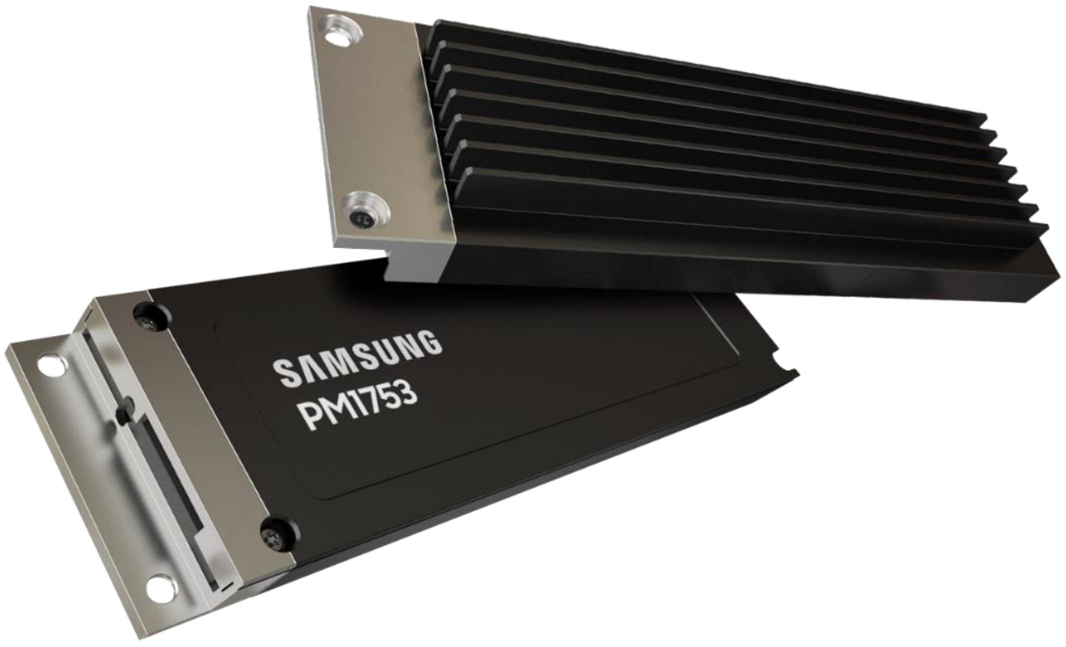
White Paper

Scaling AI Inference with KV Cache Offloading

Enhancing AI efficiency using Samsung PM1753 NVMe® SSDs

NAND AE Group (Memory)

Author:
Sungup Moon



Performance acceleration using KV cache offloading

Large language models (LLMs) continue to evolve toward processing longer contexts and performing more sophisticated reasoning. As agentic AI systems—capable of interpreting user intent and executing multi-step actions—are adopted more broadly, both conversational context and the model's internal state have expanded significantly.

Techniques such as model distillation and reasoning optimization help improve efficiency, but they also drive rapid growth in the size of key-value (KV) cache. As context lengths increase and multi-turn interactions become more frequent, GPU memory limitations become a critical bottleneck, making KV cache offloading an essential method for scaling inference workloads.

This whitepaper provides an in-depth analysis of workload characteristics and the performance impact of KV cache storage offloading. The findings offer insights into how offloading enhances scalability and improves resource efficiency across modern LLM infrastructures.

Introduction: LLM inference and KV cache offloading

What is the KV cache?

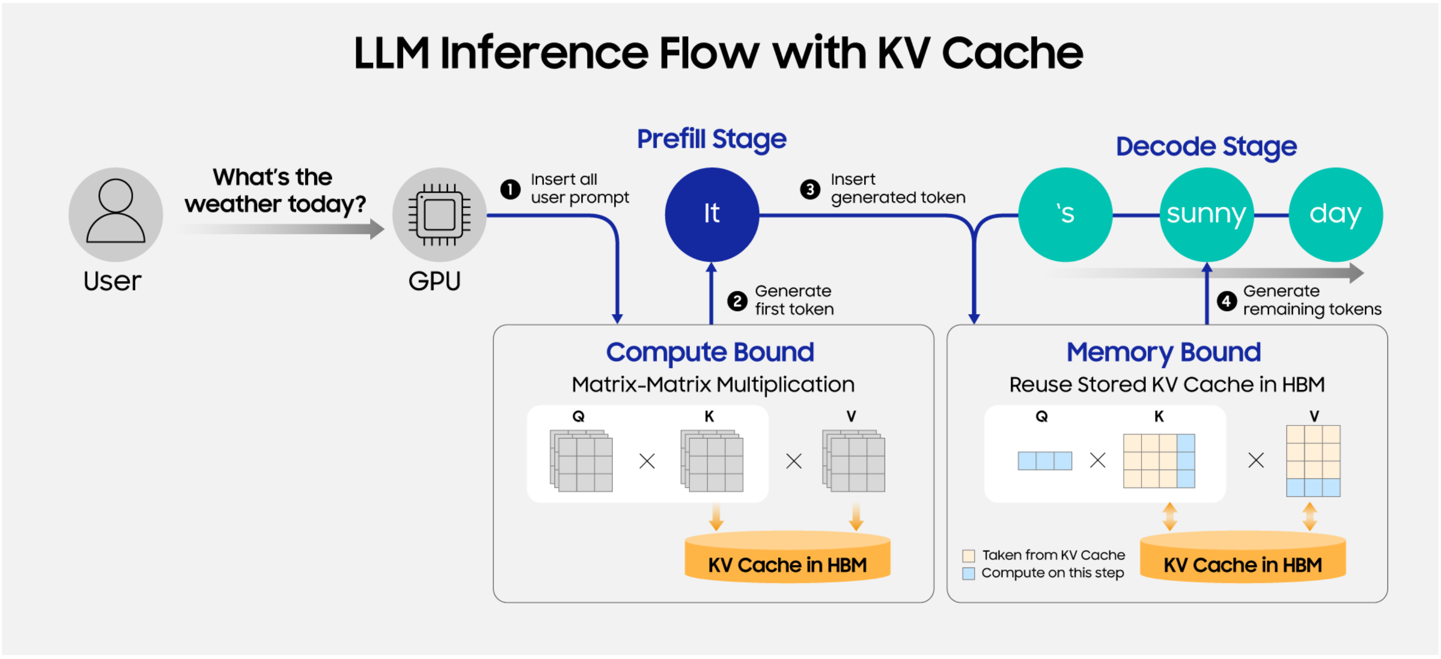


Figure 1. LLM inference flow with KV cache

LLMs generate text by repeatedly using the attention key and value tensors produced in previous steps to predict subsequent tokens. Recomputing these tensors for every new token would introduce substantial computational overhead. To avoid this, LLMs maintain a temporary memory structure called the KV cache, which stores keys and values generated at previous sequence positions across all attention layers. By reusing these cached representations, the model eliminates redundant calculations, improving both latency and throughput during inference.

LLM inference generally consists of two stages: prefill and decode.

- **Prefill stage:**
  In the prefill stage, the full prompt (Q) is processed once to generate the initial set of key-value pairs, which are stored in high-bandwidth memory (HBM) as the KV Cache. This stage is compute-bound, dominated by matrix–matrix operations, and it produces the first output token while initializing the cache for subsequent decoding.
- **Decode stage:**
  In the decoding stage, the model uses the stored KV cache together with the most recently generated token to produce the next token. Only the key-value pair for the new token is computed and appended. Because decoding proceeds token by token, it requires less compute per step but involves frequent HBM access, making it primarily memory-bound.

As a result, the KV Cache plays a critical role in optimizing decoding performance—particularly for long-context workloads and real-time generation scenarios—by enabling efficient reuse of previously computed representations.

Why KV cache offloading is important in agentic AI environments

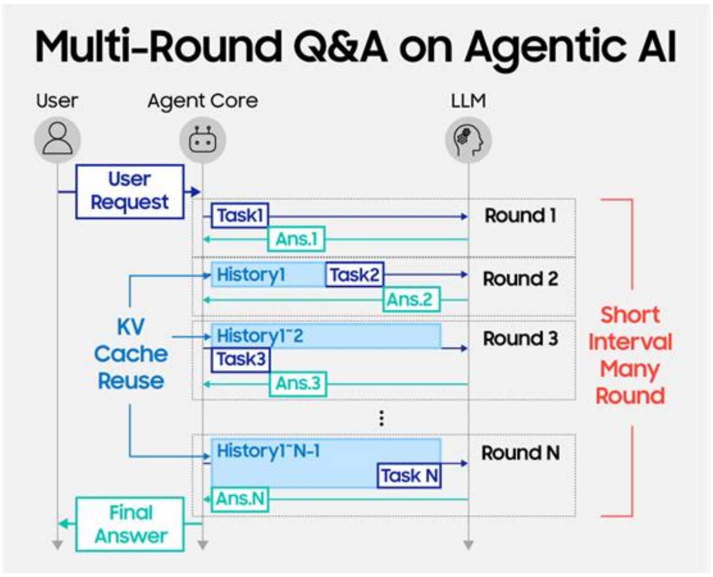


Figure 2. Multi-Round Q&A mechanism on agentic AI

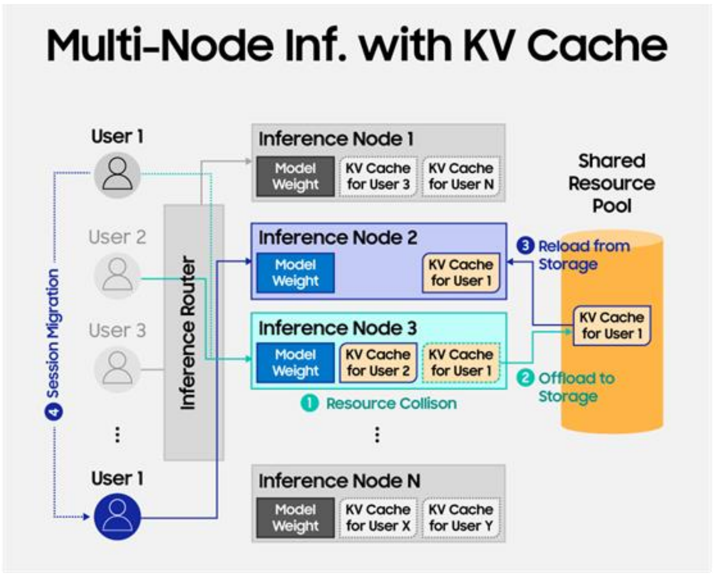


Figure 3. KV cache solution for multi-node inference

KV cache reuse in agentic AI workflows

Agentic AI frameworks rely on iterative, multi-turn interactions between an agent core and an LLM. For each user request, the agent core typically:

1. Interprets user intent
2. Generates an execution plan
3. Retrieves external information via tools
4. Integrates intermediate results
5. Repeats the cycle until the final output aligns with the user's goal

Because this loop involves repeated refinement of the same task, the model often processes variations of the same prompt multiple times (Figure 2). As the prompt grows across iterations, recomputing the entire prefill stage for each cycle becomes increasingly inefficient.

By leveraging the KV Cache, previously computed key-value pairs can be reused during decoding, avoiding redundant computation and helping maintain lower latency throughout multi-step reasoning workflows.

KV cache persistence challenges in multi-node environments

In multi-node agentic AI clusters, workloads may migrate across GPUs due to load balancing, elastic resource allocation, or fault recovery. When an active session moves to another GPU, the locally stored KV cache is lost and must be regenerated during the prefill stage on the new device, introducing significant latency.

To avoid this overhead, the KV cache must persist beyond local GPU memory. As illustrated in Figure 3, offloading the cache to external system resources—such as CPU memory or NVMe-based storage—enables fast reload and reuse after migration. For cross-node transitions, the cache must be accessible across multiple compute nodes through a shared resource, ensuring continuity during distributed execution.

In these dynamic environments, KV cache offloading operates not merely as a performance optimization but as a foundational architectural mechanism for maintaining stable latency, predictable throughput, and scalable inference operations.

Samsung PM1753 NVMe SSD

The PM1753 is a low-power TLC-based SSD built with eighth-generation V-NAND technology. It is engineered to support the high-throughput and write-intensive access patterns common in AI workloads while helping reduce overall power consumption. Delivering up to 14.5 GB/s sequential throughput and 3.3M random read IOPS, the PM1753 provides the bandwidth and parallelism required for KV cache offloading at scale. It also integrates enterprise-grade features—including advanced power-loss protection (PLP), AES-XTS 256-bit hardware encryption, and OCP compliance—to ensure consistent performance and reliability in large-scale AI inference and data-intensive environments.

Workload characteristics

To evaluate the behavioral patterns and performance impact of KV cache offloading, we configured a test environment using an Intel 6th-generation Xeon platform equipped with eight NVIDIA H100 NVL GPUs connected via PCIe. Eight parallel LLaMA-3.1 8B inference services were deployed, enabling controlled measurement of multi-GPU offloading efficiency under representative KV cache migration scenarios.

Detailed hardware and software specifications are provided in Appendix: Implementation notes.

I/O characteristics

When LMCache* is used to manage the KV cache, the model groups multiple tokens into larger KV cache blocks that can be shared across GPUs. In our setup, the maximum prompt length was set to 20,000 tokens, which LMCache divided into 256-token blocks. Each block occupied approximately 33 MB, resulting in 89 total blocks. Thus, when a session migrates between GPUs, the system transfers these 33 MB KV cache block files, creating a workload characterized primarily by large-block sequential I/O during KV cache offloading.

** LMCache: An open-source KV cache management layer designed for LLM inference. It stores and reuses key-value tensors across queries and engines to reduce recomputation, improve latency, and increase throughput—especially in long-context scenarios.*

In this LMCache-based environment, storage writes occur only for newly generated tokens, while read operations dominate during session migration as the target GPU reloads the required KV cache blocks. Across the full evaluation, the read/write ratio was approximately 92% reads and 8% writes, demonstrating a distinctly read-dominant pattern.

At the system level, aggregate I/O exhibited roughly 58% random and 34% sequential distribution, reflecting modest skew toward random access. However, at the per-process level, the access pattern becomes significantly more sequential—approximately 78% sequential reads—since each inference service issues dedicated 33 MB sequential reads for its assigned KV cache blocks.

The apparent randomness emerges from concurrency: multiple inference services issue sequential read streams concurrently, causing global interleaving of large read operations.

Additionally, because the application layer issues 33 MB reads, 96% of I/O requests at the Linux block layer exceeded 1 MB, indicating a heavy bias toward large-chunk transfers.

Consequently, storage devices used for KV cache storage offloading must deliver high sustained throughput for large-block operations and maintain strong sequential-access detection, even under multi-process, interleaved workload conditions.

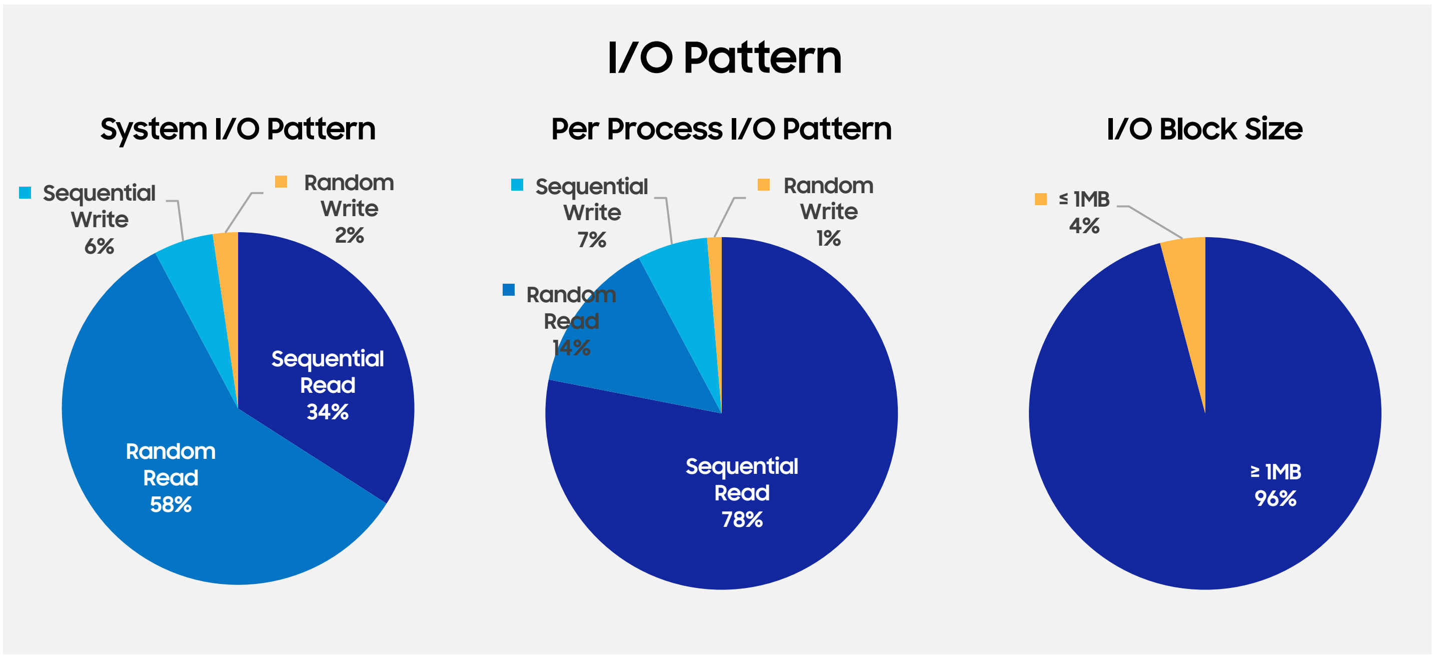


Figure 4. IO characteristics of SSD during benchmark execution

Logical block address (LBA) access pattern

In the vLLM and LMCache-based environment, the I/O characteristics previously described manifest as the LBA access pattern shown in Figure 5. The XFS file system used in this evaluation manages the 8 TB PM1753 as 32 allocation groups, resulting in 32 distinct access regions across the device. As multiple processes concurrently issue sequential I/O requests, several independent sequential streams occur simultaneously within overlapping LBA regions.

By examining a 100 ms interval within a single allocation group, the sequential I/O behavior relevant to NVMe devices becomes more apparent. Each 33 MB KV cache block file is segmented by the Linux block layer into sub-requests—primarily 2.8 MB and 6.1 MB in size. These sub-requests are then further divided according to the device's maximum data transfer size (MDTS), resulting in a series of MDTS-aligned operations issued to the NVMe controller.

Because these request streams overlap in time, the NVMe device receives multiple concurrent sequential-read sequences. Therefore, the device's ability to detect and maintain sequential access patterns across interleaved streams is essential for achieving optimal performance. For devices configured with a smaller MDTS, additional mechanisms—such as read prefetching—become increasingly important to preserve LBA locality and maintain throughput under such workloads.

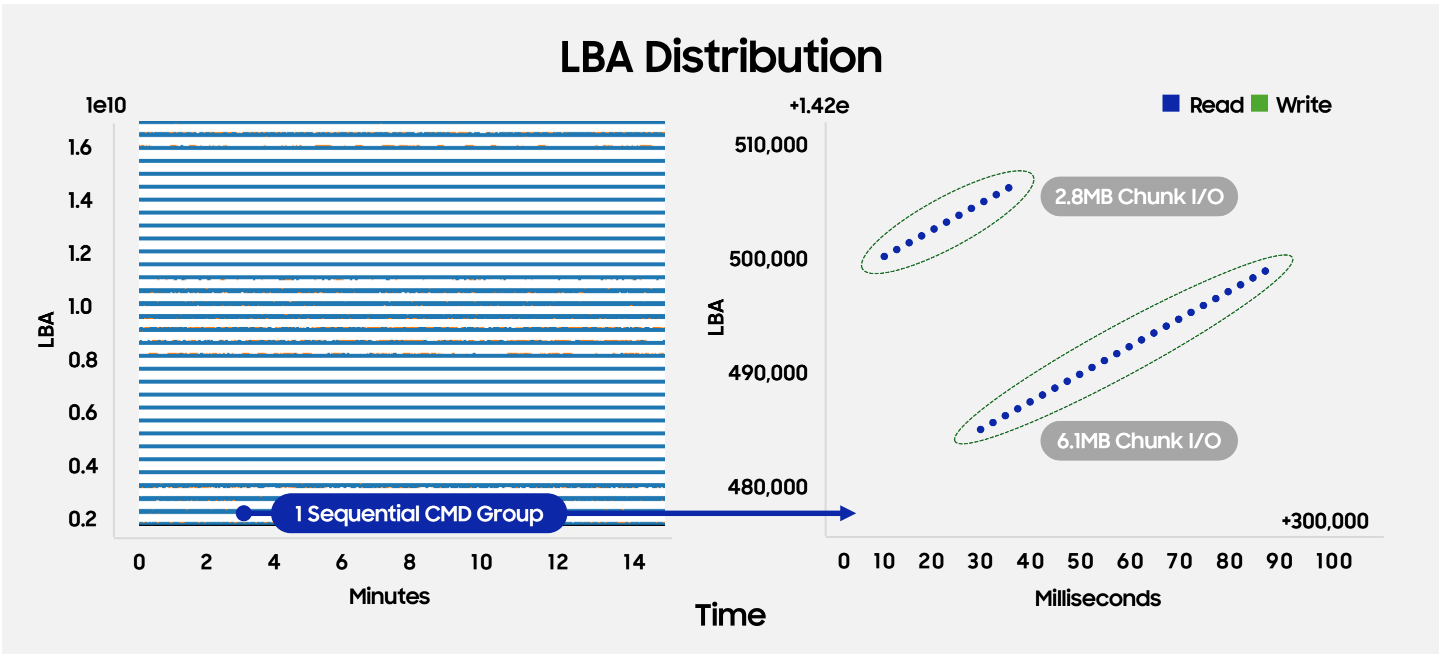


Figure 5. Logical Block Address (LBA) distribution graph of SSD during benchmark execution

Throughput and queue depth analysis

The evaluation environment employed a RAID-0 array consisting of ten NVMe devices. Throughput and queue depth behavior were measured as concurrent user count increased in steps of 20, from 20 up to the system's maximum of 280 users. With a query interval of 20 seconds per user, the required QPS ranged from 1 to 14. Since each session required approximately 3 GB of KV cache, the aggregate throughput demand scaled proportionally with active user count, reaching an average of up to 37 GB/s.

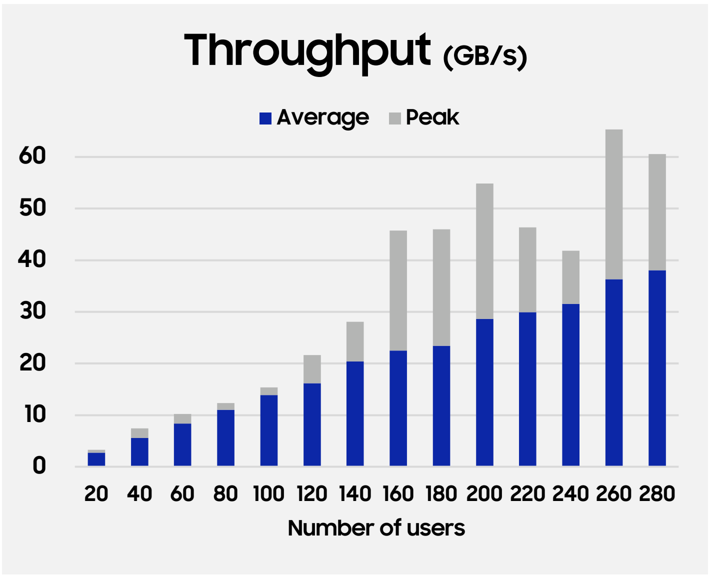


Figure 6. KV cache required throughput through number of users

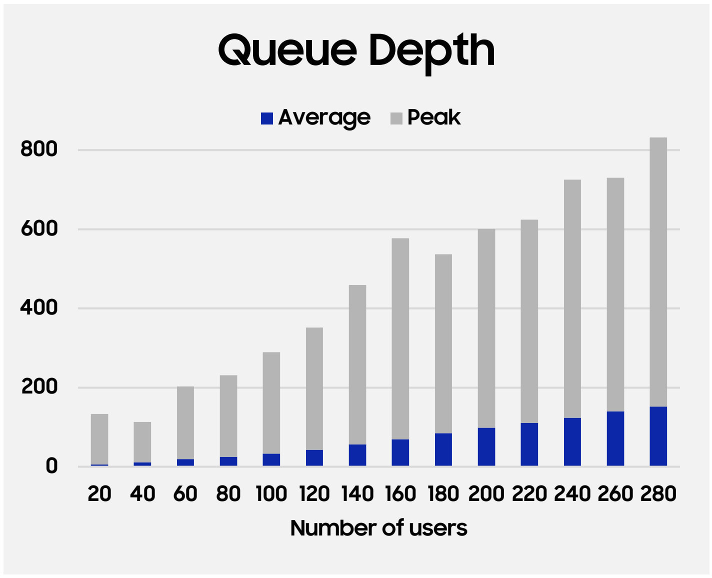


Figure 7. Average and peak queue depth during benchmark

The PM1753 supports PCIe Gen5 and delivers a maximum sequential throughput of 14.5 GB/s. In a ten-device RAID-0 configuration, the theoretical peak performance reaches up to 140 GB/s. Even with eight GPUs issuing KV cache operations, the average throughput requirement remains within a manageable range.

However, because all GPUs may issue I/O simultaneously, peak throughput can surge to nearly twice the average. Similarly, the number of in-flight I/O operations—typically around 160—can spike to as high as 800 (a 5x increase), resulting in a highly fluctuating access pattern. To ensure stable performance during KV cache offloading, the storage subsystem should provide at least 2x the average bandwidth so that these transient bursts can be absorbed without degrading service responsiveness.

Performance comparison

LLM performance comparison

The primary performance indicators for LLM inference are time to first token (TTFT) and tokens per second (TPS), which represent latency and throughput, respectively. While high throughput demonstrates strong computational capability, user experience depends heavily on maintaining acceptable latency. This becomes even more critical in agentic AI systems, where multiple agents interact in iterative reasoning loops. Latency increases can cascade through these loops, creating system-wide bottlenecks and lowering overall responsiveness.

Figure 8 compares TTFT with and without KV cache storage offloading. Using a 1-second latency threshold as a baseline for acceptable LLM responsiveness, the non-offloading configuration begins to show latency growth beyond 140 concurrent users, with TTFT rising proportionally to load. In contrast, when KV cache storage offloading is enabled, the system maintains sub-second latency up to 240 concurrent users, supporting 1.7× more users at equivalent latency. From a scale-out perspective, this enables the same service capacity with fewer GPU servers.

Even within the 20–140 user range—where both configurations meet sub-second latency—the offloading-enabled system provides more consistent cache-loading behavior, maintaining TTFT at approximately 0.5 seconds. This highlights a clear advantage over the compute-only prefill execution, delivering faster and more predictable response times under varying concurrency levels.

From a TPS standpoint (Figure 9), the configuration using KV cache offloading generates approximately 1.5× more tokens under the same 240-user condition. By eliminating repeated prefill operations, offloading frees GPU compute resources for decoding, sustaining higher TPS. Without offloading, the system begins to saturate at around 160 concurrent users, while the offloading-enabled system continues to scale effectively, demonstrating improved scalability and resource utilization.

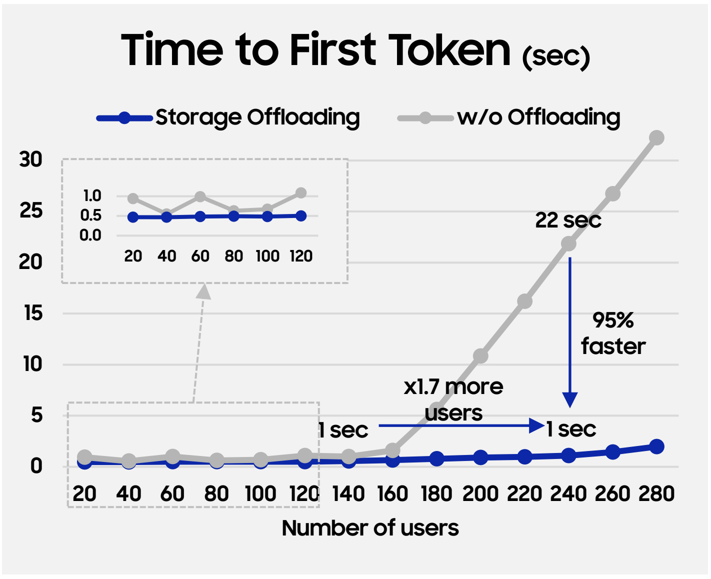


Figure 8. TTFT comparison between the storage offloading on/off

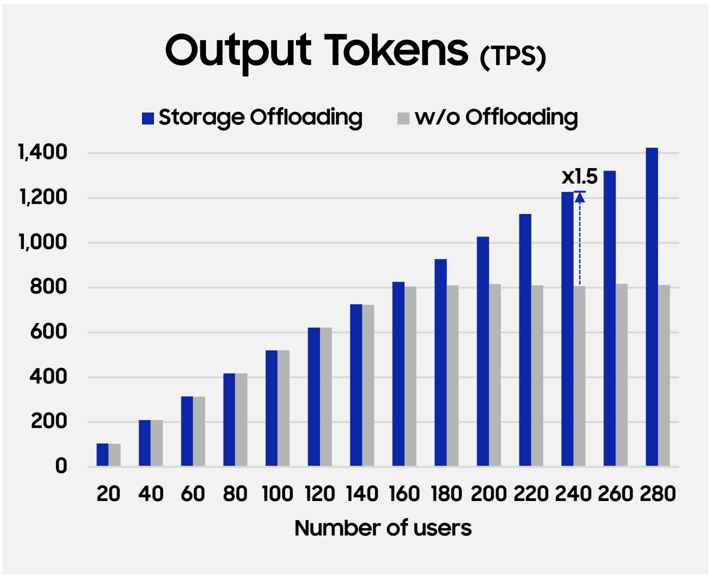


Figure 9. TPS comparison between the storage offloading on/off

Power consumption

As shown in the previous TPS evaluation, KV cache storage offloading reduces the proportion of prefill computation, increasing the relative share of the decoding stage in overall processing. In LLM inference, the prefill stage is compute-bound due to the batch processing of input tokens, while the decoding stage reuses previously generated tokens and requires significantly fewer compute resources. As illustrated in Figure 10, this shift reduces GPU utilization to approximately 49%, highlighting the efficiency gains provided by storage offloading.

The decrease in GPU utilization also improves thermal characteristics. As shown in Figure 11, without offloading, all system fans operate at their maximum expected speed (15,000 RPM) to dissipate GPU heat. With KV cache storage offloading enabled, the reduced thermal output allows the CPU/Memory/Storage-side fans to operate at around 7,800 RPM and the GPU-side fans at approximately 9,800 RPM, resulting in a system-wide average of about 8,800 RPM.

This decrease in thermal load and fan speed translates directly into energy savings. As illustrated in Figure 12, the total system power consumption drops to roughly 53% of the non-offloading configuration, demonstrating the substantial power-efficiency benefits of KV cache storage offloading.

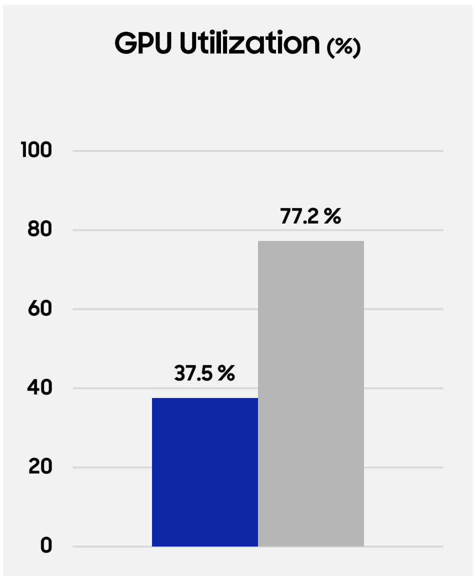


Figure 10. GPU utilization comparison

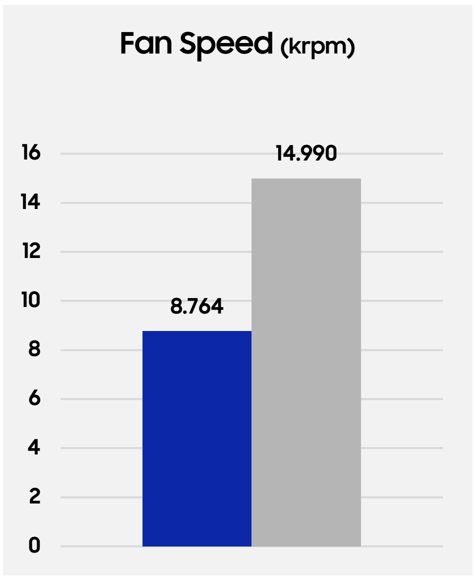


Figure 11. Fan speed comparison

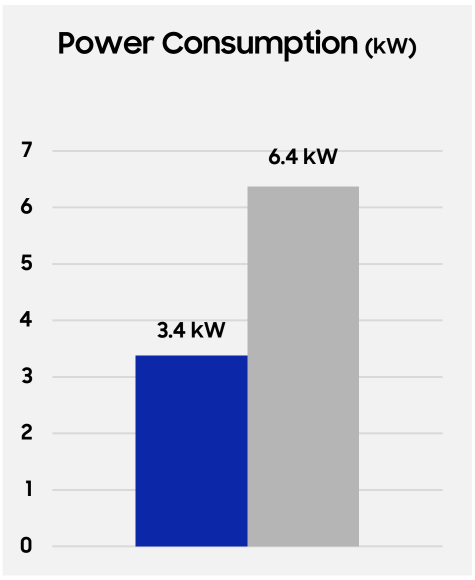


Figure 12. Power consumption comparison

Efficiency comparison

Based on the performance results presented above, the efficiency of systems adopting KV cache storage offloading can be evaluated from two perspectives: initial deployment cost and operational cost.

Deployment cost: Performance per dollar

GPU-based LLM systems have a distinctive characteristic in which users directly perceive computation time, making latency and responsiveness critical. Because such systems often require multiple GPU servers operating concurrently, both capital cost and operational complexity increase significantly. As shown in Figure 14, GPUs account for the majority of total server cost, far exceeding other system components. Therefore, rather than reducing deployment cost by adjusting server hardware configurations, it is far more effective to maximize GPU utilization and performance efficiency within existing server configurations.

To evaluate economic efficiency, we compared deployment efficiency over a three-year operational period using tokens per second per dollar, a metric that captures effective performance delivered per unit of investment. As shown in Figure 13, in the 0–160 concurrent user range, both configurations deliver similar TPS because sufficient GPU compute capacity is available regardless of offloading. However, in this region, the offloading-enabled system incurs approximately 4% higher, directly reflecting the additional NVMe component cost.

Beyond 180 concurrent users, KV cache storage offloading enables the system to bypass repeated prefill computation, freeing GPU compute capacity for decoding and increasing total token throughput. In this range, the additional NVMe cost is fully offset by the increased TPS. At 240 users, the configuration with KV cache storage offloading achieves 1.5× improvement in cost efficiency, demonstrating substantial economic advantages under high-concurrency scenarios.

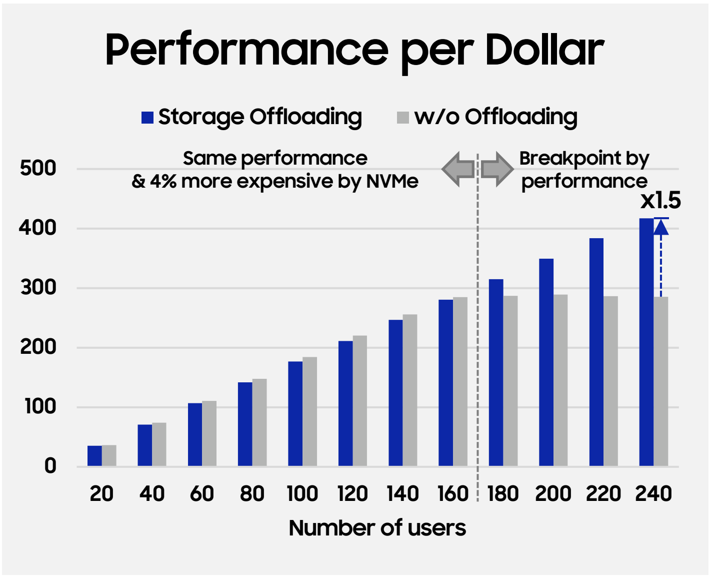


Figure 13. TPS per Dollar comparison through number of users

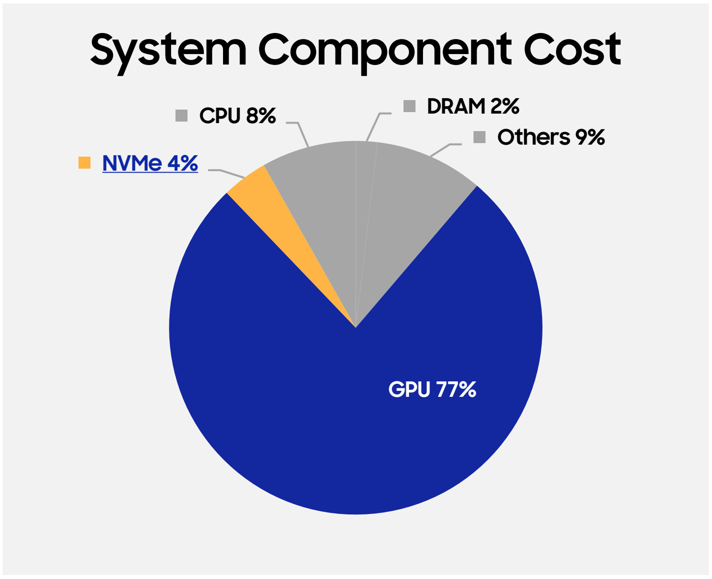


Figure 14 System component cost in 8 x H100 GPU server

Operational efficiency: Performance per watt

As previously described, reduced GPU utilization not only lowers direct GPU power consumption but also decreases overall system thermal load, thereby lowering total energy usage. In GPU servers, maintaining power efficiency is especially critical because high-performance cooling—particularly fan-driven airflow—contributes significantly to total system power consumption.

When normalizing operational efficiency using tokens per second per watt, systems employing KV cache storage offloading consistently achieve lower operating costs across all concurrency levels due to reduced overall system power consumption. Although additional NVMe devices increase power usage by approximately 6%, this overhead is outweighed by significant reductions in the power drawn by GPUs and cooling fans, which dominate the system's energy profile.

At the system's maximum evaluated concurrency (240 users), the offloading-enabled configuration achieves a 2.9× improvement in power efficiency, highlighting the substantial operational benefits and long-term cost reductions enabled by KV cache storage offloading.

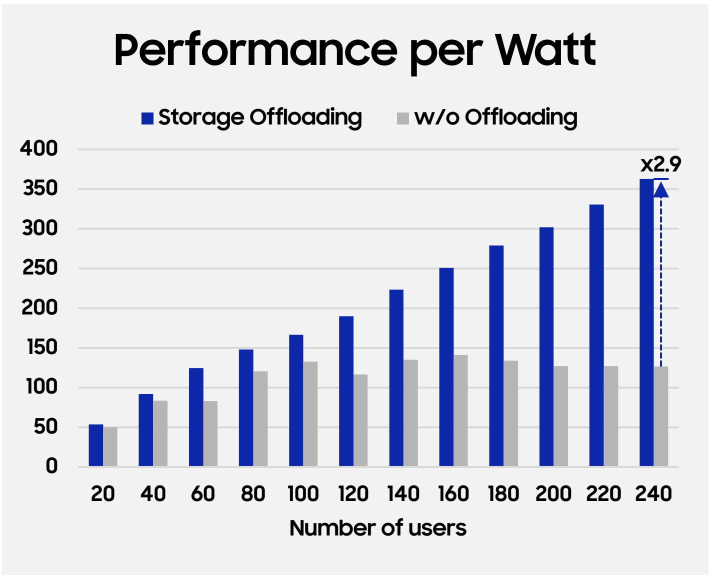


Figure 15. TPS per Watt comparison through number of users

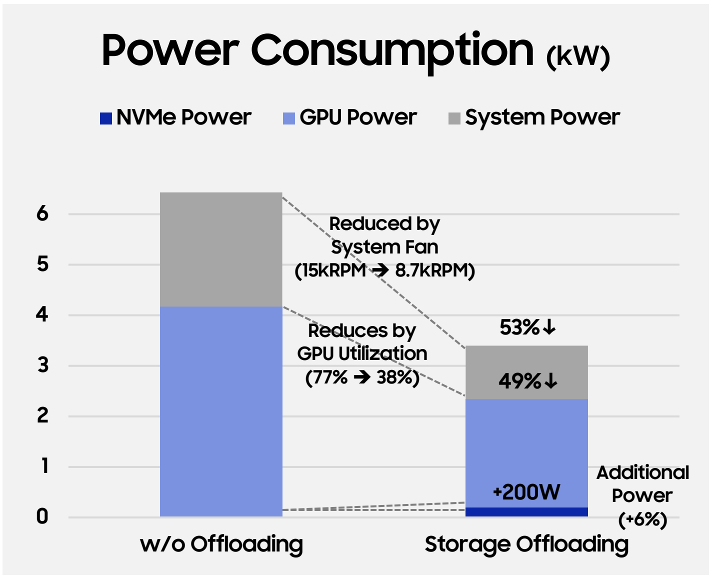


Figure 16. Power consumption rate in 8 x H100 GPU server

Conclusion

The experimental results show that, in LLM service environments, most I/O operations involve large blocks of 1 MB or more. Although each inference process issues predominantly sequential read requests, concurrent activity across multiple processes results in globally interleaved access patterns. Consequently, the storage subsystem must handle what appears to be randomized large-block reads despite underlying per-process sequentiality. These observations highlight two essential requirements for the storage layer:

- High sustained throughput
- Robust detection and preservation of sequential access patterns under concurrency

Both are critical determinants of overall system competitiveness.

From a performance perspective, the LLaMA-3.1-8B model operating across eight inference services sustained an average KV cache throughput of approximately 36 GB/s, indicating that the workload does not inherently demand extreme continuous bandwidth. However, instantaneous throughput frequently exceeded 2x the average, reflecting substantial variability and emphasizing the need for storage systems capable of absorbing transient I/O bursts.

Under these conditions, an architecture employing KV cache storage offloading provides benefits beyond simple TTFT optimization:

- Supports 1.7× more concurrent users on the same hardware
- Generates 1.5× more output tokens by reducing repeated prefill computation

These results demonstrate that KV cache storage offloading is a core mechanism for enabling scalable and stable QoS in large-scale LLM deployments.

From a system-efficiency standpoint, the offloading approach significantly reduces compute load during decoding, lowering average GPU utilization and overall power consumption to roughly half of the non-offloading configuration. This translates to:

- 1.5× higher cost efficiency
- Up to 2.9× higher energy efficiency

Beyond direct efficiency gains, these improvements enhance datacenter thermal density, power efficiency, and long-term operational sustainability—factors increasingly critical in modern high-density LLM inference clusters.

Taken together, these findings illustrate that as LLM-driven AI continues to evolve toward more dynamic, multi-agent, and real-time Agentic AI workloads, KV cache storage offloading becomes a foundational infrastructure requirement rather than an optional optimization. The results further validate that the PM1753 is well-aligned with the I/O patterns and performance demands of offloading-centric architectures, positioning it as a strong fit for next-generation large-scale LLM inference systems.

Technical specifications

Implementation notes

	KV Cache Storage Offloading Test Platform
Server System	Supermicro® SYS-522GA-NRT
CPU	2x Intel® 6th generation Xeon 6952 (96-core, 192 Thread)
Memory	768GB
GPU	8x NVIDIA PCIe H100 NVL (96GB)
SSDs	10x Samsung PM1753 NVMe SSD 7.68TB
OS	Ubuntu 22.04.2 LTS
Kernel	linux-image-6.5.0-1024-nvidia
NVIDIA Driver / CUDA	nvidia-open-570.86.15-0ubuntu1, cuda-tools-12-8, nvidia-gds-12-8
Inference Service Software	vllm-0.9.2, lmcache-0.3.3, uvicorn-0.34.3, fastapi-0.116.0
Benchmark Tools	LMBenchmark

1. 1TB = 1,000,000,000,000 Bytes, 1GB = 1,000,000,000 Bytes. Actual usable capacity may be less (due to formatting, partitioning, operating system, applications or otherwise).
2. Version 0.3.3-0.3.8 of LMCache support the evaluation configuration used in this whitepaper, including the GDS backend.
   However, staring from version 0.3.9, changes in the KV cache block file management mechanism prevent the same evaluation methodology (using GDS backend) from being applied.
3. Power consumption was measured using IPMI for server-level data and nvidia-smi for GPU-level data.