

Unified Memory Architecture: The Foundation for Sustained AI Inference

Authors: Kiran Matturi, Sharath Chandra Ambula, Tara Abarca

Executive Summary

Modern AI inference is increasingly limited by how systems organize and access memory, not by peak compute performance. While model weights are relatively fixed, the runtime state required for inference, including KV caches, activations, and multi-model execution, grows with context length, concurrency, and agentic workloads. In client and edge systems, this active memory footprint can quickly exceed the memory available to any individual compute engine. When that happens, performance becomes dominated by data movement and eviction cycles, resulting in sharp declines in throughput and efficiency.

Unified memory architecture (UMA) addresses this challenge by exposing system memory as a shared resource accessible by the CPU, GPU, and NPU. Rather than partitioning memory across isolated domains, UMA allows models and runtime state to be allocated once and accessed directly where needed. This reduces duplication, minimizes data movement, and transforms available system memory into usable memory for AI workloads.

As AI evolves toward longer context windows, multi-model pipelines, and agentic systems, system capability depends less on theoretical peak compute and more on the ability to keep active data resident and accessible. For client and mobile platforms, where compute engines share a common memory and power budget, UMA provides the practical foundation for running larger models, supporting richer experiences, and enabling sustained AI execution on-device.

Key Insights

AI inference is constrained by memory, not compute

As context lengths, concurrency, and agentic workloads grow, capacity and bandwidth increasingly determine sustained system performance.

Runtime state exceeds model memory by multiple factors

KV caches, activations, and intermediate tensors often consume significantly more memory than the model weights themselves under real inference workloads.

Memory limits introduce performance discontinuities

When active workloads exceed accelerator-local memory, systems shift from compute-bound execution to data-movement and eviction cycles, causing sharp throughput degradation.

UMA is a system architecture, not a memory technology

By exposing a unified address space across CPU, GPU, and NPU, UMA reduces memory fragmentation, eliminates unnecessary duplication, and improves access to available system memory.

Client and mobile AI amplify the value of UMA

In shared, power-constrained environments, the ability to keep larger amounts of runtime state resident and accessible becomes a key determinant of on-device AI capability.

1 Systems evolution toward heterogeneous compute

For modern AI inference, the way systems organize and access memory, not just peak compute, is becoming the primary determinant of sustained system performance. Modern client and edge computing systems have evolved from CPU-centric designs into heterogeneous compute platforms incorporating multiple specialized engines, including central processing units (CPU), graphics processing units (GPU), and neural processing units (NPU). These engines are designed to accelerate different classes of workloads, general-purpose compute, parallel numerical operations, and AI inference respectively.

Despite this diversity in compute capability, sustained AI performance increasingly depends on how efficiently CPUs, GPUs, and NPUs can access, share, and retain data in memory. Across AI inference workloads, memory organization has evolved from a supporting consideration to a defining system constraint.

2 Memory organization in contemporary systems

Contemporary heterogeneous systems were designed around fixed memory domains and predictable data movement patterns. While these designs have served traditional workloads well, they introduce structural constraints that become increasingly visible as working sets grow and span multiple compute engines.

Three constraints are particularly relevant:

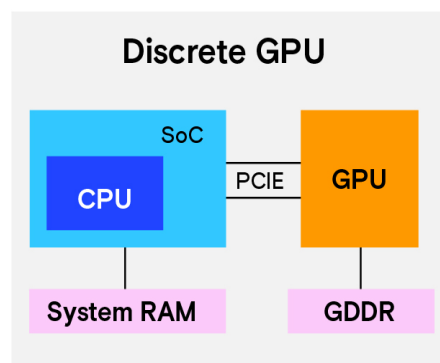
- Memory fragmentation and duplication across isolated memory pools reduce effective capacity.
- **Locality limitations**, where accelerator performance is constrained by finite local memory capacity.
- Performance overhead associated with data movement across memory domains becomes a dominant system bottleneck.

The extent to which these limitations appear varies across system architectures.

2.1 Discrete memory architecture

In discrete systems, CPU and GPU operate on separate physical memory pools:

Figure 1: Discrete memory architecture with CPU and GPU operating on separate physical memory pools connected through PCIe



- CPU accesses system DRAM, typically DIMM/LPDDR/LPCAMM technologies
- GPU accesses dedicated high-bandwidth memory (e.g., GDDR)
- Data crossing between the two requires explicit transfers over an interconnect (e.g., PCIe)

This architecture was well-suited for traditional workloads where data movement was predictable and bounded. However, it introduces fundamental inefficiencies:

- Data duplication across memory pools
- Latency and bandwidth overhead during transfers
- Limited effective working set constrained by GPU-local memory

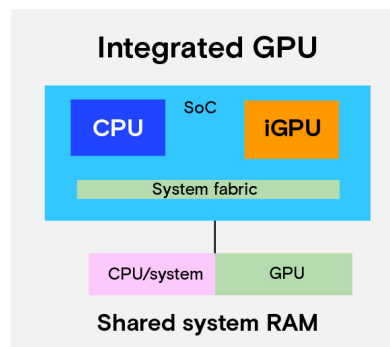
These constraints become particularly visible when working sets exceed accelerator memory capacity, forcing costly data movement cycles.

2.2 Shared memory systems (integrated GPU)

Integrated systems where GPU is part of SoC, address some of these inefficiencies by allowing CPU and GPU to share system DRAM. This eliminates explicit data copies between separate pools and improves memory utilization. GPU maximum share on the system will be predefined and beyond that GPU faces out of memory issues.

Mobile SoCs are the traditional home of this integrated, shared-memory model: the CPU, GPU, and NPU sit on a single die and share one LPDDR pool — its bandwidth and its capacity alike. Examples include Qualcomm's Snapdragon 8 Elite [13] and MediaTek's Dimensity 9400 [14].

Figure 2: Shared Memory Architecture with CPU and integrated GPU accessing a common system memory pool



However, shared memory systems retain important limitations:

- Memory access is typically mediated through a central control domain (CPU-owned)
- Bandwidth is shared and contested between compute engines
- Access patterns are not symmetric, limiting true concurrency

3 Nature of AI inference

For decades, heterogeneous memory hierarchies were effective because the workloads they supported were predictable. Graphics, media pipelines, and productivity applications exhibit structured access patterns, allowing tiered memory designs to operate efficiently.

AI-first workloads do not behave in the same way. Their working sets are both large and dynamic, especially when models reason iteratively, update context, and reuse intermediate results across CPU, GPU, and NPU. As a result, the system bottleneck is no longer dominated by arithmetic throughput, but by the overhead and hard limits imposed by isolated memory pools, which were not designed for coherent sharing at this scale.

AI inference workloads impose two fundamentally different types of memory demand: static memory and dynamic memory. Static memory consists of model weights and associated runtime buffers that are fixed and known at load time, while dynamic memory evolves continuously during execution through structures such as KV caches, activations, and intermediate tensors.

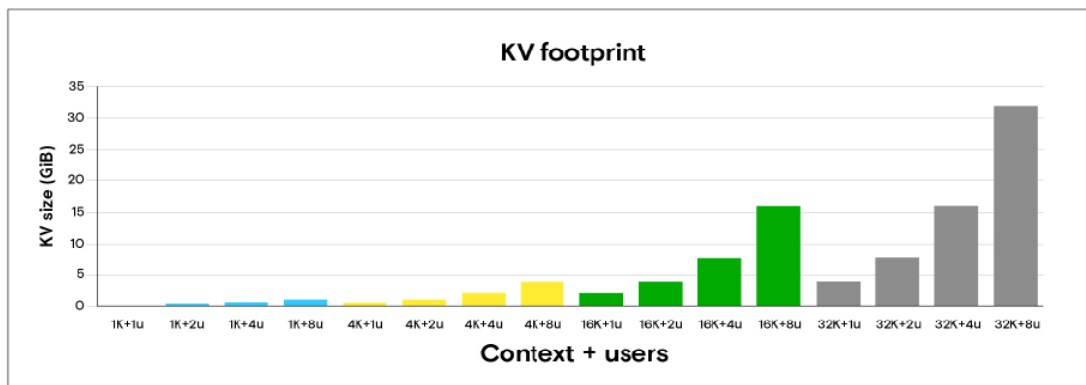
As a result, AI working set size is not statically bounded and cannot be efficiently provisioned within fixed memory domains. The total memory footprint depends not only on model size, but also on context length, concurrency, agent state, and runtime execution behavior.

In contrast, dynamic memory evolves during runtime. It includes structures such as the key-value (KV) cache, activations, and intermediate tensors, all of which scale with context length, batch size, and the number of concurrent users. Unlike static memory, this component cannot be precisely provisioned in advance, as it depends on real-time workload behavior. Multiple concurrent users share the same model weights, but each maintains its own KV cache; dynamic memory therefore scales with the number of users, their batch sizes, and context length.

For example, a 4-bit quantized Llama 3.1 8B model occupies approximately 4GB of static model memory [4], yet can generate more than 32GB of KV-cache state when serving eight concurrent users at a 32K-token context length. In this scenario, dynamic runtime state exceeds the model itself by roughly eight times.

How this figure is derived. Per-token KV-cache size = 2 (key and value) × number of layers × number of KV heads × head dimension × bytes per element. For Llama 3.1 8B under FP16 with grouped-query attention (32 layers, 8 KV heads, 128 head-dimension, 2 bytes per element), this is approximately 128KB per token. At a 32K-token context that is about 4GB per user, and across 8 concurrent users approximately 32GB — close to eight times the 4GB static footprint of the 4-bit quantized weights.

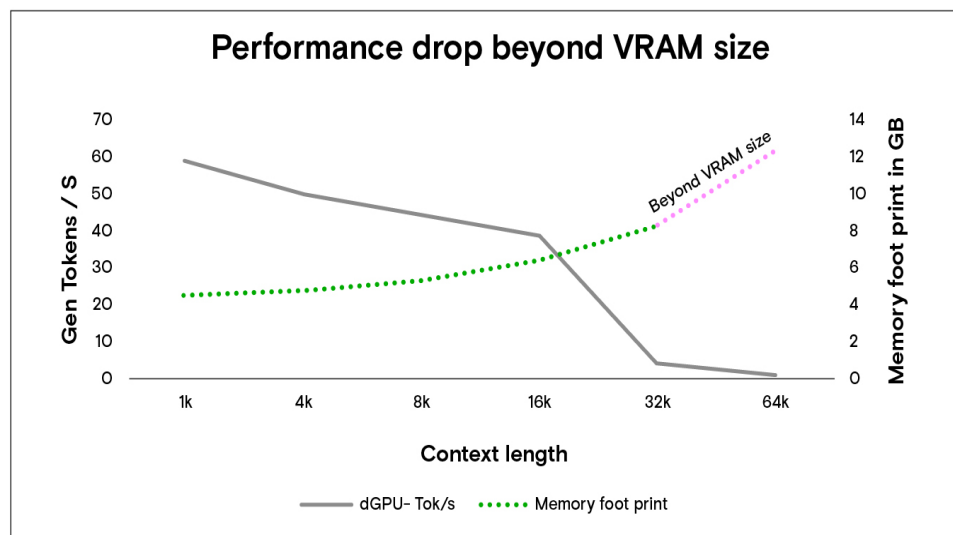
Figure 3: Scaling behavior of KV-cache memory across varying context lengths and concurrent inference sessions



This introduces a fundamental trade-off in discrete memory architectures. One option is to over-provision for worst-case dynamic demand, which results in poor utilization while preventing additional or parallel workloads from leveraging unused capacity. The alternative is to provision for typical usage, but impose constraints on context length, batch size, or concurrency to remain within VRAM limits. When these limits are exceeded, performance does not degrade gradually; instead, it drops sharply due to eviction cycles and multi-gigabyte data transfers over the CPU–GPU interconnect.

The example below illustrates this behavior on a discrete GPU system with 8GB VRAM and an additional 8 GB of shared system memory. The workload is built on a fixed 4GB model (static memory), while context length is progressively increased. As the dynamic memory component—primarily the KV cache—grows with context, the total working set eventually exceeds the available VRAM capacity, forcing data movement and triggering performance degradation.

Figure 4: Token generation rate (tokens/sec) versus memory footprint with increasing context length



As context length increases, the KV cache grows, driving higher total memory demand. Once this demand exceeds available VRAM capacity, token generation throughput drops sharply due to multi-gigabyte data transfers over PCIe and repeated eviction/reload cycles. This behavior reflects a hard capacity boundary in discrete GPU memory, where performance becomes dominated by data movement rather than computation.

3.1 Agentic AI makes memory a continuous requirement

The memory challenge becomes significantly more acute with agentic AI, where workloads no longer have a clear beginning or end. Unlike traditional inference, agentic systems operate in continuous loops—planning, executing, evaluating, and refining across extended chains of actions [7], [8]. Each step introduces new state, scratch space, tool outputs, and intermediate results that must remain available and coherent throughout the execution loop. In this context, long context windows are not an optional feature; they are fundamental to maintaining continuity and utility.

An additional characteristic of agentic behavior is the parallel activation of multiple models, each performing different roles such as reasoning, perception, tool use orchestration, or interaction. These models must coexist in memory simultaneously, along with both their static components (model weights) and their dynamic working state (context, KV

cache, and intermediate data). This further amplifies memory pressure, as the system must accommodate not just a single evolving working set, but multiple concurrent and interdependent working sets within the same memory envelope.

Keeping multiple models and their working state resident in one large pool also reduces repeated reloading of weights from storage: once the working set no longer fits, systems must move data across the slower storage hierarchy, well below the bandwidth of any in-memory path [3].

The limitation is no longer how fast systems compute, but whether they can hold and sustain the working set required to compute at all.

It is important to clarify that the limitation described here is not purely a question of total system memory capacity. In many systems, sufficient memory may exist across CPU DRAM and accelerator memory combined, but it remains partitioned and not universally accessible.

In discrete GPU architecture, the effective working set is constrained by the size of the GPU's local VRAM, even when additional system memory is available. While some platforms allow overflow into system DRAM via shared allocations, this introduces significant performance penalties due to low-bandwidth, high-latency transfers over the CPU-GPU PCIe interconnect. As a result, memory beyond the VRAM boundary exists but is not practically usable for sustained high-performance inference.

Integrated GPU (iGPU) and shared-memory designs allow both CPU and GPU to access system DRAM. However, this access remains constrained to a shared pool, with the GPU operating as a secondary requester and the CPU remaining the primary controller of memory access. When working sets grow large, contention and limited effective memory within the shared pool lead to similar limitations, preventing full utilization of available system DRAM for sustained AI workloads.

The challenge is therefore not a lack of physical memory, but the inability of contemporary architectures to utilize available memory as a unified resource. UMA addresses this problem by eliminating artificial memory boundaries and exposing system memory as a single coherent pool accessible by all compute engines.

4 The architectural requirement: Unified and coherent memory

The characteristics described above: dynamic memory growth, multi-model concurrency, and continuous execution, are not edge cases. They define how modern AI systems operate. Taken together, they expose a fundamental mismatch between AI workloads and traditional memory architectures.

Discrete and heterogeneous memory designs were built around partitioned memory systems, where CPU and accelerator memory exist as separate pools connected through explicit data movement. This model assumes that workloads can be segmented, staged, and transferred predictably. AI inference disrupts this assumption. Its working set is not only large, but continuously evolving, shared across compute elements, and often required to remain simultaneously resident.

As a result, the constraint is no longer just capacity, but how memory is organized and accessed. The problem is not simply that memory is insufficient, but that it is fragmented and non-coherent, forcing dynamic workloads to operate across artificial boundaries.

This leads to a key architectural requirement:

- AI workloads require a single, coherent memory space that allows both static and dynamic working sets to coexist and grow without crossing system boundaries.

- UMA addresses this requirement by eliminating the separation between memory domains and replacing it with a shared address space accessible to all compute engines.
- UMA is defined by a single unified address space, not by the physical location of the GPU. As a result, UMA can be implemented in multiple system configurations depending on platform design.

Some systems integrate the GPU within the same SoC as the CPU, connected through an on-die common fabric. Others deploy the GPU as a separate chip, connected to the SoC through a high-bandwidth, cache-coherent interconnect.

While these two approaches differ in physical layout and interconnect technology, they share the same architectural principle: CPU and GPU operate as peer requesters to a unified memory pool with no ownership boundaries or explicit data movement.

In this configuration, the GPU resides outside the SoC and is connected through a high-bandwidth, cache-coherent interconnect/link. Despite the physical separation, the CPU and GPU operate within a single uniform address space, accessing the same unified memory pool without explicit data movement. This model extends UMA principles across chip boundaries while preserving peer-level memory access.

Figure 5: UMA with GPU implemented as a separate chip

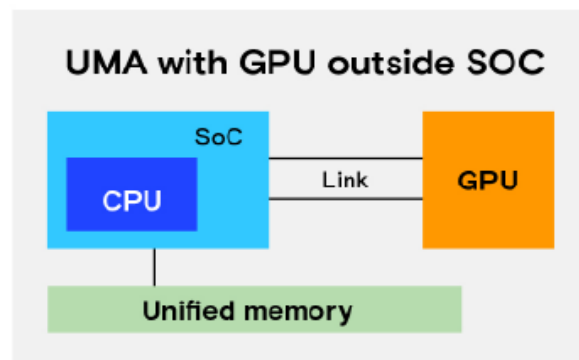
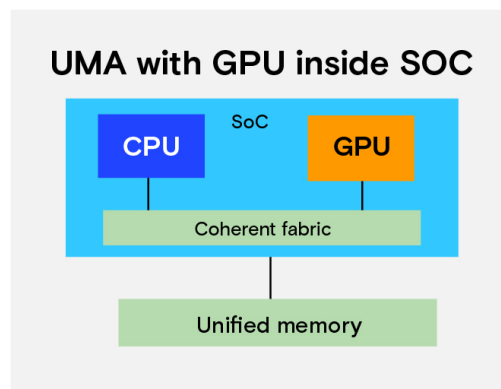


Figure 6: UMA with GPU integrated inside the SoC



In this design, the CPU and GPU reside within a single system-on-chip and are connected through a common on-die fabric. Both compute engines access a shared unified memory pool directly, with no data duplication or transfer overhead. This approach is commonly used in tightly integrated platforms like AMD Halo, Apple Mac mini.

4.1 Defining UMA as a system-level principle

UMA is not a new memory technology, but a system-level design in which CPU, GPU, and other compute elements operate as peer requesters to a single physical memory pool.

UMA is not a replacement for all memory hierarchies, but addresses the requirements of memory-intensive, stateful AI workloads whose working sets span multiple compute engines and evolve dynamically at runtime.

This reflects a deliberate trade-off. Unified pools — typically built on shared LPDDR — generally provide lower peak bandwidth than the dedicated GDDR attached to discrete accelerators. UMA's advantage therefore emerges where working-set capacity and data-movement overhead dominate, that is, where the alternative is eviction and transfer stalls; for workloads that fit comfortably within high-bandwidth local memory, that bandwidth can remain the better trade.

In this model:

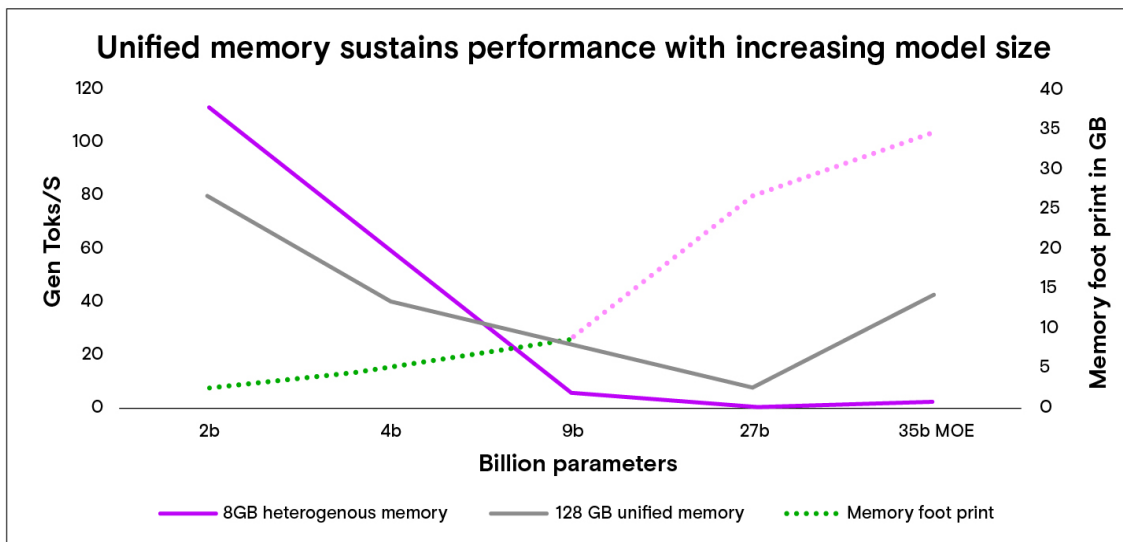
- Memory is allocated once, not duplicated across domains
- Data is accessed through shared addressing, rather than explicit transfers
- Static model weights and dynamic working state coexist within the same memory space
- Memory capacity is no longer bounded by a fixed accelerator-local pool, but by the aggregate system memory available to the workload

Unlike traditional memory architectures that assume bounded and localized working sets, UMA is designed to accommodate the dynamic and continuously evolving memory demands of modern AI inference. By exposing system memory as a single pool, UMA removes the locality constraints, fragmentation, and data-movement overheads that increasingly limit AI execution at scale.

This architectural shift removes the hard boundaries that previously separated model weights, context, and intermediate state, allowing AI workloads to scale in a manner consistent with their natural behavior.

The impact of this architectural difference can be observed across two primary stressors of AI workloads: model scale and context length.

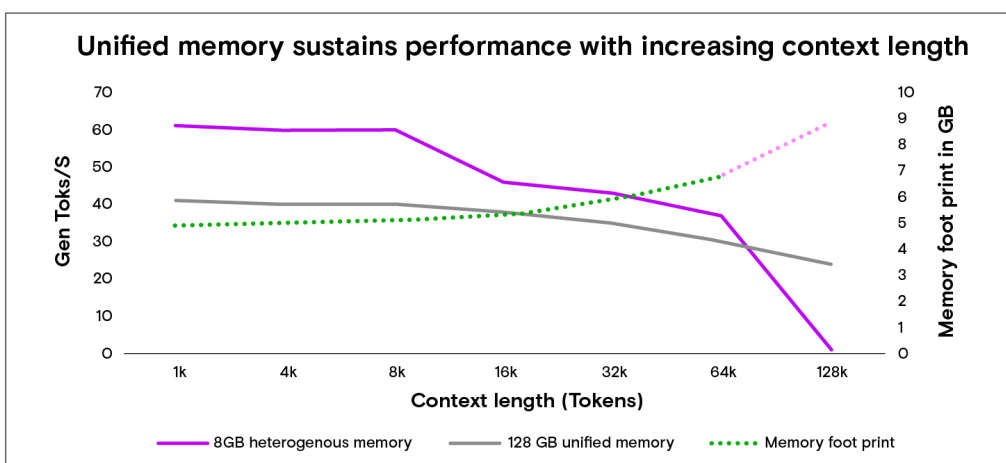
Figure 7: Token generation rate and memory footprint versus increasing model parameters



As model parameters increase, total memory footprint grows steadily. Once the working set exceeds heterogeneous VRAM capacity, throughput degrades sharply due to increased data movement over PCIe, while unified-memory architectures sustain higher token generation rates by accommodating the expanding working set within system DRAM.

Increasing context length raises memory demand through growth of the attention working set. Performance drops sharply when this demand exceeds VRAM capacity due to off-package memory transfers, whereas unified memory architectures maintain more stable throughput by allowing the working set to span system DRAM.

Figure 8: Token generation rate and memory footprint versus increasing context length



5 Related work

Unified memory is not a new idea, and the framing in this paper builds on more than a decade of prior art. The HSA Foundation's heterogeneous system architecture [9] and AMD's hUMA [10] established coherent, shared address spaces across CPU and GPU; Trough SW introduced Unified Memory [11] as a single, runtime-managed pointer space. On the client side, Apple's M-series [12] brought on-package unified memory to mainstream systems, and game consoles have used shared-memory designs for over a decade. Cross-chip designs — used in AI personal work stations — extend the same coherence across chips through a cache-coherent link.

What this paper emphasizes is not the mechanism of unified memory but its motivation for today's client and edge workloads. Earlier treatments framed unified memory mainly around programmer convenience or graphics-compute interoperability. Here the emphasis is the large, dynamic, and simultaneously resident working sets of memory-bound and increasingly agentic inference, and the observation that a single coherent address space — realized either inside an SoC or across a cache-coherent link — is the practical way to keep those working sets usable. This view is consistent with the memory-bound analysis of transformer inference by Pope et al. (2023) [1] and the memory-management techniques of PagedAttention / vLLM (2023) [2].

6 Memory architecture as a determinant of AI system behavior

AI inference is no longer a bounded, transactional workload. It is continuous, stateful, and increasingly defined by the size and persistence of its working set.

Traditional memory architectures struggle in this environment—not due to lack of total capacity, but because that capacity is partitioned, non-coherent, and constrained by locality. As working sets exceed accelerator-local limits, systems transition from compute-bound execution to transfer-bound behavior, introducing discontinuities that directly impact usability.

UMA addresses this at the system level. By exposing a single coherent address space across compute engines, UMA allows static model weights and dynamic runtime state to coexist and scale without crossing artificial boundaries. The result is not additional memory, but full utilization of available system memory under real workload conditions.

This distinction is critical. The constraint facing modern AI systems is not simply how much memory exists, but whether that memory can be accessed, shared, and sustained as the workload evolves.

As models grow, context expands, and agentic execution becomes the norm, memory architecture moves from a secondary design consideration to a primary determinant of system capability. Systems will be judged less by peak compute under ideal conditions, and more by their ability to sustain real workloads without crossing hard capacity boundaries.

The limiting factor for AI is no longer the ability to compute, it is the ability to sustain the working set required to compute.

7 Recommendations

A memory-bound view of inference suggests that system design should start from the working set rather than from peak compute. This is especially true for client and mobile platforms, where memory is shared and power-constrained: what matters most is how much of the model and its runtime state can stay resident, and how efficiently every compute engine can reach it. Designing around a single uniform address space — so that CPU, GPU, and NPU operate on the same data without copies — tends to preserve more usable capacity and more predictable performance than adding isolated memory to any one engine.

In practice, a few priorities follow:

- Size from the working set. Plan capacity and bandwidth around context length and concurrency, not model weights alone.
- Favor consistent, shared-memory topologies. Pair them with wide, power-efficient memory (for example, LPDDR-class memory and modular options such as LPCAMM2) so capacity and bandwidth scale together.
- Report sustained throughput honestly. Treat sustained tokens-per-second under realistic working sets as the headline metric, and state the memory configuration that produced it.
- Minimize movement where coherence is incomplete. Where unified memory is not available end-to-end, reduce cross-engine data movement and keep the most frequently accessed state resident.

8 Future outlook

The relevance of a memory-centric view is likely to grow as the memory roadmap advances. On client and mobile platforms, higher-bandwidth and power-efficient memory — LPDDR5X today and LPDDR6 as it is adopted [15] — together with modular formats such as LPCAMM2 make larger, shared working sets more practical on-device. On-package memory, already common in client SoCs, continues to shorten the distance between compute and data. These trends point in the same direction: more of a model and its runtime state can stay resident and shared, close to where it is used.

Several workload trends will shape how much memory inference actually needs. KV-cache compression [5], more aggressive quantization [4], [6], speculative decoding, mixture-of-experts routing, and emerging state-space or hybrid architectures may each reduce the dynamic-memory footprint over time. The roughly eight-to-tenfold ratio of dynamic to static memory seen today is best read as a current data point rather than a fixed rule, and it may narrow as these techniques mature. The underlying need for shared access across engines — particularly for concurrent, multi-model workloads — is largely independent of these changes and is likely to persist.

Comparable coherence ideas are being extended beyond a single package at larger scales — CXL memory pooling is one example — but that is a separate, server-tier trajectory. This paper stays with the client and edge, where memory locality and coherence largely determine which AI workloads a device can sustain. As models and context grow, the ability to hold and share a uniformed working set — not peak compute — is likely to remain the primary determinant of on-device AI capability.

Author bios

Kiran Matturi is a Director of Systems Design Engineering, leading host system architecture for next-generation AI workloads across mobile, client, and edge platforms. He possesses deep expertise in system architecture, with a focus on memory and storage subsystems, performance optimization, and end-to-end system design. His work centers on translating emerging AI use cases into data-driven system requirements spanning compute, memory, and software.

Sharath Chandra Ambula is a Staff Systems Engineer and Mobile Architect at Micron Technology, where he focuses on system architecture and performance analysis for mobile and client platforms, with an emphasis on memory-subsystem behavior. He brings over 10 years of industry experience spanning SoC system performance engineering and memory-centric system design across compute, interconnect, and memory subsystems. Sharath's work bridges system architecture and memory technology, with a focus on how standards, platform design choices, and real-world workloads influence performance, power, and scalability in modern client systems.

Tara Abarca is a marketing communications leader at Micron, focused on how advanced memory technologies power the next era of computing, from mobile devices to AI PCs and edge intelligence. As Principal for the Mobile and Client Business Unit, she drives strategic storytelling that highlights real-world impact and innovation across the ecosystem. Her work bridges technical depth and market relevance, helping audiences understand how data-driven experiences come to life.

References

Figure data and methodology (Figures 4, 7, and 8). Discrete-memory results were measured on an discrete architecture; unified-memory results on a relatively similar performing unified memory architecture. Inference was run with llama.cpp, and token-generation rate and memory footprint were captured in software. Discrete and unified-memory configurations were compared on the same model, quantization, and workload.

1. Pope, R., Douglas, S., Chowdhery, A., et al. (2023). Efficiently Scaling Transformer Inference. Proceedings of MLSys. <https://arxiv.org/abs/2211.05102>
2. Kwon, W., Li, Z., Zhuang, S., et al. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention (vLLM). Proceedings of SOSP. <https://arxiv.org/abs/2309.06180>
3. Sheng, Y., Zheng, L., Yuan, B., et al. (2023). FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. Proceedings of ICML. <https://arxiv.org/abs/2303.06865>
4. Lin, J., Tang, J., Tang, H., et al. (2024). AWQ: Activation-Aware Weight Quantization for LLM Compression and Acceleration. Proceedings of MLSys. <https://arxiv.org/abs/2306.00978>
5. Liu, Z., Yuan, J., Jin, H., et al. (2024). KIVI: A Tuning-Free Asymmetric 2-bit Quantization for the KV Cache. Proceedings of ICML. <https://arxiv.org/abs/2402.02750>
6. Xiao, G., Lin, J., Seznec, M., et al. (2023). SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. Proceedings of ICML. <https://arxiv.org/abs/2211.10438>
7. Yao, S., Zhao, J., Yu, D., et al. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. Proceedings of ICLR. <https://arxiv.org/abs/2210.03629>
8. Liu, X., et al. (2024). AgentBench: Evaluating LLMs as Agents. Proceedings of ICLR. <https://arxiv.org/abs/2308.03688>
9. HSA Foundation (2018). HSA Platform System Architecture Specification, Version 1.2. hsafoundation.com — HSA System Architecture Spec 1.2 (PDF)
10. AMD (2013). heterogeneous Uniform Memory Access (hUMA). AMD hUMA announcement
11. Apple (2024). Mac mini (M4 / M4 Pro) — Technical Specifications. <https://www.apple.com/mac-mini/specs/>
12. Qualcomm (2024). Snapdragon 8 Elite Mobile Platform — Product Brief. [qualcomm.com](https://www.qualcomm.com) — Snapdragon 8 Elite Product Brief (PDF)
13. MediaTek (2024). Dimensity 9400 Flagship 5G Chipset. [mediatek.com](https://www.mediatek.com) — Dimensity 9400
14. JEDEC (2025). LPDDR6 Standard (JESD209-6). [jedec.org](https://www.jedec.org) — LPDDR6 (JESD209-6) press release