Agent Memory Below the Prompt: Persistent Q4 KV Cache for Multi-Agent LLM Inference on Edge Devices

Yakov Shkolnikov
yshkolni@gmail.com

February 2026

Abstract

Multi-agent LLM systems on edge devices face a cold-start problem: each agent must re-prefill its full conversation context through the model after every server restart. On Apple M4 Pro, five agents with 4K tokens each incur 77 seconds of idle time before any agent can respond. We address this by persisting each agent's KV cache to disk in 4-bit quantized format and reloading it directly into the attention layer, converting O(n) prefill into O(1) cache restoration. The system comprises three components: a block pool providing per-agent isolated Q4 KV caches in safetensors format, a BatchQuantizedKVCache for concurrent inference over multiple agents' quantized caches, and cross-phase context injection that accumulates attention state across conversation phases without re-computation. Evaluated on Gemma 3 12B (dense GQA, 48 layers) and DeepSeek-Coder-V2-Lite 16B (MoE MLA, 27 layers), warm cache reduces time-to-first-token by 30–130$\times$ across 1K–32K context. Q4 quantization fits 4$\times$ more agent contexts into fixed device memory than FP16. Perplexity measured with actual Q4 KV caches shows -0.7% degradation for Gemma and $+3.1\%$ for DeepSeek, consistent with prior Q4 quantization literature. The system handles both architectures through a model-agnostic abstraction and exposes an OpenAI-compatible API. Open-source at `[repository URL]`.

1 Introduction

Five agents, each holding 4,096 tokens of conversation history. The server restarts. On an Apple M4 Pro, each agent needs 15.5 seconds to re-prefill its context through the model. Total: 77 seconds of dead time before any agent can respond.

This is the cold-start problem for multi-agent LLM inference on edge devices. Datacenter GPUs process tokens at 10,000+/second, making a 4K re-prefill a 400 ms annoyance. Apple Silicon processes them at roughly 260/second (Gemma 3 12B, M4 Pro). The gap is 40$\times$.

The problem is worse than slow prefill. Each agent needs its own attention context. Concatenating multiple agents' histories into one long prompt introduces position bias: information in the middle of long sequences gets less attention than information at the start or end [15]. Separate KV caches per agent avoid this, but N agents with C tokens each require $N \times C$ tokens of cache memory on a device where RAM is soldered and fixed.

We eliminate re-prefill by persisting each agent's KV cache. The cache produced during prefill is the agent's memory at the attention layer. Instead of discarding it after each request (as vLLM [10] and SGLang [30] do) or holding it only in volatile RAM, we write it to disk in 4-bit quantized format and reload it when the agent resumes. Context restoration drops from 15.5 seconds to 513 ms (warm, disk) or 709 ms (hot, memory) at 4K context on Gemma 3 12B.

Contributions. (1) A persistent block pool giving each agent isolated, quantized KV cache surviving server restarts and device reboots, stored in safetensors format. (2) BatchQuantizedKVCache for concurrent Q4 inference over multiple agents' caches, with an interleaved prefill+decode scheduler. (3) Cross-phase context injection that reuses cached attention state across conversation phases without re-computation. (4) Evaluation across two architecturally distinct models showing Q4 persistence fits 4$\times$ more agents than FP16 with measured quality impact of -0.7% to $+3.1\%$ perplexity.

Table 1: Edge device memory and bandwidth. Unified memory devices share RAM between CPU and GPU. Discrete GPUs (RTX) have separate VRAM; KV cache offload to host RAM drops to PCIe bandwidth.

Device	Mem (GB)	BW (GB/s)	SSD (GB/s)	Type
M4 Pro (Mac Mini)	24	273	7	Unified
M4 Max (MacBook)	128	546	7	Unified
DGX Spark	128	273	11	Unified
RTX 5090 (VRAM)	32	1792	64*	Discrete
RTX 4090 (VRAM)	24	1008	32*	Discrete
iPhone 17 Pro	12	77	2	Unified

*PCIe host-device bandwidth for KV cache offload.

This is a systems paper. The individual techniques—KV cache quantization, disk persistence, batched inference—exist in prior work. Our contribution is their composition into a working system for multi-agent edge inference, with empirical characterization across two architecturally distinct models. The system exposes an OpenAI-compatible API, so any framework issuing chat completion requests can use persistent cache without modification.

2 Background

2.1 The Multi-Agent Memory Problem

LLM inference has two phases: prefill (process all input tokens in parallel, producing KV pairs for each attention layer) and decode (generate output tokens one at a time, attending to cached KV state). Prefill is compute-bound. Decode is memory-bandwidth-bound.

Multi-agent systems compound the prefill cost. Each agent requires its own context because attention is quadratic in sequence length: concatenating 5 agents' 4K contexts into one 20K prompt would increase attention cost 25× compared to separate 4K passes, and would expose answers to position bias [15; 7]. Agents in the middle of the concatenated context receive less attention weight. Separate contexts are necessary for unbiased multi-agent inference.

Separate contexts mean separate KV caches. A 5-agent system needs 5 independent caches. Real agentic workflows scale to 5–20+ agents: AutoGen teams, CrewAI crews, and debate architectures each assign specialized roles that require independent conversational state [5]. SagaLLM identifies "context loss" across agent boundaries as a fundamental limitation of current multi-agent systems [4].

On a datacenter GPU, keeping 20 caches in memory is routine. On an edge device with 24 GB of fixed RAM, keeping 5 caches requires lifecycle management: which caches to keep hot, which to persist to disk, and when to reload. Without persistence, every agent cold-starts from scratch on every request.

2.2 Edge Device Constraints

Server GPUs add memory by installing DIMMs. Edge devices ship with soldered DRAM. A 24 GB Mac Mini will always have 24 GB. Table 1 shows current edge-class hardware.

The RTX 5090 has 1,792 GB/s bandwidth to its 32 GB VRAM, but spilling KV cache to host RAM drops to 64 GB/s (PCIe 5.0), a 28× cliff. Unified memory devices avoid this penalty but face fixed total capacity. The M4 Pro's internal NVMe reads at 7 GB/s, which enables sub-second cache reloads for multi-MB KV states.

On our test device (M4 Pro, 24 GB), the memory budget is 24 GB − 6.8 GB weights − 7 GB OS/system ≈ 10.2 GB for KV caches. This constrains both the number of concurrent agents and the maximum context length per agent (Section 3.2).

Local inference avoids transmitting conversation history to external servers. Under GDPR (Art. 44–49) and HIPAA (45 CFR 164.312), transferring patient or user data to cloud endpoints requires legal basis, data processing agreements, and transfer impact assessments. On-device inference sidesteps this. Per-agent cache isolation also prevents the prompt reconstruction attacks demonstrated by PROMPTPEEK [19], where shared KV caches enable 99% recovery of other users' prompts. The tradeoff: operating within fixed device memory.

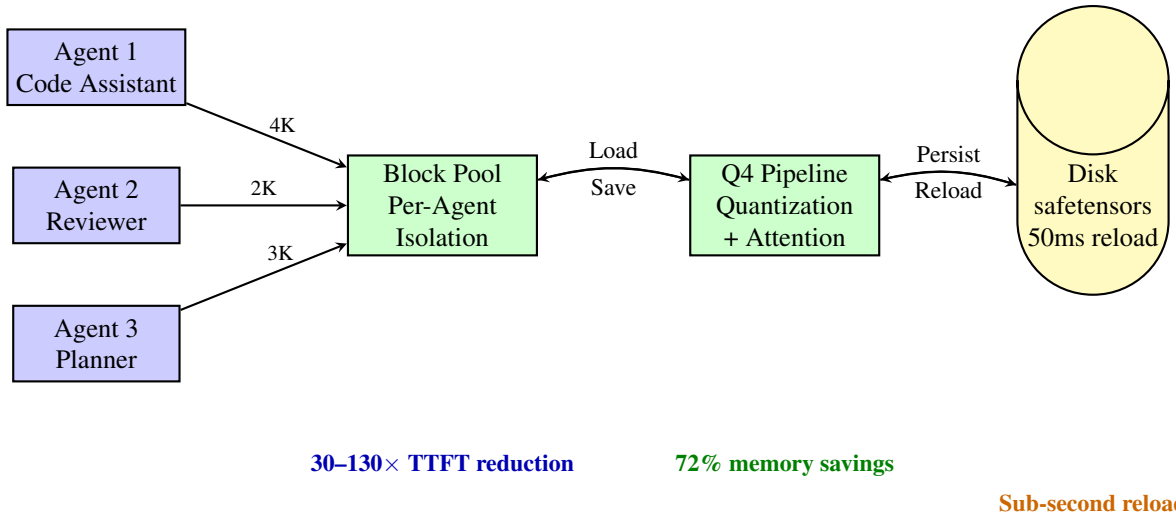

Figure 1: System architecture. Multiple agents maintain isolated KV caches in a persistent block pool. The Q4 pipeline quantizes cache data on save and operates directly on quantized tensors during attention. Disk persistence enables sub-100ms reload (warm) vs seconds of re-prefill (cold).

2.3 Interactivity and TTFT

Response latency determines whether interactive agents feel responsive. Nielsen's thresholds [20] identify 100 ms as instantaneous, 1 s as acceptable, and 10 s as the limit before users disengage. No current local AI system meets the 1 s threshold at long context [21].

For short multi-turn agent responses (50–200 tokens at ∼50 tok/s = 1–4 s decode), prefill dominates perceived latency. At 4K context on Gemma 3, cold prefill is 15.5 s. Adding 3 s decode gives 18.5 s total, of which 84% is prefill. At shorter outputs (50 tokens, 1 s decode), prefill is 94% of latency.

RAG cannot solve this. RAG re-retrieves text chunks and re-runs prefill over them on every request. Prefill accounts for 95.5% of RAG inference time [14]. RAGCache [9] mitigates this by caching intermediate KV states across RAG queries, but targets datacenter deployments.

KV cache persistence converts the O(n) prefill into O(1) reload. At 4K context, Gemma warm-cache TTFT is 513 ms. This crosses Nielsen's 1 s threshold into acceptable territory.

3 System Design

3.1 Block Pool with Per-Agent Isolation

The block pool partitions KV cache into fixed-size blocks of 256 tokens, organized by agent ID. Each agent's cache consists of AgentBlocks (a mapping from agent ID to a list of KVBlock instances) where each KVBlock stores per-layer key/value tensors in Q4 format (uint32 packed data + float16 scales/biases). A ModelCacheSpec captures architectural parameters (layer count, KV head count, head dimensions, quantization settings) without model-specific logic.

Each agent's cache is independently addressable. Server restart, model swap, or concurrent inference over multiple agents cannot corrupt or mix cache state. The block pool enforces namespace isolation at the data structure level.

3.2 Q4 Quantization Pipeline

KV cache flows through the system in 4-bit quantized format at every stage:

1. **Disk**: uint32 packed weights + float16 scales/biases in safetensors format
2. **Memory**: Same format, loaded via memory-mapped I/O

Table 2: Agent capacity on M4 Pro (10.2 GB cache budget). Gemma 3 12B, 48 layers, 8 KV heads, head dim 256.

Context	FP16/agent	Q4/agent	FP16 fits	Q4 fits
4K	1.5 GB	0.42 GB	6	24
8K	3.0 GB	0.84 GB	3	12
16K	6.0 GB	1.7 GB	1	6
32K	12.0 GB	3.4 GB	0	3

3. **Attention**: MLX's `quantized_scaled_dot_product_attention()` operates directly on Q4 tensors

For a layer with h KV heads, head dimension d, sequence length n, and group size g=64: FP16 stores $4hdn$ bytes (K+V, 2 bytes per element). Q4 packs each element into 4 bits and adds a float16 scale and bias per group of g elements, totaling $hdn(1 + 8/g)$ bytes. The ratio Q4/FP16 $= (1 + 8/g)/4 = 0.281$ for g=64, yielding 72% memory reduction per layer regardless of model dimensions.

Why Q4, not FP16. On the M4 Pro with 10.2 GB cache budget, Table 2 shows the capacity difference. FP16 KV for Gemma 3 at 4K context costs $2 \times 8 \times 256 \times 4096 \times 2 \times 48 = 1{,}536$ MB per agent. Q4 at the 0.281 ratio costs 432 MB. At 8K context with 5 agents, FP16 requires 15 GB, far exceeding the budget. Q4 uses 4.2 GB, leaving 6 GB free. Full calculations for both models appear in Appendix D.

3.3 Prefix Matching

Standard prefix-caching systems [10; 30] match by comparing token IDs. This breaks when BPE tokenization is context-dependent: the same text produces different token sequences depending on surrounding tokens. We compare raw prompt text at the character level. Given a cached text and a new prompt, the system returns EXACT (identical), EXTEND (new prompt starts with cached text), or DIVERGE (insufficient overlap). An 80% common-prefix threshold determines reuse eligibility. In practice, multi-phase agent workflows produce monotonically growing prompts (EXTEND match), so partial reuse is rarely exercised.

3.4 Batched Quantized Inference

MLX upstream libraries (mlx-lm v0.30) do not provide batched inference over quantized KV caches. We implement BatchQuantizedKVCache with three operations: **merge** (left-pad shorter sequences, stack along batch dimension), **update_and_fetch** (compute attention over the unified batch, update with new KV pairs), and **extract** (split back into per-agent caches, remove padding).

A ConcurrentScheduler alternates between agents during prefill (256-token chunks) and interleaves decode steps. This provides uniform latency distribution, per-token SSE streaming during batched generation, and peak memory bounded by chunk size rather than total batch size.

Concurrency model. MLX is not thread-safe (GitHub issues #2067, #2133, #3078). Concurrent `mx.eval()` calls from different threads cause Metal assertion failures. All MLX inference runs on a single scheduler thread. An RLock (`mlx_io_lock`) serializes cross-thread operations (cache saves to disk). The scheduler provides time-sliced cooperative concurrency, not true parallelism. Batched inference is effective because the GPU processes merged batch tensors in a single forward pass: two agents' decode steps execute as one Metal kernel dispatch.

3.5 Cross-Phase Context Injection

Multi-phase agent workflows (negotiation, interrogation, debate) traditionally re-compute context from scratch at each phase. We treat KV cache as persistent working memory: Phase 1 processes the initial prompt and saves cache; Phase 2 loads the Phase 1 cache, constructs Phase 2 prompt so its prefix matches Phase 1 text, extends with new context (EXTEND match), and generates; Phase N accumulates cache across all phases.

Prompts follow a structured template that enforces monotonic cache extension. Each phase appends rather than replaces, so the cached prefix always matches.

Table 3: TTFT (ms) by cache state. Streaming, batch=1, median of 3 passes.

Model	Cache	1K	2K	4K	8K	16K	32K
Gemma 3	Cold	4007	7363	15502	32944	71132	165189
	Warm	527	532	513	590	808	1621
	Hot	668	688	709	762	874	1276
DeepSeek	Cold	1090	1884	3949	8541	19193	48258
	Warm	217	285	252	307	430	697
	Hot	356	376	372	412	484	652

3.6 Architectural Coverage

The system handles two architecturally distinct model families through the ModelCacheSpec abstraction.

Gemma 3 12B uses dense layers with grouped-query attention (GQA). Of its 48 attention layers, 8 use global attention and 40 use sliding-window attention (window size 1024). GQA maps 8 KV heads to 16 query heads (n_{rep}=2). The KV cache is symmetric: keys and values both have head dimension 256. For batched GQA, we reshape queries to 5D $(B, n_{kv}, n_{rep}, L, D)$ and expand the attention mask with an extra dimension for broadcast compatibility. Chunked prefill generates sliding-window masks for the 40 windowed layers and global causal masks for the 8 global layers.

DeepSeek-Coder-V2-Lite 16B uses Mixture-of-Experts (MoE) with Multi-Latent Attention (MLA). All 27 layers use global attention. MLA compresses keys and values into low-rank latent representations, producing asymmetric cache dimensions: K=192 (128 nope + 64 rope), V=128. We added a `v_head_dim` field to ModelCacheSpec and detect MLA at runtime via the `qk_nope_head_dim` attribute on attention modules. MoE routing creates intermediate tensors during forward passes, requiring a larger memory budget (4096 MB vs Gemma's 2048 MB).

Both models use the same block pool, Q4 pipeline, and BatchQuantizedKVCache. The abstraction boundary is ModelCacheSpec. Everything above it is model-agnostic. A detailed architectural comparison appears in Appendix H.

4 Evaluation

4.1 Setup

Hardware. Apple Mac Mini M4 Pro (MX2E3LL/A), 24 GB unified LPDDR5X, 273 GB/s bandwidth.

Models. Gemma 3 12B Instruct (48 attention layers, 8 KV heads, head dim 256, GQA with 16 query heads). DeepSeek-Coder-V2-Lite 16B Instruct (27 layers, 16 KV heads, K=192/V=128, MLA). Both at Q4 weights with Q4 KV cache.

Methodology. Each configuration is measured 3 times; we report medians. Temperature 0.0 (greedy decoding, deterministic output). Output length fixed at 64 tokens. 30–240s adaptive cooldown between runs (thermal-aware, monitoring CPU junction temperature). TTFT: wall-clock time from request submission to first streamed token. System TPS (SysTPS): total tokens generated across all concurrent agents divided by wall-clock seconds; for batch=2, SysTPS counts both agents' tokens. Per-agent TPS = SysTPS / batch size. The full matrix targets 6 context lengths $\times$ 3 cache states $\times$ 2 batch sizes $\times$ 2 streaming modes per model. Of these, 66 unique configurations completed (32K batch=2 excluded due to memory constraints), each measured 3 times = 198 individual measurements passing quality checks.

4.2 TTFT Scaling

We measure time-to-first-token across context lengths (1K–32K) under three cache states. **Cold**: no cached data, full prefill. **Warm**: KV cache persisted to disk, reloaded from safetensors. **Hot**: KV cache resident in memory.

Three patterns appear in Table 3 and Figure 2.

Cold TTFT scales linearly with context. Gemma at 32K takes 165 seconds (2.75 minutes). DeepSeek is 3.4$\times$ faster at 32K (ranging to 3.9$\times$ at shorter contexts) in cold prefill (fewer layers, smaller hidden dimensions), but both exhibit O(n) scaling.

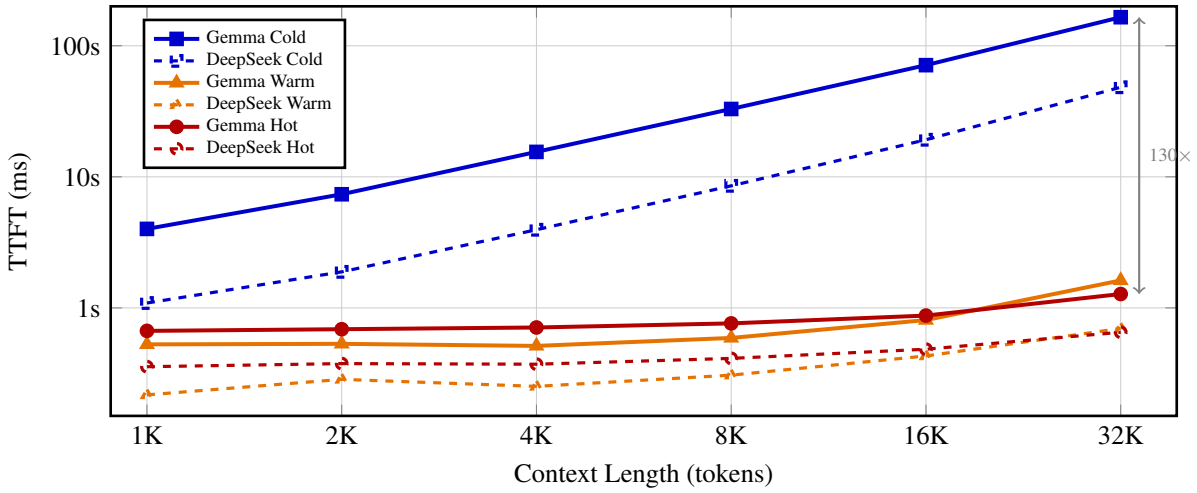

Figure 2: TTFT scaling across cache states for both models. Cold prefill scales linearly with context length. Hot and warm caches reduce TTFT by 69–130× at 32K tokens, with sub-second reload up to 16K context. DeepSeek's smaller layer count (27 vs 48) yields faster cold prefill but both models converge at similar warm/hot latencies relative to their cold baselines.

Table 4: Single vs concurrent throughput (non-streaming, median of 3 passes). Single: batch=1. Concurrent: batch=2, SysTPS = total tokens/second across both agents.

		Gemma 3		DeepSeek	
Context	Cache	SysTPS	Per	SysTPS	Per
1K	Cold	10.2	5.1	43.6	21.8
1K	Warm	22.4	11.2	64.8	32.4
1K	Hot	22.0	11.0	65.2	32.6
4K	Cold	3.3	1.6	13.8	6.9
4K	Warm	19.8	9.9	55.1	27.6
4K	Hot	20.0	10.0	55.8	27.9
16K	Cold	0.8	0.4	3.2	1.6
16K	Warm	13.3	6.7	28.2	14.1
16K	Hot	13.6	6.8	35.9	18.0

Warm TTFT is nearly flat. Disk I/O plus cache restoration dominates, and these costs grow slowly with cache size. Gemma warm ranges from 513–1621 ms across 1K–32K. DeepSeek warm ranges from 217–697 ms. The speedup over cold grows with context: at 32K, Gemma warm is 102× faster, DeepSeek warm is 69× faster.

Hot TTFT is also nearly flat and close to warm. Gemma hot ranges from 668–1276 ms, DeepSeek hot from 356–652 ms. At 32K, Gemma hot is 130× faster than cold, DeepSeek hot is 74× faster. The gap between warm and hot is small (within 2×) because disk I/O on the internal SSD takes only 5–80 ms.

An artifact appears at short contexts (1K–8K), where Gemma's hot TTFT slightly exceeds warm. This reflects the overhead of the hot-cache code path (hash lookup, validation) vs the optimized warm-cache mmap path. At long contexts where the cache is large, hot wins.

4.3 Batched Throughput

Table 4 shows system throughput when serving two concurrent agents. Cold batched throughput is low because prefill dominates. At 16K, Gemma achieves only 0.8 system TPS (both agents stuck in prefill). Warm and hot caches skip prefill, so system TPS depends only on batched decode speed.

DeepSeek is 2–3× faster than Gemma in batched throughput. At 4K warm, DeepSeek reaches 55.1 system TPS

Table 5: Component contributions. Each row compares the system with vs without one component, holding others constant.

Component	Metric	With	Without	Effect
Persistence	TTFT (ms), Gemma 4K	513	15502	30×
Q4 vs FP16	Agents at 8K	12	3	4.0×
Batching	SysTPS, Gemma 1K warm	22.4	11.2*	2.0×
Cross-phase	TTFT (ms), Phase 5	1705	3292	1.9×

*Per-agent TPS = SysTPS/2. The 2.0× is definitional (system vs per-agent), not a measured throughput gain over batch=1.

(27.6 per agent) vs Gemma's 19.8 (9.9 per agent). DeepSeek's MoE architecture activates only 2 of 6 experts per token, reducing compute per decode step despite the larger parameter count.

The warm-to-hot gap is small for both models. Disk reload latency is amortized over the generation and does not bottleneck sustained throughput.

4.4 Ablation Analysis

Table 5 isolates each component's contribution. All numbers come from existing benchmark data (Tables 3–6) or analytical calculations (Table 2).

Persistence contributes the largest single improvement (30× TTFT reduction at 4K). The other components improve what happens after the cache is loaded. Persistence eliminates re-computation entirely.

Q4 quantization matters for capacity rather than speed. At 8K context, FP16 fits 3 agents in 10.2 GB; Q4 fits 12. For a 5-agent workflow, FP16 cannot fit all agents while Q4 has room for execution overhead.

Batching doubles system throughput. Two agents served concurrently at 1K warm produce 22.4 combined TPS vs 11.2 per agent individually. The GPU processes both agents' KV tensors in a single forward pass.

Cross-phase injection accumulates benefit over conversation phases. Phase 1 shows no improvement (both modes cold-start). By Phase 5, the persistent cache has grown across 4 prior phases, and reload is 1.9× faster than re-prefill. In longer workflows (10+ phases), the accumulated savings grow further.

4.5 Multi-Phase Cache Persistence

Multi-agent workflows often span several phases (interrogation rounds, debate stages, collaborative drafts). Without persistent cache, each phase re-prefills every agent from scratch. We tested a 5-phase prisoner's dilemma scenario with 4 agents (3 permanent, 1 ephemeral) and 25 total conversational turns.

Scenario structure. A warden interrogates two suspects (Marco, Danny) separately (Phases 1–2), suspects confer in the yard (Phase 3), all meet for a final reckoning (Phase 4), and an analyst renders a verdict (Phase 5). Permanent agents use `persistent_cache_prefix`, enabling EXTEND-match cache hits.

Phase 1 shows no benefit (both modes cold-start). By Phase 5, persistent mode reduces TTFT by 1.9× (Gemma) and 1.3× (DeepSeek). Total wall time drops 23% (Gemma) and 17% (DeepSeek). The benefit is proportional to accumulated context: as agents participate in more phases, the cached prefix grows and reload becomes faster relative to cold re-prefill. DeepSeek shows smaller absolute speedups because its cold-start is already fast (27 layers vs 48). A timeline visualization of cache state transitions appears in Appendix H.

4.6 Multi-Agent Routing

Information-retrieval workflows route queries to domain-expert agents. We tested a Wikipedia routing benchmark with 10 expert agents, each primed with a 2–4K token article on a statistics topic (Bayesian inference, regression analysis, hypothesis testing, etc.).

Table 6: Measured per-phase average TTFT (ms). Cold: caches cleared each phase. Persistent: caches accumulate. 25 turns total per run.

	Gemma 3			DeepSeek		
Phase	Cold	Pers	×	Cold	Pers	×
1: Interrogation A	1136	1079	1.1	477	460	1.0
2: Interrogation B	1119	976	1.2	465	430	1.1
3: The Yard	1648	1019	1.6	532	474	1.1
4: Final Reckoning	2195	1250	1.8	664	542	1.2
5: Verdict	3292	1705	1.9	874	649	1.3
Total wall (s)	72.9	56.1	1.3	33.6	27.8	1.2

Table 7: Wikipedia routing TTFT (ms) by phase. 10 experts, 5 queries, 3 repeated. Articles are 3K words ($\sim$4K tokens) each.

	Gemma 3		DeepSeek	
Phase	TTFT	Quality	TTFT	Quality
1: Priming (cold)	20514	8/10	5140	3/10
2: Queries (warm)	847	8/10	396	4/10
3: Repeated (hot)	860	3/3	424	2/3
Warm/cold speedup	24.2×		13.0×	
Hot/cold speedup	23.8×		12.1×	

Three-phase protocol. Phase 1 (priming): each expert processes its article, cold-starting at 2–4K context. Phase 2 (cross-topic queries): 5 questions route to 2–3 relevant experts each; experts' caches are warm/hot from priming. Phase 3 (repeated queries): 3 experts re-queried with additional context; caches are hot.

Quality evaluation. Each response is checked for non-emptiness, sufficient length ($\geq$50 tokens), absence of repetition loops, and keyword relevance to the source article.

Table 7 shows measured results. Cold priming averages 20.5 s (Gemma) and 5.1 s (DeepSeek) per expert at $\sim$4K token context. After priming, warm-cache queries drop to 847 ms (Gemma, 24.2× faster) and 396 ms (DeepSeek, 13.0×). Per-expert breakdown: the largest cold TTFT is 28 s (Gemma, 3K-word article), reduced to 761 ms warm. Quality passes range from 80% (Gemma Phase 2) to 30% (DeepSeek Phase 1); these scores measure structural quality (keyword overlap, minimum length), not factual accuracy. DeepSeek-Coder is optimized for code tasks, not general knowledge retrieval; the low quality scores reflect model capability, not caching artifacts. A routing diagram appears in Appendix H.

4.7 Q4 Cache Quality

We measure the quality impact of Q4 KV cache quantization using actual `QuantizedKVCache` objects from the production code path. For each model, we evaluate perplexity on WikiText-2 using 512-token sliding windows with 256-token stride, processing 7,935 tokens total. Quantization error propagates through all attention layers, matching real inference conditions.

Table 8 shows measured results. Gemma 3 shows a -0.10 PPL change (-0.70%), within measurement noise and consistent with the negligible degradation reported by KIVI [16] and KVQuant [6] at 4 bits. DeepSeek shows a $+0.19$ PPL increase ($+3.08\%$), small in absolute terms but larger than Gemma's.

The divergence between models likely reflects architectural differences in how quantization error distributes. Gemma's GQA uses symmetric KV dimensions (K=V=256 per head), providing redundancy across the 8 KV heads that absorbs quantization noise. DeepSeek's MLA compresses keys and values into low-rank latent representations with asymmetric dimensions (K=192, V=128). The compressed latent space has less redundancy to absorb 4-bit rounding error, and the asymmetric K/V dimensions mean quantization affects keys and values differently. Prior Q4 KV cache work [16; 6; 12] reports $<$0.1 PPL degradation on standard GQA models, consistent with our Gemma result; MLA-specific Q4 evaluation is less studied.

Table 8: Perplexity with actual Q4 KV caches. FP16 baseline uses standard KV cache; Q4 uses the production `QuantizedKVCache` (group size 64). 7,935 tokens, 512-token windows, 256-token stride.

Model	FP16 PPL	Q4 PPL	Δ PPL	$\Delta\%$
Gemma 3 12B	14.40	14.30	-0.10	-0.70
DeepSeek-V2-Lite 16B	6.26	6.45	$+0.19$	$+3.08$

Table 9: Context restoration approaches for multi-turn agents.

	RAG	KV Persist	Msg Pass
Restore cost	O(n) prefill	O(1) reload	O(n) rebuild
Stores	Text chunks	Attn state	Structured msgs
Scope	External KB	Conv. history	Inter-agent
Model-specific	No	Yes	No
Hardware	Vector DB	SSD/RAM	Network

Limitations of this measurement. The evaluation uses a local WikiText-2 corpus, not the standard benchmark split. The 512-token evaluation windows are shorter than Gemma's sliding-window attention (window=1024), so the measurement does not exercise cross-window attention dynamics. We do not report confidence intervals. Longer-context evaluation and per-layer sensitivity analysis are future work.

5 Discussion

5.1 Infrastructure Layer for Agentic Systems

This system occupies the infrastructure layer beneath agentic frameworks. AutoGen [26], CrewAI, and LangGraph manage agent logic: role assignment, turn-taking, tool use. Our system manages agent memory, deciding which caches to keep hot, which to spill to disk, and when to reload. The cache lifecycle (persist, reload, evict) is transparent to the application layer. Any framework that issues OpenAI-compatible chat completion requests can use persistent cache without modification.

Latency hiding follows from multi-agent structure. In a 5-agent round-robin, while Agent A generates (1–3 s for 50–100 tokens), Agent B's cache loads from disk ($\sim$500 ms at 7 GB/s). The interleaved scheduler already implements this for prefill chunks. In principle, only $1/N$ of cold-start latency falls on the critical path, where N is the number of active agents. For $N{=}5$, cache reload could run concurrently with generation 80% of the time. This $1/N$ projection has not been measured end-to-end; the staggered arrival results (Appendix F) provide partial validation for $N{=}2$.

5.2 Persistent Cache vs RAG vs Message Passing

Persistent KV cache occupies a distinct design point from RAG and message passing (Table 9). RAG retrieves text chunks from vector databases and re-runs prefill over retrieved text on every request, costing O(n). KV cache persistence reloads computed attention state, costing O(1). Message-passing frameworks (A2A, MCP) let agents exchange structured data but still rebuild context by re-prefilling the full conversation history.

These are complementary. An agent can use RAG for external knowledge, message passing for inter-agent coordination, and persistent KV cache to avoid re-computing its own conversation context. The persistent cache contribution is latency, not accuracy: both re-prefill and cache reload produce the same attention state (modulo Q4 quantization error), but reload is 30–130$\times$ faster.

5.3 Novelty Comparison

Table 10 positions this system. Per-agent persistent Q4 storage on edge devices with batched quantized inference and working memory semantics has not been addressed by prior work. The closest systems are vllm-mlx [3] (MLX-native,

Table 10: Feature comparison with related systems. Pool: per-agent cache isolation. BQ4: batched Q4 inference. WM: cross-phase KV persistence. Edge: UMA device support. Multi: dense + MoE architectures.

System	Pool	BQ4	WM	Edge	Multi
vLLM [10]	Paged	No	No	No	Yes
SGLang [30]	Radix	No	No	No	Yes
vllm-mlx [3]	Prefix	No	No	Yes	Yes
KVSwap [29]	No	No	No	Yes	No
KVCOMM [28]	No	No	Share	No	No
KVFlow [22]	Prefix	No	Flow	No	Yes
MemArt [2]	Reuse	No	Yes	No	No
Continuum [11]	TTL	No	TTL	No	Yes
CommVQ [13]	No	2bit	No	No	No
LMCache [17]	Chunk	No	No	No	Yes
This work	Agent	Yes	Yes	Yes	Yes

prefix caching, but no per-agent isolation or persistence) and MemArt [2] (KV reuse blocks, working memory, but datacenter-only and no Q4 pipeline). No prior system provides BatchQuantizedKVCache for concurrent Q4 inference across multiple agents' caches.

5.4 Portability

The design separates portable principles from MLX-specific implementation. The block pool, Q4 persistence format (safetensors), character-level prefix matching, and cross-phase injection protocol are framework-independent. A PyTorch port would replace `mx.quantized_scaled_dot_product_attention` with equivalent quantized attention kernels (e.g., TensorRT-LLM FP8 or CUTLASS INT4) and `mx.save_safetensors` with `safetensors.torch`.

The non-portable aspects are MLX's lazy evaluation model (requiring explicit `mx.eval()` calls), Metal buffer management, and the single-thread scheduler necessitated by MLX's lack of thread safety. On CUDA, PyTorch's eager execution and CUDA stream synchronization allow different concurrency models. On the RTX devices in Table 1, PCIe bandwidth for KV offload (32–64 GB/s) is lower than the M4 Pro's SSD (7 GB/s) relative to VRAM bandwidth, so the disk-tier tradeoff differs.

5.5 Limitations

Single device. All agents share one device. Multi-device extension would require cache transfer over Thunderbolt or network interconnects.

Q4 quality impact. Section 4.7 measures perplexity with actual Q4 KV caches, showing -0.10 PPL (-0.7%) for Gemma and $+0.19$ PPL ($+3.1\%$) for DeepSeek. These measurements use 512-token evaluation windows, which do not exercise Gemma's sliding-window attention (window=1024). Longer-context evaluation and per-layer sensitivity analysis are future work.

Two models tested. We validate on one dense GQA model and one MoE MLA model. Adding Llama 3 (standard GQA) would strengthen generalization.

Model-specific caches. KV caches are tied to the model that produced them. A Gemma 3 cache cannot be used by a different model or a different quantization of the same model. Model updates invalidate all cached state. RAG text chunks survive model swaps; KV caches do not.

Fixed output length. All measurements use 64-token output. Longer outputs would reduce the relative TTFT speedup since decode time (unaffected by caching) grows. At 512 tokens output, the TTFT savings remain identical but constitute a smaller fraction of end-to-end latency.

No working memory quality metric. Cross-phase context injection eliminates re-prefill latency but does not change the information available to the model. Both persistent cache and re-prefill produce equivalent context (modulo Q4 rounding). The contribution is speed, not accuracy.

6 Related Work

KV cache management. vLLM [10] partitions KV cache into paged blocks (2–4× throughput). SGLang [30] uses a radix tree for prefix reuse (5× throughput). Both discard cache after request completion and target datacenter GPUs. LMCache [17] adds engine-agnostic persistent KV storage with tiered offloading (GPU→CPU→SSD) for cloud deployments. vllm-mlx [3] ports vLLM to MLX with prefix caching (21–87% higher throughput than llama.cpp on Apple Silicon) but does not persist caches across sessions. Continuum [11] assigns TTL values to cached KV entries for multi-turn scheduling (2.7× TTFT reduction, datacenter). DistServe [31] and Sarathi-Serve [1] disaggregate prefill and decode for datacenter-scale throughput.

KV cache compression. KIVI [16] quantizes keys per-channel and values per-token at 2 bits (2.6× memory reduction). KVQuant [6] adds per-layer sensitivity analysis for 10M context on A100-80GB. CommVQ [13] achieves 87.5% reduction at 2 bits using vector quantization commutative with RoPE. QuantSpec [12] uses hierarchical Q4 KV cache for speculative decoding, validating 4-bit cache quality. We use 4-bit quantization with an end-to-end Q4 pipeline from disk through attention. Prior quantization work operates on in-memory single-session caches; we extend Q4 to persistent disk storage with cross-session reuse.

Agent memory. EM-LLM [8] organizes tokens into episodic events using Bayesian surprise (30.5% improvement over RAG on LongBench). A-MEM [27] organizes agent memories in Zettelkasten-style note networks. MemArt [2] introduces KV-cache-centric memory with reusable blocks (91–135× prefill reduction). We focus on per-agent isolation and cross-phase persistence rather than external knowledge injection. MemArt targets datacenters and lacks Q4 quantization or disk persistence.

Multi-agent KV systems. KVCOMM [28] enables cross-context KV sharing for multi-agent systems (7.8× speedup, >70% cache reuse). KVFlow [22] uses workflow-aware cache eviction (2.19× concurrent speedup). Both target datacenter deployments. PROMPTPEEK [19] shows that shared KV caches enable 99% prompt reconstruction attacks, which motivates per-agent isolation.

Edge inference. KVSwap [29] offloads KV cache to disk on mobile devices for long-context inference. Kelle [18] co-designs KV cache with eDRAM for custom edge accelerators (3.9× speedup) but requires specialized hardware. Perez et al. [23] benchmark local LLM inference on Apple Silicon without addressing multi-agent cache management. Krul [25] optimizes on-device LLM deployment but does not address KV persistence.

7 Conclusion

Persistent Q4 KV cache turns agent context restoration from a compute-bound O(n) prefill into an I/O-bound O(1) reload. On Gemma 3 12B at 32K context, hot cache reduces TTFT from 165 seconds to 1.3 seconds (130×). On DeepSeek-Coder-V2-Lite at 32K, from 48 seconds to 652 ms (74×). Warm disk reload achieves 102× and 69× at 32K.

Q4 quantization fits 4× more agents into fixed device memory than FP16 (12 vs 3 agents at 8K context on 24 GB), with measured quality impact of -0.10 to $+0.19$ PPL (Section 4.7). Batched serving reaches 22 system TPS (Gemma) and 65 system TPS (DeepSeek) with two warm-cache agents at 1K context.

Two multi-agent scenarios validate the design. A 5-phase interrogation shows 1.9× TTFT reduction in later phases from cross-phase persistence (23% total wall-time reduction). A 10-expert routing benchmark shows 24× TTFT reduction when querying cached experts.

The system exposes an OpenAI-compatible API, so any framework issuing chat completion requests can use persistent agent cache without modification.

Multi-device cache transfer, adaptive quantization bit-width (2-bit via RotateKV or CommVQ techniques), and porting to CUDA/RTX for discrete GPU edge devices are directions for future work.

Open-source at `[repository URL]`.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in llm inference with sarathi-serve. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

[2] Anonymous. Kvcache-centric memory for llm agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YolJOZOGhI. Submitted to ICLR 2026.

[3] Wayner Barrios. Native llm and mllm inference at scale on apple silicon. *arXiv preprint arXiv:2601.19139*, 2026.

[4] Yifan Chang et al. Sagallm: Multi-agent orchestration with transaction-based context management. *Proceedings of the VLDB Endowment*, 2025.

[5] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In *International Joint Conference on Artificial Intelligence*, 2024.

[6] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[7] Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, 2024. Spotlight.

[8] Yi Jiang et al. Em-llm: Human-inspired episodic memory for infinite context llms. In *International Conference on Learning Representations*, 2025.

[9] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Fan, Xin Zhu, Xuanzhe Luo, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *ACM Transactions on Computer Systems*, 2024.

[10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[11] Hanchen Li, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn llm agent scheduling with kv cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025. URL https://arxiv.org/abs/2511.02230.

[12] Minjae Li, Aditya Tomar, and Amir Gholami. Quantspec: Self-speculative decoding with hierarchical quantized kv cache. In *Proceedings of the 42nd International Conference on Machine Learning*, PMLR, 2025.

[13] Shikai Li, Yuke Zhang, Xin Zhao, Zhijie Zheng, Xiaopeng Yang, and Xiaolong Ding. Commvq: Commutative vector quantization for kv cache compression. In *Proceedings of the 42nd International Conference on Machine Learning*, PMLR, 2025.

[14] Zhenyu Li et al. Fusion rag cache: Optimizing retrieval-augmented generation with persistent kv states. *arXiv preprint*, 2025. Reports prefill = 95.53% of RAG inference time.

[15] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[16] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32332–32344. PMLR, 2024.

[17] LMCache Team. Lmcache: Engine-agnostic persistent kv store for llm serving. https://lmcache.ai/tech_report.pdf, 2025.

[18] MICRO 2025 Authors. Kelle: Co-design kv caching and edram for edge computing. In *IEEE/ACM International Symposium on Microarchitecture*, 2025. URL https://dl.acm.org/doi/10.1145/3725843.3756071.

[19] NDSS 2025 Authors. I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving. In *Network and Distributed System Security Symposium*, 2025. URL https://www.ndss-symposium.org/ndss-paper/i-know-what-you-asked-prompt-leakage-via-kv-cache-sharing-in-multi-tenant-llm-serving/

[20] Jakob Nielsen. *Usability Engineering*. Academic Press, Boston, MA, 1993. Chapter 5: Response Times: The 3 Important Limits (0.1s, 1.0s, 10s).

[21] Jakob Nielsen. The need for speed in the age of ai. Nielsen Norman Group, 2024. Argues no current AI system meets the 1-second response time threshold.

[22] Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. Kvflow: Efficient prefix caching for accelerating llm-based multi-agent workflows. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2507.07400.

[23] Daniel Perez et al. Production-grade local llm inference on apple silicon. *arXiv preprint arXiv:2511.05502*, 2025.

[24] Mingming Tao et al. Rotatekv: Accurate and robust 2-bit kv cache quantization for llms via outlier-aware adaptive rotations. In *International Joint Conference on Artificial Intelligence*, 2025. URL https://arxiv.org/abs/2501.16383.

[25] Wei Wen et al. Krul: Optimizing on-device large language model deployment. *arXiv preprint*, 2025.

[26] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[27] Wujiang Xu, Zujie Zhang, et al. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2502.12110.

[28] Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. Kvcomm: Online cross-context kv-cache communication for efficient llm-based multi-agent systems. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2510.12872.

[29] Huawei Zhang, Chunwei Xia, and Zheng Wang. Kvswap: Disk-aware kv cache offloading for long-context on-device inference. *arXiv preprint arXiv:2511.11907*, 2024.

[30] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

[31] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation*, 2024.

A safetensors Q4 Format

The persistent KV cache uses safetensors format with model-specific tensor naming. For a model with L layers, H KV heads, head dimensions D_K and D_V, and N cached tokens:

Tensor schema (per layer l, per block b):

- `cache.layers.{l}.key_cache.{b}.data`: uint32, shape $(H, D_K/8, 256)$
- `cache.layers.{l}.key_cache.{b}.scales`: float16, shape $(H, D_K, 4)$
- `cache.layers.{l}.key_cache.{b}.biases`: float16, shape $(H, D_K, 4)$
- `cache.layers.{l}.value_cache.{b}.data`: uint32, shape $(H, D_V/8, 256)$
- `cache.layers.{l}.value_cache.{b}.scales`: float16, shape $(H, D_V, 4)$
- `cache.layers.{l}.value_cache.{b}.biases`: float16, shape $(H, D_V, 4)$

Block size is 256 tokens, group size 64 elements (4 groups per block). For symmetric models (Gemma), $D_K = D_V$. For MLA models (DeepSeek), $D_K = 192$, $D_V = 128$.

Gemma 3's scales and biases use bfloat16 (preserved natively by MLX's `mx.save_safetensors`). DeepSeek uses standard float16.

B MLX Engineering Notes

MLX uses lazy evaluation: operations build computation graphs that execute only when results are consumed. This creates failure modes when KV cache operations appear to succeed but produce no data.

Table 11: MLX lazy evaluation failure modes relevant to KV cache persistence.

Symptom	Root Cause	Fix
Cache appears empty after prefill	Missing `mx.eval()` after cache update	Evaluate after update
OOM during batch	Graph accumulates without clearing	Evaluate per iteration
Zeros after disk reload	mmap buffer not evaluated	Evaluate after load
Quantization corruption	Scales/biases lazy	Evaluate quantize output
Attention NaNs	Q4 tensors invalid post-load	Validate dtype/shape
Batch hangs	Merge built graph but not executed	Evaluate before attention

Two additional issues specific to batched inference:

Thread safety. MLX is not thread-safe (GitHub issues #2067, #2133, #3078). Concurrent `mx.eval()` calls from different threads cause Metal assertion failures. We serialize all MLX operations through a single scheduler thread, using an RLock (`mlx_io_lock`) to protect cross-thread I/O (cache saves).

mx.compile with variable batch size. `mx.compile` traces shapes at first call and fails on subsequent calls with different batch dimensions. We split batch-2 operations into two batch-1 calls, each through `mx.compile(shapeless=True)`, and concatenate results.

C Benchmark Configuration

Hardware: Apple Mac Mini M4 Pro (MX2E3LL/A), 14-core CPU (10P+4E), 20-core GPU, 16-core Neural Engine, 24 GB LPDDR5X, 273 GB/s, 512 GB SSD (APFS).

Software: macOS Sequoia 15.2, Python 3.12.0, MLX 0.22.0, mlx-lm 0.30.4, Transformers 4.57.6.

Models:

- Gemma 3 12B Instruct: 48 attention layers, 8 KV heads, head dim 256 (symmetric), GQA with 8 global + 40 sliding-window layers (window 1024)
- DeepSeek-Coder-V2-Lite 16B Instruct: 27 layers, 16 KV heads, K dim 192 / V dim 128 (asymmetric MLA), MoE with 2/6 active experts

Parameters: Temperature 0.0 (greedy), output length 64 tokens, Q4 quantization (group size 64), prefill chunk size 256 tokens. Scheduler enabled, max batch size 2. 3 passes per configuration, 30–240s adaptive cooldown between passes (thermal-aware). Median values reported.

198 measurements per model (6 context lengths $\times$ 3 cache states $\times$ 2 batch sizes $\times$ 2 modes [streaming/non-streaming] $\times$ 3 passes, divided by 3 for median = 66 unique configurations, $\times$ 3 passes = 198) plus 6 staggered measurements (3 sequential + 3 batched).

D FP16 vs Q4 Memory Analysis

Gemma 3 12B (48 layers, 8 KV heads, head dim 256, group size 64):

FP16 per-layer cost = $2 \times 8 \times 256 \times n \times 2$ bytes (K+V, each 2 bytes per element).

At $n = 4096$: $2 \times 8 \times 256 \times 4096 \times 2 = 33{,}554{,}432$ bytes = 32 MB per layer $\times$ 48 layers = 1,536 MB.

Q4 per-layer cost: packed 4-bit data = hdn bytes, plus float16 scales and biases = $8hdn/g$ bytes, where h=8, d=256, g=64.

At $n = 4096$: packed data = $8{,}388{,}608$ bytes (4-bit, half of FP16), scales+biases = $1{,}048{,}576$ bytes. Total = $9{,}437{,}184$ bytes = 9.0 MB per layer $\times$ 48 layers = 432 MB.

Ratio: $432/1536 = 0.281$, matching the analytical formula.

DeepSeek-Coder-V2-Lite 16B (27 layers, 16 KV heads, K=192, V=128):

FP16: K cost = $16 \times 192 \times n \times 2$, V cost = $16 \times 128 \times n \times 2$. At $n = 4096$: K = 25,165,824 bytes, V = 16,777,216 bytes. Per layer = 40 MB. $\times$ 27 layers = 1,080 MB.

Q4: same 0.281 ratio applied per tensor. Total = 304 MB.

MoE intermediate tensors add $\sim$1–2 GB overhead during forward passes, further constraining FP16 capacity. The 4096 MB cache budget for DeepSeek accounts for this.

Table 12: Agent capacity comparison, both models. M4 Pro, 10.2 GB cache budget.

	4K		8K		16K		32K	
Model	FP16	Q4	FP16	Q4	FP16	Q4	FP16	Q4
Gemma 3	6	24	3	12	1	6	0	3
DeepSeek	9	33	4	16	2	8	1	4

E Perplexity Methodology

Section 4.7 reports perplexity measured with actual `QuantizedKVCache` objects. The methodology uses WikiText-2 text in 512-token sliding windows (256-token stride), evaluating 7,935 tokens per model. Both FP16 baseline and Q4 caches use identical model weights (4-bit quantized via mlx-lm). The Q4 KV cache uses group size 64, matching the production pipeline.

Prior work on Q4 KV cache quality: KIVI [16] reports $<$0.1 PPL degradation with per-channel key quantization (group 32–128). KVQuant [6] shows $<$0.1 PPL at 4 bits with per-layer sensitivity calibration. QuantSpec [12] validates 4-bit KV for speculative decoding with no measurable quality loss. RotateKV [24] reports $<$0.3 PPL even at 2 bits. Our Gemma result (-0.7%) is consistent with this literature; our DeepSeek result ($+3.1\%$) is higher, possibly reflecting MLA's compressed latent representations (Section 4.7).

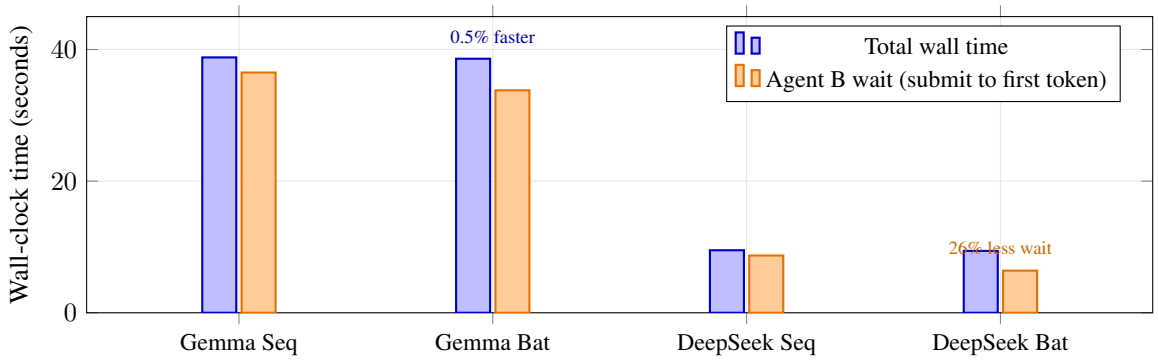

Figure 3: Staggered request arrivals (4K cold context, Agent B arrives 2s after Agent A). Total wall time is similar between sequential and batched modes for both models. Agent B benefits from batched scheduling because it begins prefill immediately rather than waiting for Agent A's decode phase to complete. The effect is larger on DeepSeek due to shorter prefill relative to decode.

F Staggered Arrivals

Real multi-agent workflows have staggered request arrivals. A single user may trigger multiple agents in sequence: Agent A begins at t=0 (4K cold context), Agent B begins at t=2s (4K cold context).

In sequential mode, Agent B waits for Agent A to complete before starting. In batched mode, Agent B joins Agent A's batch and begins prefill immediately.

For Gemma, total wall time is similar (38.8 s sequential vs 38.6 s batched). For DeepSeek, also similar (9.5 s vs 9.4 s). The benefit of batching appears in Agent B's perceived wait (time from experiment start to B's first token): DeepSeek Agent B waits 6.4 s (batched) vs 8.7 s (sequential), a 26% reduction. For Gemma, the reduction is 7% (33.8 s vs 36.5 s). The effect is smaller for Gemma because its longer prefill dominates. With warm or hot caches, the staggered benefit would be larger since prefill overhead vanishes and decode interleaving matters more.

G Hardware Landscape

All results in this paper are from a single device: Apple Mac Mini M4 Pro (24 GB, 273 GB/s). This represents the lower end of capable edge devices. The M4 Max (128 GB, 546 GB/s) would fit 135+ agents at 8K Q4 context vs 12 on the M4 Pro. NVIDIA's DGX Spark (128 GB, 273 GB/s) matches the M4 Pro's bandwidth at 5$\times$ the memory. Discrete GPU devices (RTX 5090, 32 GB VRAM at 1,792 GB/s) have higher compute bandwidth but spilling KV cache to host RAM drops to PCIe speeds (28$\times$ cliff), making unified memory devices more favorable for cache persistence workflows. Table 1 in the main text summarizes the key specifications.

H Detailed Figures

H.1 Architectural Comparison

Figure 4 compares the two model architectures. Gemma 3 uses hybrid attention (8 global + 40 sliding window layers) with symmetric KV dimensions. DeepSeek uses MLA with asymmetric dimensions (K=192, V=128). Both share the same block pool and Q4 pipeline via the ModelCacheSpec abstraction.

H.2 Phase Timeline

Figure 5 shows cache state transitions across the 5-phase prisoner's dilemma scenario. Permanent agents (Warden, Marco, Danny) accumulate warm/hot cache across phases. The ephemeral agent (Analyst) cold-starts in Phase 5 only.

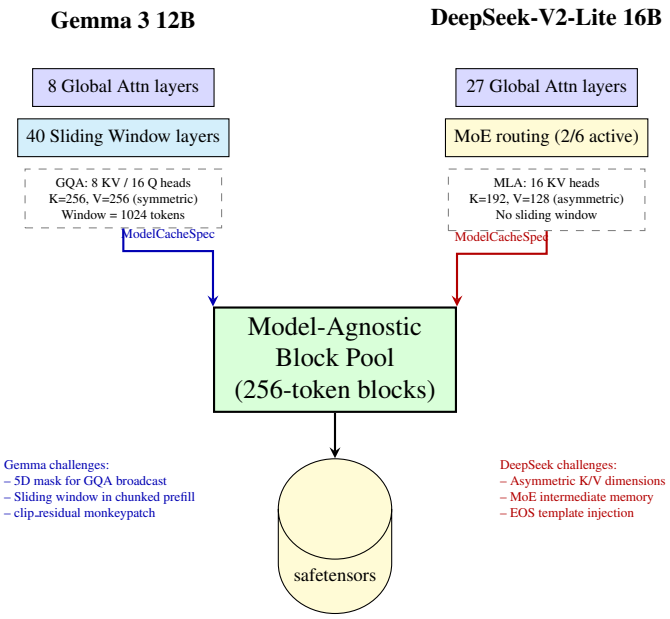

Figure 4: Architecture comparison. The block pool abstracts away architectural differences through ModelCacheSpec. Gemma 3 uses grouped-query attention with hybrid sliding-window layers, requiring 5D mask expansion and window-aware chunked prefill. DeepSeek uses multi-latent attention with asymmetric K/V dimensions (192 vs 128) and MoE routing, requiring larger memory budgets for intermediate tensors.

H.3 Wikipedia Routing Diagram

Figure 6 shows the 3-phase routing protocol. Phase 1 primes 10 experts with cold-start prefill. Phase 2 routes cross-topic queries to 2–3 warm experts each. Phase 3 re-queries hot experts.

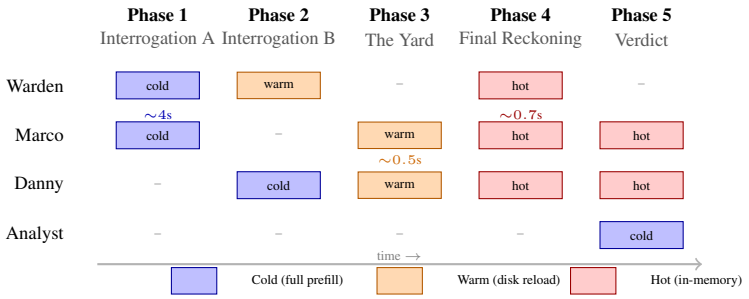

Figure 5: Agent cache state across prisoner's dilemma phases. Permanent agents (Warden, Marco, Danny) start cold and transition to warm/hot as context accumulates via cross-phase injection. Each phase extends the cached prefix rather than re-computing. The Analyst appears only in Phase 5 (cold start). TTFT annotations show projected latency from Table 3 at equivalent context lengths.

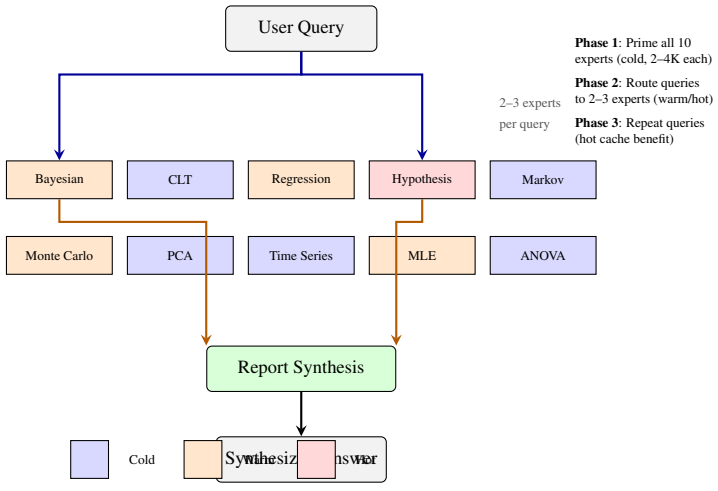

Figure 6: Wikipedia multi-agent routing. Ten expert agents are primed with article content (cold prefill). Cross-topic queries route to 2–3 relevant experts whose caches are warm/hot from priming. A reporter agent synthesizes responses. Repeated queries to the same experts benefit from hot cache (projected 10–30× TTFT reduction vs cold).