www.youtube.com/watch?v=fmXfWad-NiA`, 2023.

[13] Brad Burres, Dan Daly, Mark Debbage, Eliel Louzoun, Christine Severns-Williams, Naru Sundar, Nadav Turbovich, Barry Wolford, and Yadong Li. Intel's hyperscale-ready infrastructure processing unit (ipu). In *2021 IEEE Hot Chips 33 Symposium (HCS)*, pages 1–16. IEEE, 2021.

[14] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 715–730, New York, NY, USA, 2025. Association for Computing Machinery.

[15] Shiyang Chen, Rain Jiang, Dezhi Yu, Jinlai Xu, Mengyuan Chao, Fanlong Meng, Chenyu Jiang, Wei Xu, and Hang Liu. Kvdirect: Distributed disaggregated llm inference, 2024.

[16] Weijian Chen, Shuibing He, Haoyang Qu, Ruidong Zhang, Siling Yang, Ping Chen, Yi Zheng, Baoxing Huai, and Gang Chen. IMPRESS: An Importance-Informed Multi-Tier prefix KV storage system for large language model inference. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 187–201, Santa Clara, CA, February 2025. USENIX Association.

[17] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, and Zeke Wang. Demystifying Datapath Accelerator Enhanced Off-path SmartNIC . In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pages 1–12, Los Alamitos, CA, USA, October 2024. IEEE Computer Society.

[18] Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. Do large language models need a content delivery network?, 2024.

[19] Sean Choi, Muhammad Shahbaz, Balaji Prabhakar, and Mendel Rosenblum. λ-nic: Interactive serverless compute on programmable smartnics. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pages 67–77, 2020.

[20] Weihao Cui, Yukang Chen, Han Zhao, Ziyi Xu, Quan Chen, Xusheng Chen, Yangjie Zhou, Shixuan Sun, and Minyi Guo. Optimizing slo-oriented llm serving with pd-multiplexing, 2025.

[21] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy, July 2019. Association for Computational Linguistics.

[22] DeepSeek AI. Fire-Flyer File System (3FS). `https://github.com/deepseek-ai/3FS`, 2025.

[23] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, 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, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan

Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[24] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. 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, T. 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, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

[25] Antaeus Feldspar. An Explanation of the Deflate Algorithm. https://zlib.net/feldspar.html, 1997.

[26] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, Renton, WA, April 2018. USENIX Association.

[27] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-efficient large language model serving for multi-turn conversations with cachedattention. In *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'24, USA, 2024. USENIX Association.

[28] In Gim, Guojun Chen, Seung-Seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa, editors, *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024.

[29] Google Cloud. *Network Bandwidth — Compute Engine Documentation*. Google Cloud, 2025.

[30] Google Cloud. *Networking and GPU machines: Network Bandwidth — Google Cloud Compute Engine Documentation*. Google Cloud, 2025.

[31] Dan Graur, Oto Mraz, Muyu Li, Sepehr Pourghannad, Chandramohan A. Thekkath, and Ana Klimovic. Pecan: Cost-Efficient ML data preprocessing with

automatic transformation ordering and hybrid placement. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 649–665, Santa Clara, CA, July 2024. USENIX Association.

[32] Tyler Griggs, Xiaoxuan Liu, Jiaxiang Yu, Doyoung Kim, Wei-Lin Chiang, Alvin Cheung, and Ion Stoica. Mélange: Cost efficient large language model serving by exploiting gpu heterogeneity, 2024.

[33] Zerui Guo, Hua Zhang, Chenxingyu Zhao, Yuebin Bai, Michael Swift, and Ming Liu. Leed: A low-power, fast persistent key-value store on smartnic jbofs. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 1012–1027, New York, NY, USA, 2023. Association for Computing Machinery.

[34] Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 539–558, Carlsbad, CA, July 2022. USENIX Association.

[35] Ke Hong, Lufang Chen, Zhong Wang, Xiuhong Li, Qiuli Mao, Jianping Ma, Chao Xiong, Guanyu Wu, Buhe Han, Guohao Dai, Yun Liang, and Yu Wang. semi-pd: Towards efficient llm serving via phase-wise disaggregated computation and unified storage, 2025.

[36] Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[37] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. Memserve: Context caching for disaggregated llm serving with elastic memory pool, 2024.

[38] Intel Corporation. Intel Infrastructure Processing Unit (IPU) SoC E2100: Product Brief. Technical report, Intel Corporation, 2024.

[39] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. Skyplane: Optimizing transfer cost and throughput using Cloud-Aware overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1375–1389, Boston, MA, April 2023. USENIX Association.

[40] Yichao "Peak" Ji. Context Engineering for AI Agents: Lessons from Building Manus. `https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus`, July 2025.

[41] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376, Singapore, December 2023. Association for Computational Linguistics.

[42] Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, and Vidushi Dadu. Rago: Systematic performance optimization for retrieval-augmented generation serving. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ISCA '25, page 974–989, New York, NY, USA, 2025. Association for Computing Machinery.

[43] YOUHE JIANG, Fangcheng Fu, Xiaozhe Yao, Guoliang HE, Xupeng Miao, Ana Klimovic, Bin CUI, Binhang Yuan, and Eiko Yoneki. Demystifying cost-efficiency in LLM serving over heterogeneous GPUs. In *Forty-second International Conference on Machine Learning*, 2025.

[44] YOUHE JIANG, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin CUI, Ana Klimovic, and Eiko Yoneki. Thunderserve: High-performance and cost-efficient LLM serving in cloud environments. In *Eighth Conference on Machine Learning and Systems*, 2025.

[45] Youhe Jiang, Ran Yan, Xiaozhe Yao, Yang Zhou, Beidi Chen, and Binhang Yuan. Hexgen: generative inference of large language model over heterogeneous environment. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[46] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation, 2024.

[47] Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Zhuoqing Mao. Compute or load KV cache? why not both? In *Forty-second International Conference on Machine Learning*, 2025.

[48] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. Gear: An efficient kv cache compression recipe for near-lossless generative inference of llm, 2024.

[49] Jongyul Kim, Insu Jang, Waleed Reda, Jaeseong Im, Marco Canini, Dejan Kostić, Youngjin Kwon, Simon Peter, and Emmett Witchel. Linefs: Efficient smart-nic offload of a distributed file system with pipeline parallelism. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 756–771, New York, NY, USA, 2021. Association for Computing Machinery.

[50] Taehyun Kim, Deondre Martin Ng, Junzhi Gong, Youngjin Kwon, Minlan Yu, and KyoungSoo Park. Rearchitecting the TCP stack for I/O-Offloaded content delivery. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 275–292, Boston, MA, April 2023. USENIX Association.

[51] 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 '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery.

[52] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, Santa Clara, CA, July 2024. USENIX Association.

[53] Bojie Li, Kun Tan, Layong (Larry) Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 1–14, New York, NY, USA, 2016. Association for Computing Machinery.

[54] Yubo Li, Xiaobin Shen, Xinyu Yao, Xueying Ding, Yidi Miao, Ramayya Krishnan, and Rema Padman. Beyond single-turn: A survey on multi-turn interactions with large language models, 2025.

[55] Yueying Li, Zhanqiu Hu, Esha Choukse, Rodrigo Fonseca, G. Edward Suh, and Udit Gupta. Ecoserve: Designing carbon-aware ai inference systems, 2025.

[56] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 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, Boston, MA, July 2023. USENIX Association.

[57] Gangmuk Lim, Jeongseob Ahn, Wencong Xiao, Youngjin Kwon, and Myeongjae Jeon. Zico: Efficient GPU memory sharing for concurrent DNN training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 161–175. USENIX Association, July 2021.

[58] Chien-Yu Lin, Keisuke Kamahori, Yiyu Liu, Xiaoxiang Shi, Madhav Kashyap, Yile Gu, Rulin Shao, Zihao Ye, Kan Zhu, Stephanie Wang, Arvind Krishnamurthy, Rohan Kadekodi, Luis Ceze, and Baris Kasikci. Telerag: Efficient retrieval-augmented generation inference with lookahead retrieval, 2025.

[59] Zejia Lin, Hongxin Xu, Guanyi Chen, Xianwei Zhang, and Yutong Lu. Bullet: Boosting gpu utilization for llm serving via dynamic spatial-temporal orchestration, 2025.

[60] Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta. Offloading distributed applications onto smartnics using ipipe. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 318–333, New York, NY, USA, 2019. Association for Computing Machinery.

[61] Ming Liu, Simon Peter, Arvind Krishnamurthy, and Phitchaya Mangpo Phothilimthana. E3: Energy-Efficient microservices on SmartNIC-Accelerated servers. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 363–378, Renton, WA, July 2019. USENIX Association.

[62] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, page 38–56, New York, NY, USA, 2024. Association for Computing Machinery.

[63] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 52342–52364. Curran Associates, Inc., 2023.

[64] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: a tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[65] LMCache contributors. LMCache: Supercharge Your LLM with the Fastest KV Cache Layer. `https://github.com/LMCache/LMCache`, 2025.

[66] Frank Sifei Luan, Ziming Mao, Ron Yifeng Wang, Charlotte Lin, Amog Kamsetty, Hao Chen, Cheng Su, Balaji Veeramani, Scott Lee, SangBin Cho, Clark Zinzow, Eric Liang, Ion Stoica, and Stephanie Wang. The streaming batch model for efficient and fault-tolerant heterogeneous execution, 2025.

[67] Ziming Mao, Tian Xia, Zhanghao Wu, Wei-Lin Chiang, Tyler Griggs, Romil Bhardwaj, Zongheng Yang, Scott Shenker, and Ion Stoica. Skyserve: Serving ai models across regions and clouds with spot instances. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 159–175, New York, NY, USA, 2025. Association for Computing Machinery.

[68] Marvell. Marvell OCTEON 10 DPU Platform. `https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-octeon-10-dpu-platform-product-brief.pdf`, June 2025.

[69] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. Helix: Serving large language models over heterogeneous gpus and network via max-flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, page 586–602, New York, NY, USA, 2025. Association for Computing Machinery.

[70] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. Spotserve: Serving generative large language models on preemptible instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24, page 1112–1127, New York, NY, USA, 2024. Association for Computing Machinery.

[71] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In Hwan Doh, and Arvind Krishnamurthy. Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 106–122, New York, NY, USA, 2021. Association for Computing Machinery.

[72] Derek G. Murray, Jiří Šimša, Ana Klimovic, and Ihor Indyk. tf.data: a machine learning data processing framework. *Proc. VLDB Endow.*, 14(12):2945–2958, July 2021.

[73] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[74] NVIDIA Corporation. NVIDIA® BlueField®-3 DPU Datasheet. Technical report, NVIDIA Corporation, 2021.

[75] NVIDIA Corporation. *Introduction to the NVIDIA DGX A100 System — DGX A100 System Topology*. NVIDIA Corporation, 2024.

[76] NVIDIA Corporation. *CUDA Driver API: Green Contexts*. NVIDIA Corporation, 2025.

[77] NVIDIA Corporation. *DOCA Comch – NVIDIA DOCA SDK Documentation*. NVIDIA Corporation, 2025.

[78] NVIDIA Corporation. *DOCA Compress — NVIDIA DOCA SDK Documentation (v3.1.0)*. NVIDIA Corporation, 2025.

[79] NVIDIA Corporation. *Dynamo Distributed KV Cache Manager (NVIDIA Dynamo SDK v0.2.0)*. NVIDIA Corporation, 2025.

[80] NVIDIA Corporation. *Multi-Process Service (MPS) — NVIDIA Documentation*. NVIDIA Corporation, 2025.

[81] NVIDIA Corporation. NVIDIA Blackwell Architecture. `https://www.nvidia.com/en-us/data-center/technologies/blackwell-architecture/`, 2025.

[82] NVIDIA Corporation. *NVIDIA Collective Communications Library (NCCL)*. NVIDIA Corporation, 2025.

[83] NVIDIA Corporation. *Quality of Service (QoS) — NVIDIA DOCA SDK Documentation*. NVIDIA Corporation, 2025.

[84] OpenAI. *Prompt Caching — OpenAI API Guide*. OpenAI, 2025.

[85] Oracle Corporation. *Oracle Cloud Compute: Pricing Overview*. Oracle Corporation, 2025.

[86] Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows, 2025.

[87] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, page 118–132. IEEE Press, 2025.

[88] Phitchaya Mangpo Phothilimthana, Ming Liu, Antoine Kaufmann, Simon Peter, Rastislav Bodik, and Thomas Anderson. Floem: A programming system for NIC-Accelerated network applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 663–679, Carlsbad, CA, October 2018. USENIX Association.

[89] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*, pages 155–170, Santa Clara, CA, February 2025. USENIX Association.

[90] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. Automated smartnic offloading insights for network functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 772–787, New York, NY, USA, 2021. Association for Computing Machinery.

[91] 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.

[92] Henry N. Schuh, Weihao Liang, Ming Liu, Jacob Nelson, and Arvind Krishnamurthy. Xenic: Smartnic-accelerated distributed transactions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 740–755, New York, NY, USA, 2021. Association for Computing Machinery.

[93] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. FlexTOE: Flexible TCP offload with Fine-Grained parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 87–102, Renton, WA, April 2022. USENIX Association.

[94] Sudipta Saha Shubha, Haiying Shen, and Anand Iyer. USHER: Holistic interference avoidance for resource optimized ML inference. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 947–964, Santa Clara, CA, July 2024. USENIX Association.

[95] Athinagoras Skiadopoulos, Zhiqiang Xie, Mark Zhao, Qizhe Cai, Saksham Agarwal, Jacob Adelmann, David Ahern, Carlo Contavalli, Michael Goldflam, Vitaly Mayatskikh, Raghu Raja, Daniel Walton, Rachit Agarwal, Shrijeet Mukherjee, and Christos Kozyrakis. High-throughput and flexible host networking for accelerated computing. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 405–423, Santa Clara, CA, July 2024. USENIX Association.

[96] Vikranth Srivatsa, Zijian He, Reyna Abhyankar, Dongming Li, and Yiying Zhang. Preble: Efficient distributed prompt scheduling for LLM serving. In *The Thirteenth International Conference on Learning Representations*, 2025.

[97] StepFun, :, Bin Wang, Bojun Wang, Changyi Wan, Guanzhe Huang, Hanpeng Hu, Haonan Jia, Hao Nie, Mingliang Li, Nuo Chen, Siyu Chen, Song Yuan, Wuxun Xie, Xiaoniu Song, Xing Chen, Xingping Yang, Xuelin Zhang, Yanbo Yu, Yaoyu Wang, Yibo Zhu, Yimin Jiang, Yu Zhou, Yuanwei Lu, Houyi Li, Jingcheng Hu, Ka Man Lo, Ailin Huang, Binxing Jiao, Bo Li, Boyu Chen, Changxin Miao, Chang Lou, Chen Hu, Chen Xu, Chenfeng Yu, Chengyuan Yao, Daokuan Lv, Dapeng Shi, Deshan Sun, Ding Huang, Dingyuan Hu, Dongqing Pang, Enle Liu, Fajie Zhang, Fanqi Wan, Gulin Yan, Han Zhang, Han Zhou, Hanghao Wu, Hangyu Guo, Hanqi Chen, Hanshan Zhang, Hao Wu, Haocheng Zhang, Haolong Yan, Haoran Lv, Haoran Wei, Hebin Zhou, Heng Wang, Heng Wang, Hongxin Li, Hongyu Zhou, Hongyuan Wang, Huiyong Guo, Jia Wang, Jiahao Gong, Jialing Xie, Jian Zhou, Jianjian Sun, Jiaoren Wu, Jiaran Zhang, Jiayu Liu, Jie Cheng, Jie Luo, Jie Yan, Jie Yang, Jieyi Hou, Jinguang Zhang, Jinlan Cao, Jisheng Yin, Junfeng Liu, Junhao Huang, Junzhe Lin, Kaijun Tan, Kaixiang Li, Kang An, Kangheng Lin, Kenkun Liu, Lei Yang, Liang Zhao, Liangyu Chen, Lieyu Shi, Liguo Tan, Lin Lin, Lin Zhang, Lina Chen, Liwen Huang, Liying Shi, Longlong Gu, Mei Chen, Mengqiang Ren, Ming Li, Mingzhe Chen, Na Wang, Nan Wu, Qi Han, Qian Zhao, Qiang Zhang, Qianni Liu, Qiaohui Chen, Qiling Wu, Qinglin

He, Qinyuan Tan, Qiufeng Wang, Qiuping Wu, Qi-uyan Liang, Quan Sun, Rui Li, Ruihang Miao, Ruosi Wan, Ruyan Guo, Shangwu Zhong, Shaoliang Pang, Shengjie Fan, Shijie Shang, Shilei Jiang, Shiliang Yang, Shiming Hao, Shuli Gao, Siming Huang, Siqi Liu, Tiancheng Cao, Tianhao Cheng, Tianhao Peng, Wang You, Wei Ji, Wen Sun, Wenjin Deng, Wenqing He, Wenzhen Zheng, Xi Chen, Xiangwen Kong, Xianzhen Luo, Xiaobo Yang, Xiaojia Liu, Xiaoxiao Ren, Xin Han, Xin Li, Xin Wu, Xu Zhao, Yanan Wei, Yang Li, Yangguang Li, Yangshijie Xu, Yanming Xu, Yaqiang Shi, Yeqing Shen, Yi Yang, Yifei Yang, Yifeng Gong, Yihan Chen, Yijing Yang, Yinmin Zhang, Yizhuang Zhou, Yuanhao Ding, Yuantao Fan, Yuanzhen Yang, Yuchu Luo, Yue Peng, Yufan Lu, Yuhang Deng, Yuhe Yin, Yujie Liu, Yukun Chen, Yuling Zhao, Yun Mou, Yunlong Li, Yunzhou Ju, Yusheng Li, Yuxiang Yang, Yuxiang Zhang, Yuyang Chen, Zejia Weng, Zhe Xie, Zheng Ge, Zheng Gong, Zhenyi Lu, Zhewei Huang, Zhichao Chang, Zhiguo Huang, Zhirui Wang, Zidong Yang, Zili Wang, Ziqi Wang, Zixin Zhang, Binxing Jiao, Daxin Jiang, Heung-Yeung Shum, and Xiangyu Zhang. Step-3 is large yet affordable: Model-system co-design for cost-effective decoding, 2025.

[98] Foteini Strati, Xianzhe Ma, and Ana Klimovic. Orion: Interference-aware, fine-grained gpu sharing for ml applications. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, page 1075–1092, New York, NY, USA, 2024. Association for Computing Machinery.

[99] Ihab Tarazi. A State-of-the-Art Data Center for Large-Scale AI. https://www.dell.com/en-us/blog/a-state-of-the-art-data-center-for-large-scale-ai/, November 2023.

[100] Szymon Tworkowski, Konrad Staniszewski, Mikołaj Pacek, Yuhuai Wu, Henryk Michalewski, and Piotr Miłoś. Focused transformer: contrastive training for context scaling. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[101] Jiahao Wang, Jinbo Han, Xingda Wei, Sijie Shen, Dingyan Zhang, Chenguang Fang, Rong Chen, Wenyuan Yu, and Haibo Chen. Kvcache cache in the wild: Characterizing and optimizing kvcache cache at a large cloud provider. In Deniz Altinbüken and Ryan Stutsman, editors, *Proceedings of the 2025 USENIX Annual Technical Conference, USENIX ATC 2025, Boston, MA, USA, July 7-9, 2025*, pages 465–482. USENIX Association, 2025.

[102] Xingda Wei, Rongxin Cheng, Yuhan Yang, Rong Chen, and Haibo Chen. Characterizing off-path SmartNIC for accelerating distributed systems. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 987–1004, Boston, MA, July 2023. USENIX Association.

[103] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, June 1987.

[104] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, SOSP '24, page 640–654, New York, NY, USA, 2024. Association for Computing Machinery.

[105] Bingyang Wu, Zili Zhang, Zhihao Bai, Xuanzhe Liu, and Xin Jin. Transparent GPU sharing in container clouds for deep learning workloads. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 69–85, Boston, MA, April 2023. USENIX Association.

[106] Bingyang Wu, Zili Zhang, Yinmin Zhong, Guanzhe Huang, Yibo Zhu, Xuanzhe Liu, and Xin Jin. Tokenlake: A unified segment-level prefix cache pool for fine-grained elastic long-context llm serving, 2025.

[107] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers, 2022.

[108] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. AntMan: Dynamic scaling on GPU clusters for deep learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 533–548. USENIX Association, November 2020.

[109] Jiarong Xing, Yifan Qiao, Simon Mo, Xingqi Cui, Gur-Eyal Sela, Yang Zhou, Joseph Gonzalez, and Ion Stoica. Towards efficient and practical gpu multitasking in the era of llm, 2025.

[110] Ceyu Xu, Yongji Wu, Xinyu Yang, Beidi Chen, Matthew Lentz, Danyang Zhuo, and Lisa Wu Wills. Vcllm: Video codecs are secretly tensor codecs, 2024.

[111] Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[112] Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. Kvshare: An llm service system with efficient and effective multi-tenant kv cache reuse, 2025.

[113] Yann Collet and the LZ4 community. LZ4: Extremely Fast Compression Algorithm. https://github.com/lz4/lz4, 2025.

[114] Lu Ye, Ze Tao, Yong Huang, and Yang Li. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11608–11620, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

[115] 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, Carlsbad, CA, July 2022. USENIX Association.

[116] Lingfan Yu, Jinkun Lin, and Jinyang Li. Stateful large language model serving with pensieve. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 144–158, New York, NY, USA, 2025. Association for Computing Machinery.

[117] Peifeng Yu and Mosharaf Chowdhury. Salus: Fine-grained gpu sharing primitives for deep learning applications, 2019.

[118] Jie Zhang, Hongjing Huang, Xuzheng Chen, Xiang Li, Jieru Zhao, Ming Liu, and Zeke Wang. Rpcnic: Enabling efficient datacenter rpc offloading on pcie-attached smartnics. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1379–1394, 2025.

[119] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[120] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. Fast, approximate vector queries on very large unstructured datasets. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 995–1011, Boston, MA, April 2023. USENIX Association.

[121] Chenxingyu Zhao, Jaehong Min, Ming Liu, and Arvind Krishnamurthy. White-Boxing RDMA with Packet-Granular software control. In *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, pages 427–449, Philadelphia, PA, April 2025. USENIX Association.

[122] Yilong Zhao, Shuo Yang, Kan Zhu, Lianmin Zheng, Baris Kasikci, Yang Zhou, Jiarong Xing, and Ion Stoica. Blendserve: Optimizing offline inference for autoregressive large models with resource-aware batching, 2024.

[123] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 62557–62583. Curran Associates, Inc., 2024.

[124] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, Santa Clara, CA, July 2024. USENIX Association.

[125] Kan Zhu, Yufei Gao, Yilong Zhao, Liangyu Zhao, Gefei Zuo, Yile Gu, Dedong Xie, Zihao Ye, Keisuke Kamahori, Chien-Yu Lin, Ziren Wang, Stephanie Wang, Arvind Krishnamurthy, and Baris Kasikci. Nanoflow: Towards optimal large language model serving throughput. In Lidong Zhou and Yuanyuan Zhou, editors, *19th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2025, Boston, MA, USA, July 7-9, 2025*, pages 749–765. USENIX Association, 2025.
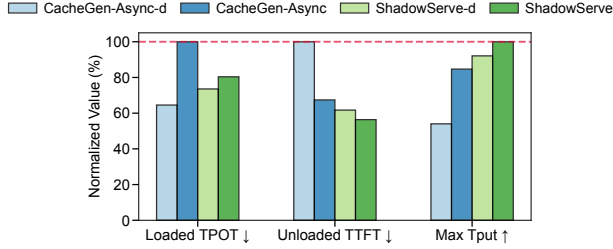
Figure 15: Performance of more baselines for output length = 32 and network bandwidth = 20 Gbps.

A   Effect of CUDA Streams

Both CacheGen-Async and ShadowServe use two custom CUDA streams in the GPU for multitasking. We evaluate two additional baselines, ShadowServe-d and CacheGen-Async-d, that use the default stream for model computation. Figure 15 shows normalized metrics for the systems under 32 output tokens and 20 Gbps bandwidth. For both systems, moving model computation to the default stream leads to lower loaded TPOT (35% lower for CacheGen-Async-d and 8% for ShadowServe-d) and higher unloaded TTFT (48% and 9%, respectively), creating interesting new points in the trade-off space. This is because kernels in the default stream (model computation) are prioritized over those in the custom stream (decompression for CacheGen-Async, and scattering for ShadowServe) in this setting. The effect is much more pronounced for CacheGen-Async, as it launches much more kernels in the custom stream. CacheGen-Async-d even achieves 20% lower loaded TPOT than ShadowServe. However, its prolonged TTFT leads to 46% lower throughput.

We note that this effect caused by the default stream is non-deterministic, as the GPU scheduler is closed-source. For example, we observe the opposite behavior in Figure 3b, where using a custom stream for model computation yields better decode performance than using the default stream.