ISSN: 2473-4901
v12 n6815

Storage-Backed Artificial Intelligence Inference: A Tiered-Memory Architecture for Extending Large Language Model Execution Beyond GPU VRAM

Francis X. Hartle III
hartle@rmu.edu
Robert Morris University
Moon Township, PA 15108 USA

Abstract

Large language models have grown rapidly in size, often beyond the video random access memory (VRAM) on the graphics processing units (GPUs) that most researchers, educators, and small organizations use or can afford. This gap limits experimentation and narrows access to advanced artificial intelligence work. Prior research has shown that heterogeneous memory, model offloading, disk-backed inference, and key-value cache paging can extend large-model execution past GPU memory limits. Building on that work, this paper presents a practical storage-backed inference framework for commodity hardware. The framework treats GPU VRAM as a hot cache, system random access memory (RAM) as a staging (warm) layer, non-volatile memory express (NVMe) solid-state drive (SSD) storage as a backing store, and hard drive storage as an archival tier. The design focuses on inference rather than training. It organizes three coordinated mechanisms into a benchmarkable prototype: asynchronous layer streaming, double-buffered GPU execution, and paged key-value cache management. The goal is not to claim that storage can replace VRAM or that offloading is new. The goal is to identify the hardware, model-size, and workload conditions under which storage-backed inference becomes practical rather than merely possible. The paper presents the system architecture, scheduling model, expected bottlenecks, evaluation methodology, and research questions needed to evaluate storage-augmented large language model inference.

Keywords: Large Language Models, Tiered Memory, Inference Offloading, GPU Memory, KV Cache, NVMe

Storage-Backed Artificial Intelligence Inference: A Tiered-Memory Architecture for Extending Large Language Model Execution Beyond GPU VRAM

Francis X Hartle III

1. INTRODUCTION

Large language model research has been shaped less by algorithmic novelty than by the steady, often unglamorous expansion of our physical or virtual computational abilities. Early transformer architectures fit on a single workstation GPU. Successive generations did not. By the time dense models with billions of parameters became standard reference points in literature, the practical question of where those parameters would reside in physical memory had become a defining constraint on who could conduct research (Aminabadi et al., 2022; Kwon et al., 2023; Sheng et al., 2023). A model whose weights, activations, temporary buffers, and key-value (KV) cache cannot fit in available VRAM cannot be executed locally, no matter how capable its architecture might be.

This memory problem narrows the population of institutions and researchers who can experiment with state-of-the-art systems. Educational, research, and small-business environments that lack large accelerator infrastructure are pushed toward smaller models or remote cloud services. Renting GPUs in the cloud has become cheaper, with datacenter-class hardware currently available for roughly four to five dollars per hour. Even so, hourly charges accumulate quickly against stagnant or shrinking institutional budgets, and price is not the only concern. Using a cloud service means sending prompts, documents, and datasets to an outside company, which raises concerns for institutions that handle student records and intellectual property. Running models locally keeps that information on hardware the institution controls (Rajbhandari et al., 2021; Ren et al., 2021). The constraint has not eased with newer hardware. Flagship consumer GPUs released in 2025 provide 32 gigabytes of VRAM, while mainstream cards remain at 12 to 16 gigabytes, even as widely used open-weight models have grown to hundreds of billions of parameters (NVIDIA, 2025a). Looking at just hardware architecture, one could logically ask whether hard drives or solid-state drives can be used as VRAM. In the strict hardware sense, the answer is no. GPU VRAM sits next to the GPU and provides bandwidth and latency that storage devices cannot match (NVIDIA, 2026). However, that narrow framing is the wrong question. A more useful research question is whether storage can serve as a lower tier in a managed memory hierarchy that extends effective model capacity beyond physical VRAM, in the same general way that prior heterogeneous-memory systems have demonstrated in both inference and training contexts (Rajbhandari et al., 2021; Sheng et al., 2023).

This paper examines that possibility. It also acknowledges that model offloading, heterogeneous memory, and disk-backed inference are established research areas with mature literatures. The central premise is that storage-backed artificial intelligence inference should not try to make a hard drive behave like VRAM. Instead, the system should treat VRAM as a scarce hot cache, system RAM as a staging and buffering layer, NVMe SSD storage as a backing store for cold model tensors and cold KV pages, and hard drives as archival storage for model checkpoints, datasets, and infrequently accessed weights. This four-tier hierarchy is consistent with prior work showing that GPU, CPU, and disk resources can be jointly scheduled for large-model execution when tensor placement and transfer costs are explicitly managed (Rajbhandari et al., 2021; Sheng et al., 2023). The proposed architecture is designed around the predictable structure of transformer inference. Layers are processed in order during each forward pass, and KV-cache access patterns can be managed through paging policies (Kwon et al., 2023).

2. SCOPE AND CONTRIBUTION

This paper does not claim to introduce offloading, paging, or heterogeneous memory as new concepts. FlexGen, ZeRO-Infinity, DeepSpeed-Inference, ZeRO-Offload, and PagedAttention have already shown important methods for extending large-model execution beyond GPU memory limits (Aminabadi et al., 2022; Kwon et al., 2023; Rajbhandari et al., 2021; Ren et al., 2021; Sheng et al., 2023). The present

work is positioned as an applied systems design and evaluation study that translates those ideas into a practical, modular, commodity-hardware architecture.

The proposed contribution has five parts. First, the paper defines an explicit four-tier memory hierarchy that separates GPU VRAM, system RAM, NVMe SSD, and hard drive archival storage according to expected access frequency and latency sensitivity. Second, it describes an asynchronous layer-streaming mechanism for transformer inference that treats VRAM as an execution cache rather than a full model container. Third, it presents a double-buffering strategy intended to overlap storage input and output, RAM staging, GPU transfer, and GPU computation. Fourth, it incorporates a paged KV-cache hierarchy inspired by prior paging approaches and extended across VRAM, RAM, and SSD tiers. Fifth, it proposes an evaluation methodology for distinguishing the boundary between theoretically possible storage-backed inference and practically useful storage-backed inference under realistic workload conditions.

This paper presents a proposed design and a plan for testing it and no working prototype or measured results are claimed. Building and benchmarking the system is the next phase of this research, using the metrics defined in Section 6.

3. BACKGROUND AND MOTIVATION

3.1 The GPU Memory Wall

Large artificial intelligence models require substantial memory for parameters, activations, optimizer states during training, and KV cache during inference. Training is especially memory intensive because gradient tensors and optimizer states must be retained and updated, which motivated systems such as ZeRO-Offload and ZeRO-Infinity (Rajbhandari et al., 2021; Ren et al., 2021). Inference removes many training-only memory requirements; however, it still requires resident model weights, temporary activations, and the growing KV cache used in autoregressive generation (Kwon et al., 2023; Sheng et al., 2023).

GPU memory growth has not kept pace with the growth of model scale. Rajbhandari et al. (2021) described this as a GPU memory wall and proposed the heterogeneous use of GPU, CPU, and NVMe memory to support extreme-scale deep learning. That work focused on training, but the same memory-capacity problem also applies to inference when model weights and KV cache exceed available VRAM. The result is a stratified research environment. The largest open-weight models are nominally available to the public, yet they remain un-runnable on the hardware that most of the public owns.

3.2 Why Storage Cannot Directly Replace VRAM

Storage differs from VRAM in three ways: bandwidth, latency, and access semantics. VRAM is built for high-throughput parallel access by GPU cores. Storage systems are built for capacity and persistence rather than direct high-frequency tensor access (NVIDIA, 2026). NVMe SSDs are fast compared with older storage devices, but they remain orders of magnitude slower than VRAM for active GPU computation. Current PCIe 5.0 NVMe drives sustain roughly 14 gigabytes per second in sequential reads, so a four-bit quantized 70-billion-parameter model of roughly 40 gigabytes can be read from storage in about three seconds. The same generation of flagship GPUs, however, moves data within VRAM at roughly 1.8 terabytes per second, more than two orders of magnitude faster (NVIDIA, 2025a). Hard drives are slower still, especially for random access. GPU kernels also expect active data to reside in GPU-accessible memory. Repeatedly fetching small tensor fragments from storage during execution would cause GPU stalls and reduce utilization to levels that defeat the purpose of using a GPU. This is why offloading systems use scheduling, batching, and prefetching rather than naive demand loading (Rajbhandari et al., 2021; Sheng et al., 2023).

A workable design must avoid fine-grained demand fetching from storage during GPU computation. The system must move large, predictable tensor blocks ahead of time. It must overlap data transfer with computation. It must keep the working set in VRAM whenever it is needed. This design principle aligns with prior heterogeneous execution systems that improve feasibility by optimizing tensor placement and data movement across memory tiers (Aminabadi et al., 2022; Sheng et al., 2023).

3.3 Related Work

Several prior systems demonstrate that heterogeneous memory can enable larger models on constrained hardware. FlexGen showed that large language model inference can aggregate GPU, CPU, and disk resources, using compression and optimized tensor placement to improve throughput under limited GPU memory (Sheng et al., 2023). DeepSpeed-Inference explored heterogeneous inference that uses CPU and NVMe memory in addition to GPU memory and compute for models that do not fit entirely in aggregate GPU memory (Aminabadi et al., 2022). ZeRO-Offload and ZeRO-Infinity demonstrated that CPU and NVMe offloading can extend training scale beyond GPU memory limits (Rajbhandari et al., 2021; Ren et al., 2021). The vLLM project and its underlying PagedAttention mechanism showed that KV-cache memory can be managed using paging concepts borrowed from operating systems, which reduces fragmentation and improves serving throughput (Kwon et al., 2023). NVIDIA GPUDirect Storage provides a technical foundation for more efficient data movement between storage and GPU memory by reducing CPU-mediated copies (NVIDIA, 2025b).

More recent systems have moved offloading research from the datacenter toward consumer and edge devices. Alizadeh et al. (2024) stored model weights in flash memory and built an inference cost model around flash bandwidth and latency, using activation sparsity, a sliding window of recently activated neurons, and row-column bundling to favor large contiguous reads. PowerInfer exploited the power-law distribution of neuron activations to keep hot neurons on a consumer GPU while computing cold neurons on the CPU (Song et al., 2024), and PowerInfer-2 extended the approach to smartphones by overlapping storage reads with computation through a fine-grained pipeline (Xue et al., 2024). FlexInfer offloads weights to SSD with asynchronous tensor-level prefetching and a policy that keeps part of each layer resident in order to smooth input and output load (Du et al., 2025). NEO showed that the CPU can serve as a compute tier rather than only a staging tier by running decode-time attention on the host processor (Jiang et al., 2025). In open-source practice, llama.cpp popularized memory-mapped weight loading with partial GPU offload (Gerganov, 2023), and AirLLM demonstrated naive sequential layer streaming from disk (Li, 2023). These tools are widely used on commodity hardware, but they lack coordinated prefetching across a full storage hierarchy.

A parallel line of work manages the KV cache, rather than model weights, across memory tiers. InfiniGen keeps the full cache in host memory and speculatively prefetches only the entries most likely to matter for the next attention computation (Lee et al., 2024). CachedAttention maintains a hierarchical KV cache across GPU memory, host memory, and SSD, with layer-wise prefetching overlapped with computation for multi-turn conversations (Gao et al., 2024). CacheGen compresses KV pages into compact bitstreams so that cold cache can be stored and streamed economically (Liu et al., 2024). At datacenter scale, Mooncake pools underutilized DRAM and SSD capacity across a cluster into a disaggregated KV cache, explicitly trading storage for computation (Qin et al., 2025), and the NVIDIA Dynamo framework now includes a cache manager that pages KV blocks across GPU memory, host memory, local SSD, and remote storage (NVIDIA, 2025c). Industry serving stacks have therefore already converged on tiered hierarchies much like the one proposed here, although they target the KV cache in datacenter deployments rather than weights and cache together on a single machine.

Mixture-of-experts models have proven especially amenable to offloading because only a subset of experts is activated for each token. Fiddler executes non-resident experts directly on the CPU instead of transferring them across the PCIe bus (Kamahori et al., 2025). KTransformers places attention computation on the GPU and expert weights in CPU memory with processor-optimized kernels, allowing models with hundreds of billions of parameters to run on a single consumer GPU paired with a desktop processor (Chen et al., 2025). Broader surveys document the rapid growth of this literature (Zhou et al., 2024).

The present paper builds on these foundations. Each mechanism in the proposed design has prior art: layer streaming, asynchronous prefetching, KV paging, and tiered caching all appear in the systems above. What the literature does not yet provide is an integrated treatment of model weights and KV cache together across all four tiers, including the archival hard drive tier, on a single commodity machine, accompanied by an evaluation framework for locating the boundary where such execution becomes practical. The closest existing systems each solve a piece of this problem. FlexGen spreads work across GPU, CPU, and disk but is built for large batch jobs rather than individual use.

CachedAttention and NVIDIA's Dynamo tier the conversation cache across memory and storage, but not the model itself. PowerInfer and KTransformers achieve impressive results on ordinary hardware, but only for models with special internal structure they can exploit. The present design is different in three ways. It manages both the model and its cache under one scheduler, it uses all four tiers down to ordinary hard drives, and it works with standard models on unmodified consumer hardware. The present design targets that gap in a form that educational institutions and small organizations can reproduce on hardware they already own.

4. RESEARCH PROBLEM

Under what hardware, model-size, and workload conditions does storage-backed large language model inference become practically useful rather than merely possible when using a tiered memory hierarchy of VRAM, system RAM, NVMe SSD, and hard drive archival storage?

This question yields six sub-questions. First, which model components must remain resident in VRAM to avoid unacceptable slowdown? Second, which tensors can be streamed from RAM or SSD without severely stalling GPU execution? Third, how far ahead must the runtime prefetch layers be in order to hide transfer latency? Fourth, how should KV-cache pages be promoted, demoted, or evicted across memory tiers? Fifth, what role, if any, can hard drive storage play beyond archival storage and initial model loading? Sixth, which workloads are appropriate for storage-backed inference, and which workloads require fully resident GPU execution?

The working hypothesis is that storage-backed inference may be practical only when the system uses predictable layer streaming, aggressive quantization, asynchronous transfers, and careful KV-cache paging together. The architecture may be most useful for batch-oriented, offline, educational, experimental, or latency-tolerant inference. It may not be useful for real-time interactive generation at high token rates.

5. PROPOSED ARCHITECTURE

Building on prior work in heterogeneous inference and offloading, the proposed system uses four memory tiers. Each tier is assigned a role based on access frequency and latency sensitivity (Figure 1). GPU VRAM is the fastest and smallest tier. It stores the active transformer layer or layers, the next-layer buffer, temporary activations, and the hottest portion of the KV cache. VRAM should be reserved for data that will be accessed immediately by GPU kernels, consistent with prior work showing that scarce GPU memory must be allocated to the most performance-critical tensors (Kwon et al., 2023; Sheng et al., 2023).

System RAM serves as the staging layer. It holds prefetched layers, pinned memory buffers, older KV-cache blocks, decompressed or partially decompressed tensors, and transfer queues. RAM is slower than VRAM, but it is much faster and more flexible than storage. This makes it useful as an intermediate tier in heterogeneous memory systems (Rajbhandari et al., 2021; Ren et al., 2021).

NVMe SSD storage serves as the backing store. It holds the full quantized model, cold model layers, cold KV-cache pages, and tensor blocks that are too large to remain in RAM. The SSD should be accessed through large sequential or batched reads whenever possible. Storage-backed tensor execution is most effective when transfer overhead is amortized across larger data movements (NVIDIA, 2025b; Rajbhandari et al., 2021; Sheng et al., 2023).

Hard drive storage serves as the archival tier. It holds model archives, checkpoints, datasets, and rarely used model versions. In the proposed architecture, the hard drive is not treated as an active real-time VRAM substitute. Its role is to support capacity, not immediate execution.

ISSN: 2473-4901
v12 n6815

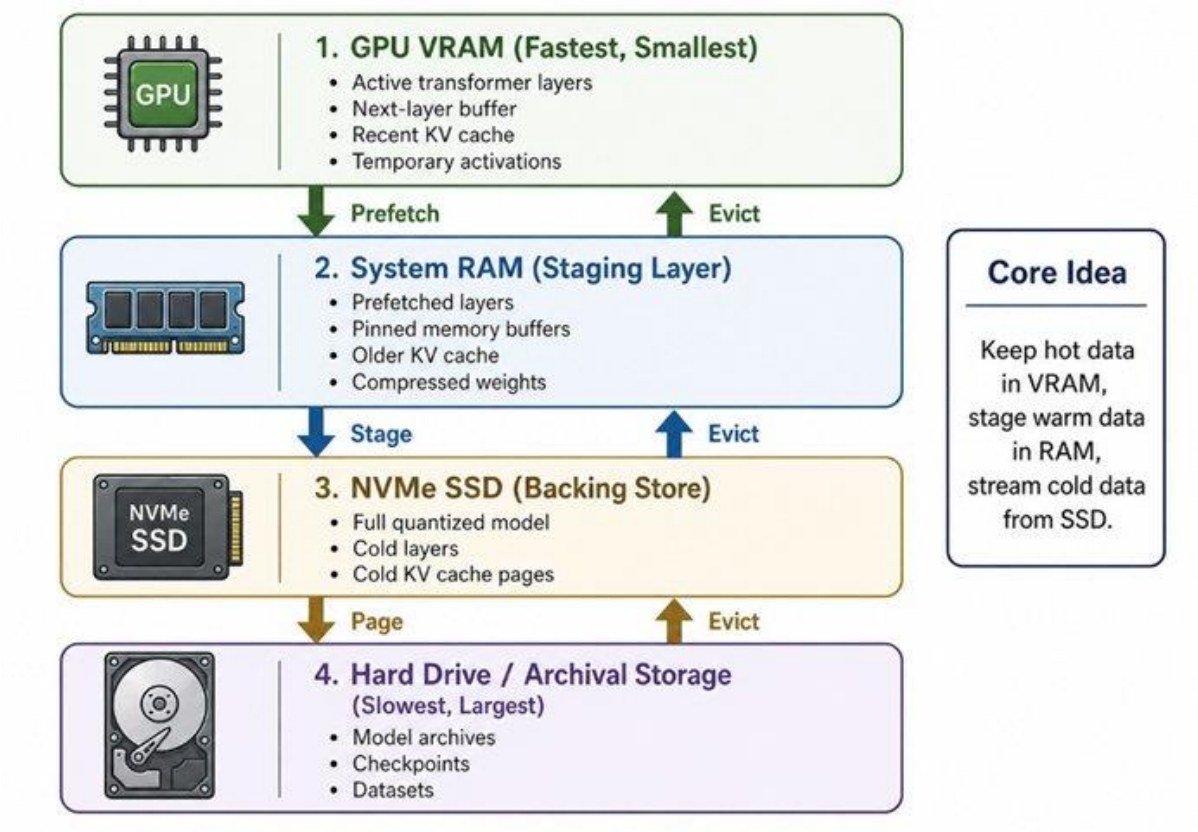

Figure 1. Four-Tier Memory Hierarchy for Storage-Backed Inference

5.1 Asynchronous Layer Streaming

Transformer inference has a useful structural property. During a forward pass, layers execute in a predictable sequence. If layer n is executing now, the system knows that layer n + 1 and layer n + 2 will soon be needed. This predictability enables layer streaming. It resembles the tensor-placement strategies used by systems that coordinate GPU, CPU, and disk memory for large-model inference (Aminabadi et al., 2022; Sheng et al., 2023).

In the proposed runtime, model layers are stored as independent or semi-independent tensor blocks. Before inference begins, the runtime loads the embedding layer, the output head where feasible, the first transformer layer or layers, and a small number of upcoming layers into VRAM and RAM. As each layer completes, the runtime evicts that layer from the active VRAM buffer if it will not be needed again during the current pass. At the same time, it asynchronously transfers upcoming layers from SSD to RAM and from RAM to VRAM.

The scheduler should avoid small random reads. It should group layer tensors into contiguous files or storage-aligned blocks. Quantized weights should be stored in a format that can be quickly transferred and decoded. If decompression is required, it should occur during the RAM staging phase or in GPU kernels designed to consume quantized weights directly.

5.2 Double-Buffered GPU Execution

Double buffering is used to overlap computation and data movement (Figure 2). The system allocates at least two VRAM buffers for layer weights. Buffer A holds the current layer being computed. Buffer B receives the next layer asynchronously. When computation on Buffer A completes and Buffer B is ready, the runtime swaps the roles of the buffers. If sufficient VRAM is available, a third buffer may be used for additional prefetch depth. This approach follows the general principle that heterogeneous-memory

ISSN: 2473-4901
v12 n6815

inference must overlap communication with computation in order to reduce GPU idle time (Aminabadi et al., 2022; Sheng et al., 2023).

This mechanism is central to the architecture. If the GPU completes one layer and the next layer is not yet resident in VRAM, execution stalls. The goal of the scheduler is to ensure that transfers complete before the GPU reaches the next layer. When stalls do occur, the system has either underestimated transfer cost, overestimated computational cost, or selected an inadequate prefetch depth.

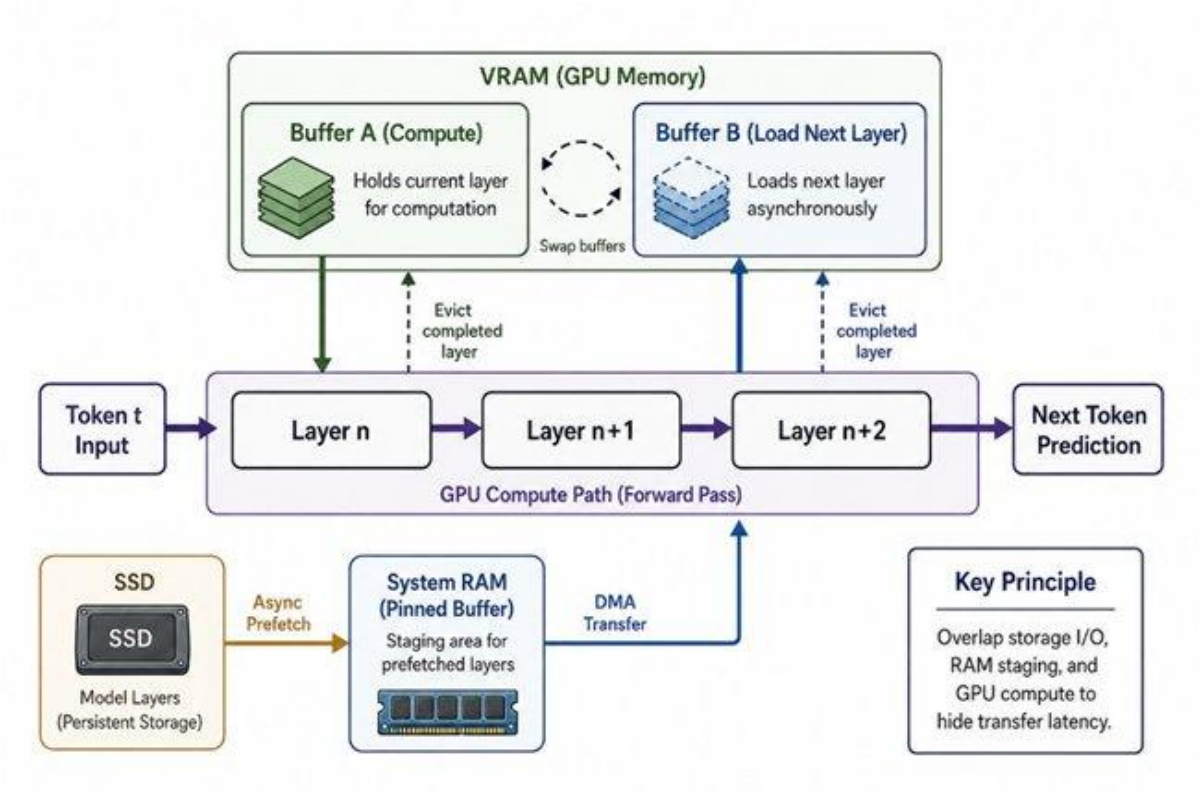

Figure 2. Double-Buffered GPU Execution Timeline

5.3 Paged Key-Value Cache Management

During autoregressive inference, each generated token adds key and value tensors to the KV cache. For long-context prompts, the KV cache can become a dominant memory consumer. Kwon et al. (2023) showed that KV-cache memory management is a central challenge in large language model serving and that paging concepts can reduce waste while improving serving throughput. If the full KV cache remains in VRAM, it reduces the space available for model layers and activations. If it is moved entirely to storage, attention computation becomes too slow.

The proposed system uses a paged KV-cache hierarchy (Figure 3). Recent KV pages remain in VRAM. They are most likely to be accessed by attention operations, and recent context is typically central to ongoing generation. Medium-age KV pages are stored in RAM. Older or lower-priority KV pages are stored on SSD. The runtime promotes pages when attention requires them and demotes pages when they become less active.

The policy is inspired by operating-system virtual memory and by the use of noncontiguous KV-cache blocks in PagedAttention to improve memory management in large language model serving (Kwon et al., 2023). The policy must be adapted to transformer attention. Unlike general-purpose memory systems, the runtime has semantic knowledge of token positions, sequence lengths, request batching, and attention windows. This knowledge can inform promotion and eviction policies. Possible policies include recency-based paging, sliding-window retention, attention-score-informed retention, request-

priority-aware paging, and hybrid policies that reserve VRAM for recent pages while using RAM or SSD for distant context.

Standard transformers are the hardest, because each new token can draw on the entire conversation held in the KV cache. The design copes in three ways: cache entries are grouped into pages, so one SSD read serves many lookups, pages are requested a step ahead, so loading overlaps computation (Lee et al., 2024); and rarely used pages are stored compressed (Liu et al., 2024). Very long conversations that draw evenly on their full history still force the cache back into RAM. Locating that boundary is an explicit goal of the evaluation.

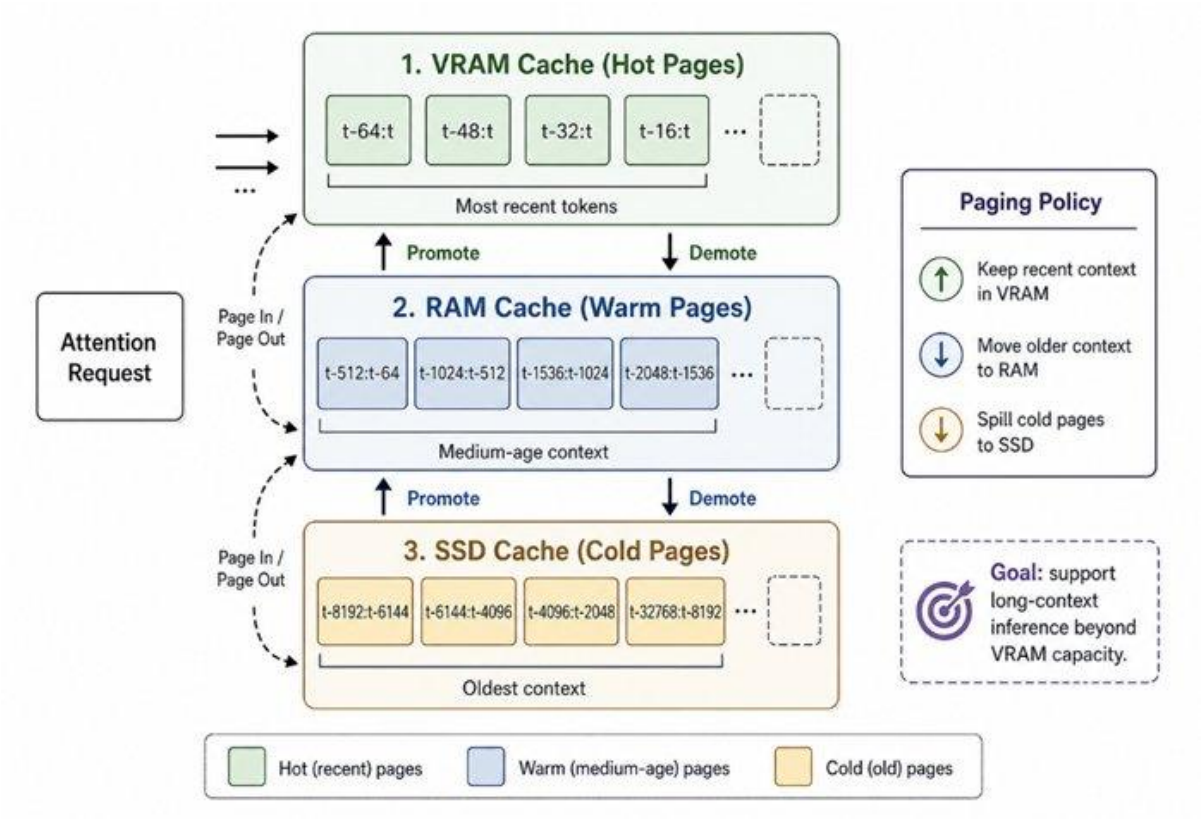

Figure 3. Paged KV-Cache Hierarchy Across VRAM, RAM, and SSD

5.4 System Design

The proposed runtime contains five major components. The memory manager tracks the location and status of every tensor block. It records whether a layer, activation buffer, or KV page is resident in VRAM, RAM, SSD, or archival storage. It also records whether the block is compressed, quantized, transferred, pending transfer, or ready for compute.

The prefetch scheduler predicts upcoming tensor demand. For dense transformer layers, this prediction is straightforward, because layer order is known in advance. For KV-cache pages, demand prediction depends on context length, attention pattern, decoding strategy, and workload batching. It is therefore inherently approximate. The eviction manager removes data from VRAM when it is no longer needed. For layer weights, the basic policy is to evict layers that have completed execution for the current token. For KV pages, eviction should consider recency, attention relevance, request priority, and current VRAM pressure.

The transfer engine coordinates asynchronous movement between SSD, RAM, and VRAM. It should use pinned memory where appropriate, support large batched reads, and exploit direct storage paths when available. Technologies such as GPUDirect Storage are relevant because they reduce CPU-mediated

copies in storage-to-GPU data paths (NVIDIA, 2025b). GPUDirect Storage remains limited to datacenter-class GPUs and file systems, however, and consumer-oriented direct storage interfaces have seen limited adoption outside gaming, so a commodity design should assume CPU-mediated transfers staged through pinned RAM buffers. The transfer engine should expose completion events so the scheduler can avoid dispatching GPU kernels before required weights are ready. Finally, the quantization and decoding layer reduces storage and transfer volume. Four-bit and eight-bit weight formats may substantially reduce the data that must be moved. FlexGen demonstrated the importance of compression and careful tensor placement when using limited GPU memory and slower backing tiers for generative inference (Sheng et al., 2023). The trade-off is that quantization can introduce quality degradation and may require additional compute for decoding or specialized kernels.

5.5 Algorithmic Sketch

A simplified inference loop proceeds in five stages. First, the model is partitioned into layer blocks and KV-cache pages. Model weights are quantized and stored on NVMe SSD. Frequently used components, including embeddings, normalization parameters, and the output head, are assigned a preferred residency level.

Second, before generation begins, the runtime loads the initial layer block into VRAM and prefetches several upcoming layer blocks into RAM. It allocates double buffers in VRAM and pinned transfer buffers in RAM.

Third, for each token, the runtime executes the transformer stack one layer at a time. While the GPU computes the current layer, the transfer engine loads the next layer into the inactive VRAM buffer. When the current layer completes, the scheduler swaps buffers, evicts completed layers, and advances the prefetch window.

Fourth, the runtime updates the KV cache. Recent KV pages remain in VRAM. Older pages are demoted to RAM or SSD depending on memory pressure and anticipated reuse. If attention requires a nonresident page, the scheduler issues a page-in request, preferably batched with other needed pages.

Fifth, the model produces the next-token prediction, and the process repeats until generation completes.

6. PROPOSED EVALUATION METHOD

The evaluation should compare four configurations: fully resident GPU inference, CPU-offloaded inference, SSD-backed inference, and hard-drive-assisted archival loading. The principal performance metrics should include tokens per second, first-token latency, average per-token latency, GPU utilization, PCIe bandwidth utilization, SSD read bandwidth, RAM bandwidth, page fault frequency, prefetch hit rate, and output quality under quantization. These metrics are consistent with how prior systems evaluate heterogeneous inference (Aminabadi et al., 2022; Kwon et al., 2023; Sheng et al., 2023). The evaluation should also include existing offloading runtimes as baselines, including llama.cpp with partial GPU offload, FlexGen, and naive sequential layer streaming (Du et al., 2025; Gerganov, 2023; Li, 2023; Sheng et al., 2023).

The evaluation should test several model sizes and hardware profiles. A practical test matrix might include a small model that fits fully in VRAM, a medium model that exceeds VRAM but fits mostly in RAM, and a larger model that requires SSD-backed streaming. Hardware profiles should include consumer GPU configurations, workstation GPUs, varied RAM capacities, PCIe 3.0 versus PCIe 4.0 or 5.0 NVMe drives, and optional direct storage paths.

Workloads should include short prompts, long-context prompts, single-request interactive inference, and batch-oriented inference. The architecture is expected to perform better in batched and latency-tolerant workloads, because more computation can be overlapped with input and output. Table 1 summarizes the proposed metrics. Metrics should be collected across fully resident, RAM-offloaded, SSD-backed, and archival-storage-assisted configurations to enable direct comparison.

Metric	Purpose	Expected Relevance
Tokens per second	Measures generation throughput	Determines practical usability
First-token latency	Measures initial response delay	Captures loading and prefill cost
Average per-token latency	Measures steady-state decoding speed	Reveals effect of repeated streaming
GPU utilization	Measures whether the GPU is idle	Indicates success of prefetching
SSD read bandwidth	Measures storage pressure	Identifies storage bottlenecks
PCIe bandwidth utilization	Measures transfer pressure	Identifies bus bottlenecks
Prefetch hit rate	Measures scheduler effectiveness	Predicts stall reduction
KV page fault frequency	Measures paging efficiency	Identifies long-context bottlenecks
Output quality	Measures quantization impact	Ensures throughput gains do not mask quality loss

Table 1. Proposed Evaluation Metrics for Storage-Backed Inference

Several testable hypotheses follow from the architecture. Layer streaming should enable larger-than-VRAM inference, but it should also reduce token throughput compared with fully resident execution. Prefetch depth and double buffering should reduce GPU idle time. SSD-backed execution should be viable for latency-tolerant workloads. Hard-drive-backed active execution should generally be impractical except for archival loading or offline preprocessing. Quantization should be essential because it reduces both storage footprint and transfer volume. Paged KV-cache management should improve long-context capacity, but it may introduce latency spikes when cold pages must be retrieved.

7. EXPECTED LIMITATIONS

The proposed design has several limitations. First, the system is unlikely to support high-performance real-time interactive inference for very large dense models if most weights must be streamed from SSD for every token. Second, hard drives are not suitable for active tensor streaming except in extremely slow, offline, or archival workflows. Third, the system may be highly sensitive to PCIe bandwidth, SSD performance, file layout, and GPU kernel scheduling. Benchmark results may not generalize cleanly across hardware platforms. Fourth, long-context inference can trigger unpredictable KV-cache page-ins that produce visible latency spikes. Fifth, aggressive quantization may degrade model quality in ways that are workload-specific.

8. SECURITY, RELIABILITY, AND DATA INTEGRITY

Storage-backed inference introduces reliability concerns that fully resident inference does not. If model weights and KV pages are continuously streamed from storage, data corruption, incomplete transfers, or synchronization errors could affect model output in ways that are difficult to detect at the application layer. The runtime should verify tensor block integrity using checksums or hashes, maintain metadata consistency across tier transitions, and recover safely from failed reads. Security considerations follow as well. KV cache may contain sensitive user prompt data. If KV pages are spilled to RAM or SSD, the system should consider encryption, secure deletion on eviction, access control, and isolation between users or requests. In multiuser environments, cache reuse and paging policies must prevent leakage across sessions.

9. DISCUSSION

The central insight of this applied systems paper is that the useful comparison is not between hard drives and VRAM. The useful comparison is between unmanaged memory failure and managed performance degradation. If a model cannot fit in VRAM, the practical choice is often either not running it at all or running it through a carefully scheduled hierarchy. Prior systems have shown that this trade-off can be useful when tensor placement, compression, and offloading are optimized rather than treated as conventional swapping (Rajbhandari et al., 2021; Sheng et al., 2023). The proposed approach accepts slower performance in exchange for expanded feasibility.

The most promising implementation path is not direct hard-drive-as-VRAM emulation. The practical path may be NVMe-backed inference with RAM staging, quantized layer streaming, and paged KV-cache management. This approach is closer to the FlexGen use of disk as part of a planned inference hierarchy and to the explicit management of KV-cache memory in PagedAttention than it is to traditional operating-system swapping (Kwon et al., 2023; Sheng et al., 2023). Hard drives remain useful for storing model archives, datasets, checkpoints, and inactive versions of weights, but not for active per-token execution.

A further opportunity is mixture-of-experts inference. In dense transformer models, every layer is used for every token. In mixture-of-experts models, only selected experts are activated. This selective activation creates more favorable conditions for storage-backed execution, because rarely used experts can remain in RAM or SSD while frequently used experts stay in VRAM. Recent systems support this expectation. Fiddler executes non-resident experts on the CPU (Kamahori et al., 2025), and KTransformers runs mixture-of-experts models with hundreds of billions of parameters on a single consumer GPU paired with a desktop processor (Chen et al., 2025). Extending expert placement downward into the SSD tier is a natural next step for the architecture proposed here.

10. CONCLUSIONS

Hard drives cannot serve as true VRAM, because they lack the bandwidth, latency, and access semantics required by GPU computation (NVIDIA, 2026). However, storage may extend the effective memory capacity of artificial intelligence inference systems when used as part of a tiered architecture, as shown by prior work on heterogeneous GPU, CPU, and disk execution (Rajbhandari et al., 2021; Sheng et al., 2023). This paper proposed a storage-backed inference runtime that places hot data in VRAM, stages warm data in RAM, streams cold model layers from NVMe SSD, and uses hard drives as archival storage. Through asynchronous layer streaming, double-buffered execution, quantization, and paged KV-cache management, the system may enable larger-than-VRAM model execution on resource-constrained hardware (Kwon et al., 2023; Sheng et al., 2023).

The goal is not to replace high-end GPU memory or to claim that offloading is a new concept. The goal is to expand access to larger models by making limited hardware more useful through a carefully designed implementation. A potential benefit is the ability to run and study larger models on existing laboratory hardware, which could support coursework and student research while reducing recurring cloud costs; verifying this benefit is an explicit goal of the planned evaluation. Future research should implement the proposed runtime, quantify its performance across model sizes and hardware configurations, compare it directly with existing offloading approaches, and identify the workload boundaries at which storage-backed inference becomes practical.

11. ACKNOWLEDGEMENTS

The author acknowledges the use of a generative artificial intelligence tool (Anthropic Claude) in preparing this manuscript. AI assistance was used in four capacities: identifying and locating relevant research literature, verifying and correcting bibliographic references, creating figures, and editing prose and formatting to meet conference requirements. All research questions, the proposed architecture, interpretation of prior work, and scholarly conclusions are the sole work of the author.

12. REFERENCES

Alizadeh, K., Mirzadeh, S. I., Belenko, D., Khatamifard, S., Cho, M., Del Mundo, C. C., Rastegari, M., & Farajtabar, M. (2024). LLM in a flash: Efficient large language model inference with limited memory. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 12562–12584. https://doi.org/10.18653/v1/2024.acl-long.678

Aminabadi, R. Y., Rajbhandari, S., Awan, A. A., Li, C., Li, D., Zheng, E., Ruwase, O., Smith, S., Zhang, M., Rasley, J., & He, Y. (2022). DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. https://doi.org/10.1109/SC41404.2022.00051

Chen, H., Xie, W., Zhang, B., Tang, J., Wang, J., Dong, J., Chen, S., Yuan, Z., Lin, C., Qiu, C., Zhu, Y., Ou, Q., Liao, J., Chen, X., Ai, Z., Wu, Y., & Zhang, M. (2025). KTransformers: Unleashing the full potential of CPU/GPU hybrid inference for MoE models. *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, 1014–1029. https://doi.org/10.1145/3731569.3764843

Du, H., Wu, S., Kharlamova, A., Guan, N., & Xue, C. J. (2025). FlexInfer: Breaking memory constraint via flexible and efficient offloading for on-device LLM inference. *Proceedings of the 5th Workshop on Machine Learning and Systems*, 56–65. https://doi.org/10.1145/3721146.3721961

Gao, B., He, Z., Sharma, P., Kang, Q., Jevdjic, D., Deng, J., Yang, X., Yu, Z., & Zuo, P. (2024). Cost-efficient large language model serving for multi-turn conversations with CachedAttention. *2024 USENIX Annual Technical Conference*, 111–126. https://www.usenix.org/conference/atc24/presentation/gao-bin-cost

Gerganov, G. (2023). *llama.cpp* [Computer software]. GitHub. https://github.com/ggml-org/llama.cpp

Jiang, X., Zhou, Y., Cao, S., Stoica, I., & Yu, M. (2025). NEO: Saving GPU memory crisis with CPU offloading for online LLM inference. *Proceedings of Machine Learning and Systems, 7*. https://arxiv.org/abs/2411.01142

Kamahori, K., Tang, T., Gu, Y., Zhu, K., & Kasikci, B. (2025). Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=N5fVv6PZGz

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 611–626. https://doi.org/10.1145/3600006.3613165

Lee, W., Lee, J., Seo, J., & Sim, J. (2024). InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. *18th USENIX Symposium on Operating Systems Design and Implementation*, 155–172. https://www.usenix.org/conference/osdi24/presentation/lee

Li, G. (2023). *AirLLM* [Computer software]. GitHub. https://github.com/lyogavin/airllm

Liu, Y., Li, H., Cheng, Y., Ray, S., Huang, Y., Zhang, Q., Du, K., Yao, J., Lu, S., Ananthanarayanan, G., Maire, M., Hoffmann, H., Holtzman, A., & Jiang, J. (2024). CacheGen: KV cache compression and streaming for fast large language model serving. *Proceedings of the ACM SIGCOMM 2024 Conference*, 38–56. https://doi.org/10.1145/3651890.3672274

NVIDIA. (2025a). *GeForce RTX 5090 graphics cards*. https://www.nvidia.com/en-us/geforce/graphics-cards/50-series/rtx-5090/

NVIDIA. (2025b). *GPUDirect Storage overview guide*. https://docs.nvidia.com/gpudirect-storage/overview-guide/index.html

NVIDIA. (2025c). *Introducing NVIDIA Dynamo, a low-latency distributed inference framework for scaling reasoning AI models*. NVIDIA Developer Technical Blog. https://developer.nvidia.com/blog/introducing-nvidia-dynamo-a-low-latency-distributed-inference-framework-for-scaling-reasoning-ai-models/

NVIDIA. (2026). *CUDA programming guide*. https://docs.nvidia.com/cuda/cuda-programming-guide/

Qin, R., Li, Z., He, W., Cui, J., Ren, F., Zhang, M., Wu, Y., Zheng, W., & Xu, X. (2025). Mooncake: Trading more storage for less computation—A KVCache-centric architecture for serving LLM chatbot. *23rd USENIX Conference on File and Storage Technologies*, 155–170. https://www.usenix.org/conference/fast25/presentation/qin

Rajbhandari, S., Ruwase, O., Rasley, J., Smith, S., & He, Y. (2021). ZeRO-Infinity: Breaking the GPU memory wall for extreme scale deep learning. *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–14. https://doi.org/10.1145/3458817.3476205

Ren, J., Rajbhandari, S., Aminabadi, R. Y., Ruwase, O., Yang, S., Zhang, M., Li, D., & He, Y. (2021). ZeRO-Offload: Democratizing billion-scale model training. *2021 USENIX Annual Technical Conference*, 551–564. https://www.usenix.org/conference/atc21/presentation/ren-jie

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., & Zhang, C. (2023). FlexGen: High-throughput generative inference of large language models with a single GPU. *Proceedings of the 40th International Conference on Machine Learning, 202*, 31094–31116. https://proceedings.mlr.press/v202/sheng23a.html

Song, Y., Mi, Z., Xie, H., & Chen, H. (2024). PowerInfer: Fast large language model serving with a consumer-grade GPU. *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 590–606. https://doi.org/10.1145/3694715.3695964

Xue, Z., Song, Y., Mi, Z., Zheng, X., Xia, Y., & Chen, H. (2024). *PowerInfer-2: Fast large language model inference on a smartphone*. arXiv. https://arxiv.org/abs/2406.06282

Zhou, Z., Ning, X., Hong, K., Fu, T., Xu, J., Li, S., Lou, Y., Wang, L., Yuan, Z., Li, X., Yan, S., Dai, G., Zhang, X.-P., Dong, Y., & Wang, Y. (2024). *A survey on efficient inference for large language models*. arXiv. https://arxiv.org/abs/2404.14294