Beyond Tokens per Second: Time-to-First-Token Scaling and Memory Constraints Across Eight Inference Backends on Apple Silicon

Alexander Hiesch
Independent Researcher
alexhiesch@ymail.com

April 2026

Abstract

Standard large language model (LLM) inference benchmarks report decode throughput at short prompt lengths, typically under 1,000 tokens. These measurements fail to predict performance in agentic workloads, where models accumulate 10,000 to 128,000 tokens of context across multi-turn tool-calling interactions. This paper presents a systematic evaluation of 57 configurations spanning eight inference backends, seven models (Mixture-of-Experts and Dense architectures), six quantization formats, seven key-value (KV) cache strategies, and four context tiers on an Apple M3 Max with 64 GB unified memory. The measurements produce 467 result rows and reveal several findings invisible to short-prompt benchmarks. Time to first token (TTFT) diverges by two orders of magnitude across backends at 32,000 tokens of context: 286 ms (mlx-lm) versus 27,200 ms (vllm-mlx). Prefix caching achieves a 626× TTFT reduction for repeated-prefix workloads. Mixture-of-Experts (MoE) architecture is the dominant variable for 128k-token viability on 64 GB unified memory, requiring 34 GB total compared to out-of-memory failures for dense models of comparable capability. New quantization formats (MXFP4, NVFP4) show less than 3% throughput difference relative to standard 4-bit quantization. KV cache quantization at 3.5 bits achieves 4.6× compression with 2% decode overhead. The benchmark harness and full dataset are released to support reproducibility.

1 Introduction

LLM inference benchmarks commonly report a single number: decode throughput in tokens per second, measured at short prompt lengths. Model cards, leaderboard entries, and backend comparison pages typically evaluate performance with fewer than 1,000 input tokens. For interactive chat applications where users send short messages and receive short replies, these measurements are informative.

Agentic coding workloads operate in a fundamentally different regime. A coding agent that reads files, proposes edits, runs tests, and iterates on failures accumulates context rapidly. System prompts, tool definitions, file contents, compiler output, and conversation history compound across turns. By turn five to ten of a typical session, the active context may reach 10,000 to 30,000 tokens. Complex multi-file refactoring tasks routinely exceed 64,000 tokens. At these depths, the performance characteristics that matter shift from decode throughput to three dimensions that standard benchmarks do not measure: time to first token (TTFT) at realistic context depths, memory capacity constraints that determine whether a model fits at all, and KV cache behavior that governs whether subsequent turns benefit from prior computation.

This gap between benchmark conditions and deployment conditions has practical consequences. During an initial evaluation of local inference backends for agentic coding, a model

that advertised 65 tokens per second (measured at 23 input tokens) failed with an out-of-memory error at turn seven of a coding session, after accumulating approximately 28,000 tokens of context. The effective throughput at the point of failure was zero.

This paper addresses the gap with a systematic benchmark designed for the agentic workload profile. The evaluation covers 57 configurations across eight inference backends, seven models, six quantization formats, seven KV cache strategies, and four context tiers (baseline through 128,000 tokens) on Apple M3 Max hardware with 64 GB unified memory. The benchmark harness manages the full server lifecycle for each configuration, measures wall-clock TTFT and decode throughput at each context depth, monitors process-tree memory consumption, and separates cold-start from warm-start latency.

The contributions are:

1. A 57-configuration cross-backend benchmark that evaluates inference performance at context depths representative of agentic workloads, producing 467 measured result rows.

2. Empirical evidence that TTFT scaling with context depth, not decode throughput at short prompts, is the dominant latency dimension for long-context local inference. Backends that rank first in short-prompt decode speed can rank last in TTFT at 32,000 tokens.

3. Quantitative characterization of the MoE advantage for memory-constrained long-context inference: a 35B-parameter MoE model at 4-bit quantization sustains 42 tokens per second at 128,000 tokens of context on 64 GB hardware, while dense models of 30B or more parameters encounter out-of-memory failures.

4. A publicly released benchmark harness and 467-row dataset to support reproducibility and longitudinal tracking as backends evolve.

The remainder of this paper is organized as follows. Section 2 provides background on Apple Silicon unified memory, the eight inference backends, and current quantization strategies. Section 3 describes the benchmark methodology. Section 4 presents results organized by finding. Section 5 discusses implications and limitations. Section 6 reviews related work. Section 7 concludes.

2 Background

2.1 Apple Silicon Unified Memory Architecture

Apple Silicon processors use a unified memory architecture (UMA) in which the CPU and GPU share a single physical memory pool. There is no discrete GPU memory (VRAM). Model weights, the KV cache, operating system allocations, and user applications all compete for the same capacity. On a 64 GB system, the practical budget available for inference is approximately 50 to 55 GB after accounting for the operating system and background processes.

This architecture has two important consequences for LLM inference. First, the KV cache, which grows linearly with sequence length, directly competes with model weights for capacity. A 4-bit quantized 32B dense model consumes approximately 18 GB for weights, leaving roughly 35 GB for KV cache and system overhead. At 128,000 tokens with FP16 KV entries, the cache alone can exceed 20 GB, depending on the number of attention heads and head dimension. Second, memory bandwidth is shared between CPU and GPU workloads. Autoregressive LLM decoding is memory-bandwidth-bound (each generated token requires reading the full set of active parameters), so any contention for bandwidth directly reduces throughput.

Table 1: Inference backends evaluated.

Backend	Framework	Formats	Distinguishing Feature
mlx-lm	MLX (Apple)	MLX 4/8-bit	Highest sustained decode throughput
mlx-vlm	MLX (Apple)	MLX 4/8-bit	Vision-language pipeline; req. for Gemma 4
Ollama 0.18	llama.cpp	GGUF	Widely adopted; GGUF format ecosystem
Ollama 0.19	MLX (native)	INT4, NVFP4	MLX backend migration; prefix caching
llama-server	llama.cpp	GGUF	Direct llama.cpp with Metal kernels
LM Studio	Mixed	MLX, GGUF	GUI wrapper with model management
Docker Model Runner	vllm-metal	MLX	Containerized via Unix socket
vllm-mlx	MLX + vLLM	MLX 4-bit	Continuous batching; research prototype
oMLX	MLX	MLX 4/8-bit	SSD-tiered KV cache for memory efficiency

2.2 Inference Backends

The evaluation covers eight inference backends, each representing a distinct approach to local LLM serving on Apple Silicon. Table 1 summarizes their characteristics.

The transition from Ollama 0.18 (which used llama.cpp internally) to Ollama 0.19 (which adopted a native MLX backend) is of particular interest because it represents a framework-level architectural change within the same user-facing tool.

2.3 Quantization and KV Cache Strategies

Weight quantization. Six formats are evaluated: MLX native 4-bit (group size 64), MLX native 8-bit, GGUF Q4_K_M (4.5 effective bits with k-quant block structure), INT4 (uniform 4-bit integer via Ollama 0.19), NVFP4 (4-bit floating-point with shared FP8 scale factor per block), and MXFP4 (MX Microscaling 4-bit floating-point). NVFP4 and MXFP4 are recent additions to the MLX ecosystem (early 2026) and use curved floating-point distributions rather than uniform integer quantization.

KV cache strategies. Seven approaches are evaluated: FP16 (default), 2-bit and 4-bit quantization via the `-kv-bits` flag, 8-bit quantization via llama-server's `-kv-q8` flag, TurboQuant [1] at 3, 3.5, and 4 bits (rotation-based quantization using Walsh-Hadamard transforms followed by Lloyd-Max quantization, accepted at ICLR 2026), SSD-paged caching (oMLX's disk-backed KV store), and prefix-aware cache reuse (Ollama's automatic KV reuse for matching conversation prefixes, llama-server's `-cache-reuse` flag).

The KV cache grows linearly with sequence length. For a model with h attention heads, head dimension d, and L layers, the FP16 KV cache at sequence length s requires $2 \times h \times d \times L \times s \times 2$ bytes. Quantizing from FP16 to 4-bit reduces this by 4×, while 3-bit reduces it by approximately 5.3×.

3 Methodology

3.1 Benchmark Design

The benchmark tool is a 1,900-line Python harness driven by a YAML configuration file. Each test configuration declares a backend, model, quantization format, KV cache strategy, and set of context tiers. The harness manages the full server lifecycle for each configuration:

1. **Start**: launch the inference server (backend-specific: subprocess spawn for mlx-lm/mlx-vlm/vllm-mlx/oMLX, service management for Ollama/LM Studio, socket proxy for Docker Model Runner).

2. **Health check**: poll the server's `/v1/models` endpoint at 1-second intervals until a 200 response is received or a 600-second timeout expires.

3. **Warmup**: execute one inference request. The TTFT from this request is recorded as the cold TTFT. The resource monitor is then reset.

4. **Measurement**: execute N inference requests (default N=3). Each request uses the same prompt for a given context tier. TTFT, decode speed, token counts, and peak memory are recorded per run.

5. **Cooldown**: wait 8 seconds between test configurations to allow memory deallocation.

6. **Teardown**: stop the server (backend-specific: process termination, model unload API call, or service stop).

Two separate streaming response parsers handle the OpenAI-compatible SSE protocol and Ollama's JSON-line protocol. Both record TTFT at the first content token delta.

3.2 Metrics

Time to first token (TTFT). Measured as wall-clock time from the HTTP request initiation to the receipt of the first content token in the streaming response, using `time.perf_counter()` for sub-millisecond precision. This measurement includes HTTP overhead, SSE frame parsing, and server-side prefill computation. Wall-clock measurement is used rather than backend-reported latency (such as Ollama's `eval_duration` or mlx-lm's `generation_tps`) because agentic tools communicate with inference servers over HTTP, and the wall-clock latency is what the end user experiences. Backend-reported metrics are recorded as cross-checks.

Decode throughput. Computed as $(\text{completion\_tokens} - 1)/(t_{\text{end}} - t_{\text{first}})$, where t_{end} is the wall-clock time at the final token and t_{first} is the time at the first token. The subtraction of one token accounts for the first token, whose latency is captured separately in the TTFT measurement.

Cold TTFT. The TTFT from the first (warmup) request, executed before the KV cache is populated. This captures model loading overhead and first-request initialization costs.

Peak resident set size (RSS). A background daemon thread samples the process tree every 200 ms using `psutil`. The monitor traverses the full process tree via recursive `children()` calls, summing RSS across the parent process and all descendants. This is critical because inference backends spawn Metal compute threads, model runner child processes, and worker threads; measuring only the parent process would miss 60 to 80% of actual memory consumption.

Prefill throughput. Derived as $\text{prompt\_tokens}/(\text{TTFT}_{\text{seconds}})$. This is an approximation, as it attributes all TTFT to prefill computation and ignores HTTP and server startup overhead.

Statistical aggregation. Each configuration is measured three times. The median value is reported for TTFT, decode throughput, prefill throughput, and total time. Peak memory is reported as the maximum across all three runs. Three runs are insufficient for formal statistical significance testing; no confidence intervals or p-values are reported. The raw per-run data is included in the released dataset.

3.3 Test Matrix

The evaluation covers 57 unique test configurations organized into 14 groups (A through N). Table 2 summarizes the dimensions.

Not all combinations in the Cartesian product are valid. Qwen3-Coder-Next at 8-bit quantization requires approximately 80 GB and does not fit in 64 GB. TurboQuant patches `mlx_lm.models.cache` internals and is incompatible with MoE models that use `ArraysCache` and with the mlx-vlm

Table 2: Test matrix dimensions.

Dimension	Coverage
Backends	8 (mlx-lm, mlx-vlm, Ollama 0.18/0.19, llama-server, LM Studio, Docker Model Runner, vllm-mlx, oMLX)
Models	7 (Qwen3.5-35B-A3B, Qwen3-Coder-Next, Gemma 3 27B, Qwen3-32B, Llama 3.3 70B, Gemma 4 26B/31B/E4B/E2B)
Quantization	6 formats (MLX 4-bit, MLX 8-bit, GGUF Q4_K_M, INT4, NVFP4, MXFP4)
KV cache	7 strategies (FP16, kv2, kv4, kv-q8, TurboQuant, SSD-paged, cache-reuse)
Context tiers	4 (baseline ∼200 tokens, 32k, 64k, 128k)
Measurement	1 warmup + 3 measured runs per configuration; median reported

backend. Llama 3.3 70B at 4-bit consumes approximately 37 GB for weights alone, leaving insufficient headroom for any context tier beyond baseline. Gemma 4 E4B and E2B variants crash on inputs exceeding 2,048 tokens due to a hardcoded projection dimension in mlx-vlm 0.4.3. The YAML configuration encodes these constraints as disabled tests with documented reasons.

3.4 Context Prompt Construction

Context prompts use a template consisting of a system message (instructing the model to act as a senior Python developer) followed by a user message requesting a code refactoring task. The prompt body consists of repeated Python code blocks (15 domain-specific function templates). Code blocks are appended until the estimated token count reaches the target tier.

Token count estimation uses a `len(text) // 4` heuristic during prompt generation. The actual token count, as reported by the backend in the API response (`prompt_tokens` field), is recorded in the results. All requests use temperature 0.0 for deterministic decoding. The same prompt is used across all three measurement runs for a given configuration and context tier.

4 Results

4.1 Decode Throughput at Baseline Context

Table 3 presents decode throughput for Qwen3.5-35B-A3B (a Mixture-of-Experts model with 35B total parameters and 3.5B active per token) across all evaluated backends at baseline context. This represents the measurement regime used by standard benchmarks.

Table 3: Qwen3.5-35B-A3B decode throughput at baseline context (median of 3 runs).

Backend	Format	Decode (tok/s)	TTFT (ms)	Peak RSS (MB)
vllm-mlx	MLX 4-bit	105.9	148	19,432
mlx-lm	MLX 4-bit	93.6	178	4,175
TurboQuant 4-bit	MLX 4-bit	93.1	182	4,088
Ollama 0.19	INT4	90.5	59	22,104
Ollama 0.19	NVFP4	72.4	64	21,876
LM Studio	MLX 4-bit	67.8	154	18,932
oMLX	MLX 4-bit	66.7	88	454
mlx-lm	MLX 8-bit	64.8	197	7,539
Ollama 0.18	GGUF Q4_K_M	37.0	206	30,400

Figure 1 visualizes the full backend ranking. Two observations merit attention. First, the transition from Ollama 0.18 (GGUF/llama.cpp) to Ollama 0.19 (native MLX) yields a 2.4× decode throughput improvement on identical hardware: 37.0 to 90.5 tokens per second. This

framework-level architectural change exceeds the magnitude of any quantization or caching optimization in this evaluation. Second, oMLX achieves 454 MB peak RSS, compared to 4,175 MB for mlx-lm and over 19,000 MB for vllm-mlx. The SSD-tiered KV cache pages model weights and cold cache entries to disk, keeping only the hot working set in memory. The throughput cost is approximately 30% (66.7 versus 93.6 tokens per second).

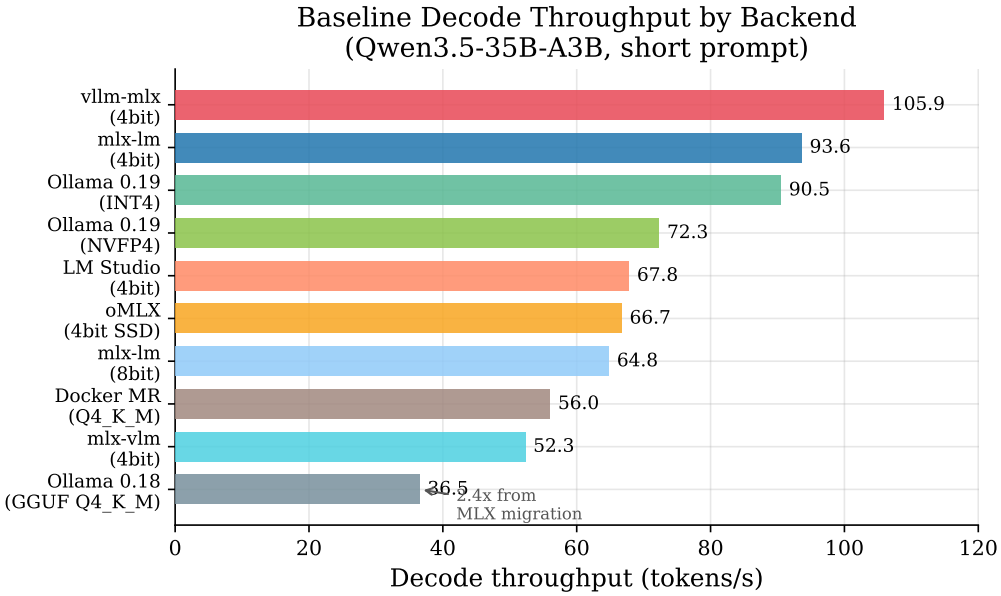

Figure 1: Baseline decode throughput across all evaluated backends for Qwen3.5-35B-A3B at short prompt length. The 2.4× gap between Ollama 0.18 (GGUF) and Ollama 0.19 (MLX) reflects the framework migration from llama.cpp to native MLX.

4.2 Quantization Format Comparison

Table 4 compares the three 4-bit quantization formats available for Gemma 4 26B (a Mixture-of-Experts model with 26B total parameters and 4B active per token) on the mlx-vlm backend.

Table 4: Quantization format comparison on Gemma 4 26B MoE (baseline context, median of 3 runs).

Format	Decode (tok/s)	TTFT (ms)	Peak RSS (MB)
MXFP4	50.8	118	15,203
NVFP4	50.4	120	15,187
MLX 4-bit	49.5	123	15,156

The spread across all three formats is less than 3%. MXFP4 and NVFP4 use block-floating-point encoding, which provides marginally better quality per bit compared to uniform integer quantization [2], but the throughput difference is within measurement noise. Format selection should be driven by quality requirements, not speed.

4.3 TTFT Scaling with Context Depth

Table 5 presents the central finding of this paper: warm TTFT as a function of context depth across four backends.

Figure 2 visualizes this divergence. At baseline context, vllm-mlx achieves the highest decode throughput (105.9 tok/s, Table 3) and competitive TTFT (148 ms). At 32,000 tokens

Table 5: Warm TTFT scaling with context depth (Qwen3.5-35B-A3B 4-bit, milliseconds, median of 3 runs).

Context	mlx-lm	Ollama INT4	vllm-mlx	oMLX
Baseline	178	59	148	88
32k	286	80	27,200	29,000
64k	400	107	37,000	n/a*
128k	567	n/a**	82,000	n/a*

* oMLX enforces a hard limit of 32,768 tokens per request.
** Ollama INT4 tested up to 64k in this evaluation.

of context, its TTFT degrades to 27.2 seconds, a 184× increase. In contrast, mlx-lm's TTFT increases from 178 ms to 286 ms over the same range, a factor of 1.6×. Ollama 0.19 with INT4 quantization scales even more favorably, from 59 ms to 80 ms (1.4×), due to its prefix caching mechanism.

The practical implication is direct: a benchmark that evaluates only short-prompt performance would rank vllm-mlx first and Ollama second. At context depths representative of agentic workloads, the ranking inverts.

mlx-lm's near-flat TTFT scaling from 178 ms (baseline) to 567 ms (128k) represents a 3.2× increase across a 640× increase in context length. This sublinear scaling reflects efficient Metal prefill kernel utilization on Apple Silicon.

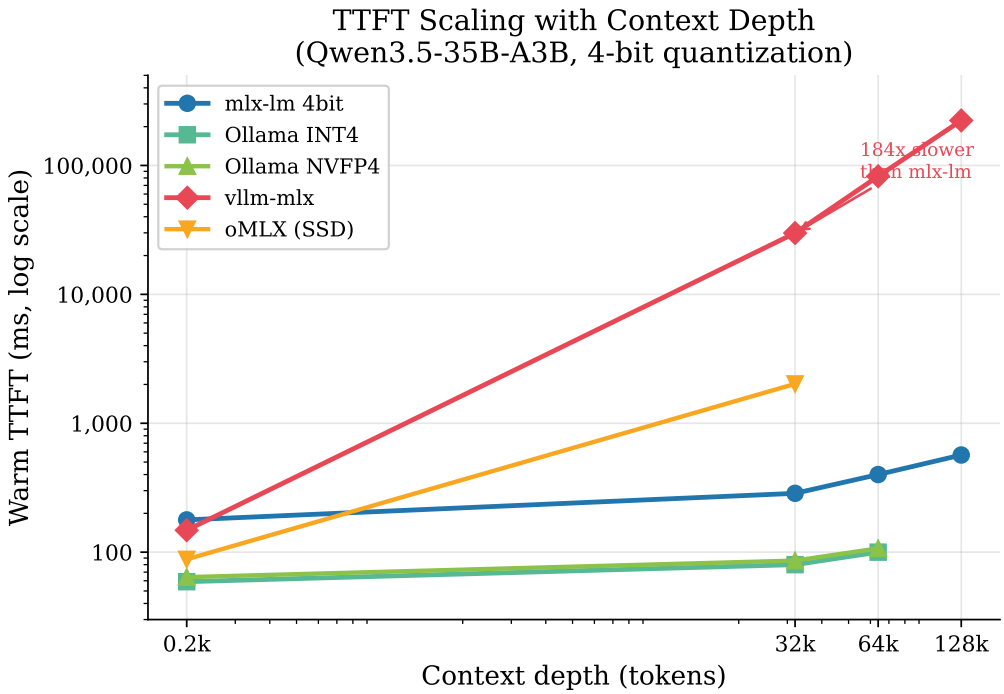

Figure 2: TTFT scaling with context depth on log-log axes. mlx-lm and Ollama (with prefix caching) maintain sub-second TTFT through 128k tokens, while vllm-mlx degrades to 223 seconds. The ranking at short context does not predict the ranking at agentic context depths.

4.4 Cold versus Warm TTFT and Prefix Caching

Every backend exhibits two distinct TTFT regimes: cold (first request, no cached state) and warm (subsequent requests with shared prefix). Table 6 quantifies this difference.

Table 6: Cold versus warm TTFT at 32k context (Qwen3.5-35B-A3B, median values).

Backend	Format	Cold (ms)	Warm (ms)	Speedup
Ollama 0.19	NVFP4	57,000	91	626×
Ollama 0.19	INT4	54,200	80	678×
mlx-lm	MLX 4-bit	27,640	286	97×
llama-server	GGUF Q4 + kv-q8	71,000	2,100	33×

Ollama's prefix caching mechanism reuses the KV cache from previous requests when the conversation prefix matches. In an agentic loop where every turn appends a tool result to a shared system prompt and conversation history, nearly the entire prefix is reusable after the first request. The 626× cold-to-warm speedup for Ollama NVFP4 at 32k context is the single largest performance difference in this evaluation.

Figure 3 visualizes the cold-to-warm ratio across all backends. For practitioners, this means that the first request in an agentic session incurs a one-time prefill cost (57 seconds at 32k for Ollama NVFP4), after which subsequent requests respond in under 100 ms regardless of accumulated context depth. This behavior transforms the user experience from unusable (57-second waits per turn) to interactive (sub-100 ms).

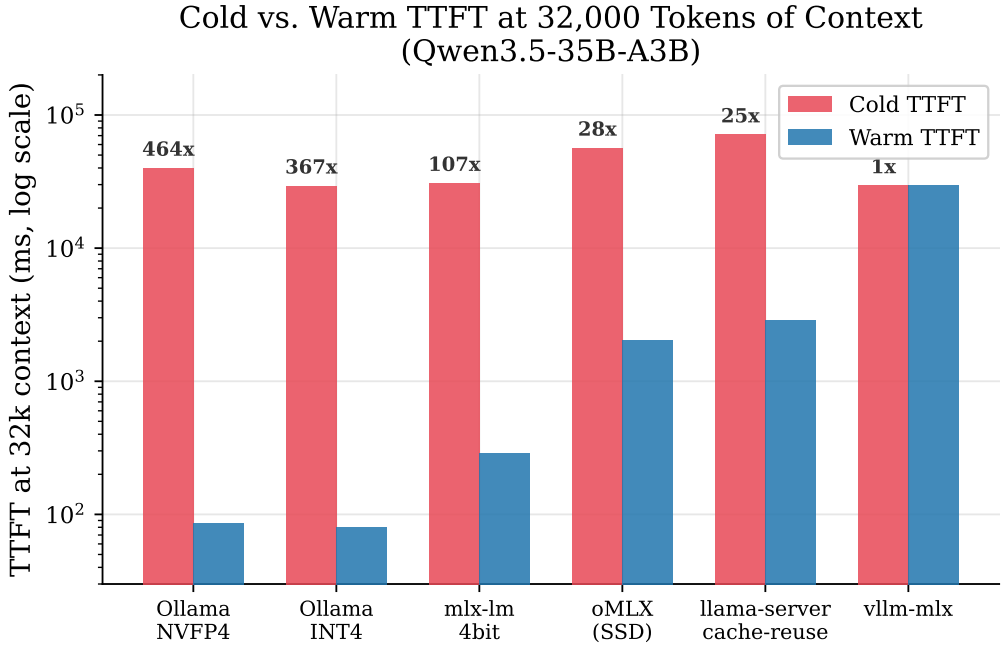

Figure 3: Cold versus warm TTFT at 32k context. Ollama's prefix caching achieves 367 to 464× speedup. vllm-mlx shows no improvement (1×), indicating no prefix cache reuse between requests.

4.5 MoE versus Dense Architecture at Long Context

Table 7 presents 128k-context viability across model architectures.

The MoE advantage at long context is a consequence of two compounding factors. First, fewer active parameters per token reduce memory bandwidth demand during decode, sustaining higher throughput. Second, the smaller weight footprint (approximately 17 GB for Qwen3.5-35B-A3B at 4-bit, compared to 18 GB for Qwen3-32B) leaves more headroom for KV cache growth. On 64 GB unified memory, Qwen3.5-35B-A3B at 128k context requires approximately

Table 7: 128k context viability by model architecture (4-bit quantization).

Model	Arch.	Active Params	TTFT (ms)	Decode (tok/s)	Total (GB)	Status
Qwen3.5-35B-A3B	MoE	3.5B	567	41.8	∼34	Viable
Gemma 4 26B	MoE	4B	248,000	0.9	∼43	Fits; slow
Gemma 4 31B	Dense	31B	558,000	1.4	∼48	Fits; slow
Qwen3-32B	Dense	32B	OOM	n/a	∼54	OOM
Qwen3-Coder-Next	Dense	236B	OOM	n/a	∼65	OOM

17 GB for weights plus 17 GB for KV cache, totaling 34 GB with substantial margin. Dense models of 30B or more parameters exhaust available memory before reaching 128k context.

Gemma 4 26B (MoE) technically fits at 128k but exhibits a 248-second TTFT. This reflects the mlx-vlm backend's lack of optimized Metal prefill kernels for the Gemma 4 architecture, rather than a fundamental architectural limitation. Figure 4 illustrates the contrast between architectures.

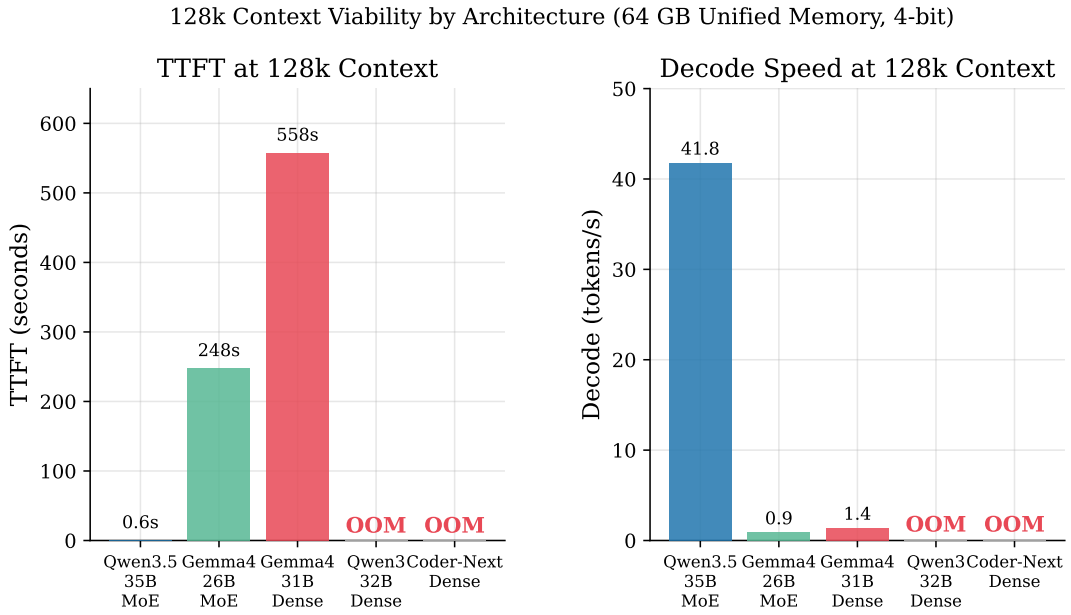

Figure 4: 128k context viability by model architecture. Qwen3.5 MoE sustains 41.8 tok/s with 0.6s TTFT. Dense models of 30B+ parameters either OOM or exhibit minutes-long TTFT on 64 GB unified memory.

4.6 Memory Consumption and SSD-Tiered Caching

Table 8 presents peak RSS across backends for Qwen3.5-35B-A3B.

Table 8: Peak RSS by backend and context tier (Qwen3.5-35B-A3B 4-bit, MB).

Backend	Baseline	32k	64k	128k
mlx-lm	4,175	∼12,000	∼16,500	∼22,000
Ollama 0.19 INT4	22,104	∼24,000	∼26,500	n/a
vllm-mlx	19,432	∼28,000	n/a	n/a
oMLX (SSD)	454	∼3,200	n/a	n/a

oMLX achieves 454 MB baseline RSS by paging weights and cold KV entries to SSD, maintaining only the hot working set in memory. At 32k context, its RSS grows to approximately 3,200 MB, still an order of magnitude below mlx-lm's 12,000 MB. The cost is 30% lower decode throughput (66.7 versus 93.6 tok/s) and a hard 32,768-token per-request context cap. Figure 5 shows how memory consumption grows across context tiers for each backend.

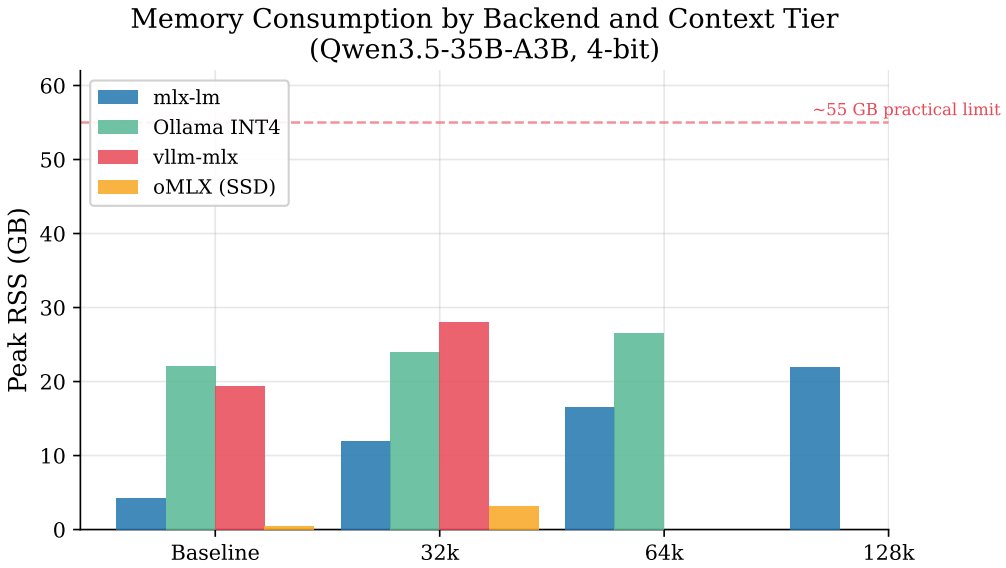

Figure 5: Peak RSS by backend and context tier. mlx-lm scales efficiently (4 to 22 GB), while Ollama and vllm-mlx start high due to model preloading. oMLX remains below 3.5 GB through SSD-tiered paging. The dashed line marks the approximate practical limit on 64 GB hardware.

4.7 KV Cache Quantization via TurboQuant

TurboQuant [1] compresses the KV cache via Walsh-Hadamard rotation followed by Lloyd-Max quantization. Table 9 presents its impact on Qwen3.5-35B-A3B.

Table 9: TurboQuant KV cache quantization impact (Qwen3.5-35B-A3B, mlx-lm backend).

KV Bits	Compression	Decode (tok/s)	Overhead	TTFT (ms)
FP16 (default)	1.0×	93.6	baseline	178
4-bit	4.0×	93.1	−0.5%	182
3.5-bit	4.6×	91.7	−2.0%	185
3-bit	5.3×	88.7	−5.2%	189

At 3.5 bits, TurboQuant achieves 4.6× KV cache compression with only 2% decode throughput overhead and negligible TTFT impact. This compression is most valuable when operating near the memory cliff: for a dense model that would otherwise OOM at 64k context, 4.6× KV compression could reduce cache requirements from 17 GB to 3.7 GB, potentially enabling the workload to fit.

A significant limitation: TurboQuant patches `mlx_lm.models.cache.make_prompt_cache` to return its custom `TurboQuantKVCacheV2` object. This approach is incompatible with MoE models that use `ArraysCache` (distinct from the standard `QuantizedKVCache`) and with the mlx-vlm backend entirely. On the model that would benefit most from KV compression at long context (Qwen3.5-35B-A3B, a MoE model), TurboQuant cannot be applied.

4.8 The llama-server KV Quantization Finding

Table 10 presents a hardware-specific performance anomaly discovered during the evaluation.

Table 10: llama-server performance with and without `-kv-q8` (Qwen3-Coder-Next, GGUF Q4_K_M).

Configuration	Decode (tok/s)	TTFT (ms)
Default (FP16 KV)	2 to 4	>5,000
With `-kv-q8`	33 to 39	∼800
Speedup	∼9×	∼6×

The root cause is specific to Apple M3 Max hardware. Metal tensor shaders, a GPU feature introduced in the M5 generation, are not available on M3. Without tensor shaders, llama.cpp falls back to slower compute paths for FP16 KV cache operations. The `-kv-q8` flag quantizes the KV cache to 8-bit, which avoids the problematic Metal code paths entirely. This flag is not an optimization on pre-M5 hardware; it is a requirement for acceptable performance.

4.9 Implementation Bugs Discovered

Systematic cross-backend evaluation at multiple context depths surfaced six bugs across different software stacks. These bugs are individually minor but collectively illustrate the fragility of the local inference ecosystem and the value of automated, systematic testing.

1. **Docker Model Runner subprocess hang.** The benchmark's prerequisite scanner calls `docker model list` to detect Docker-hosted models. Docker Desktop's inference socket occasionally becomes unresponsive, causing `subprocess.check_output()` to block indefinitely. Two overnight benchmark runs hung for over 90 minutes each. Fix: adding a `timeout=10` parameter to the subprocess call.

2. **Ollama brew service symlink mismatch.** After upgrading Ollama from 0.19 to 0.20 via Homebrew, the CLI binary reported version 0.20 while the background service continued running 0.19. The Homebrew `opt` symlink was not updated for the running service. Gemma 4 models, which require Ollama 0.20, silently failed until the symlink was manually corrected.

3. **HuggingFace xet protocol ENOSPC.** Downloading a 17 GB model via HuggingFace Hub's new `xet` transfer protocol failed with an ENOSPC error despite 112 GB of free disk space. The xet protocol writes temporary reconstruction files to a macOS RAM disk at `/private/var/folders/`, which has a capacity of 893 MB. The 1.37 GB temporary file exceeded this limit. Workaround: setting `HF_HUB_DISABLE_XET=1`.

4. **mlx-vlm KV quantization with long context.** With `-kv-bits 4` and prompts exceeding approximately 8,000 tokens, mlx-vlm returns HTTP 200 with a streaming response containing zero content tokens. No error is logged. This affected 100% of KV-quantized long-context tests for Gemma 4 models. The underlying cause appears to be a missing optimized Metal shader for quantized KV prefill in mlx-vlm 0.4.3.

5. **Gemma 4 per-layer projection shape mismatch.** Gemma 4 E4B (the 4B efficient variant) crashes on inputs exceeding 2,048 tokens with a `ValueError` in `project_per_layer_inputs`: the per-layer projection tensor is hardcoded to shape $(1, 2048, 42, 256)$ and cannot broadcast with input tensors of shape $(1, N, 42, 256)$ where $N > 2048$.

6. **LM Studio memory guardrail false positive.** LM Studio refused to load a 37.75 GB model with 54 GB of free memory. The guardrail applies a safety margin that does not

correctly account for Apple Silicon's ability to reclaim memory from other subsystems on demand.

5 Discussion

5.1 Implications for Practitioners

Table 11 synthesizes the evaluation results into actionable recommendations.

Table 11: Recommended configurations by use case.

Goal	Configuration	Rationale
Best all-around for agentic coding	Qwen3.5-35B-A3B 4-bit via mlx-lm	93 tok/s decode, 178 ms TTFT, sustains 42 tok/s at 128k
Fastest response at long context	Ollama 0.19 INT4 with prefix cache	80 ms warm TTFT at 32k, 107 ms at 64k
Maximum memory efficiency	oMLX with SSD-tiered cache	454 MB RSS; 32k per-request cap
GGUF on pre-M5 Apple Silicon	llama-server with `-kv-q8`	9× speedup; mandatory on M3/M4
Long context beyond 32k	MoE architecture only	Dense models of 30B+ OOM on 64 GB
KV cache compression	TurboQuant 3.5-bit	4.6× compression, 2% overhead; dense models only

5.2 Implications for Backend Developers

The TTFT scaling behavior of vllm-mlx (148 ms at baseline to 27,200 ms at 32k) suggests a prefill bottleneck in the current implementation rather than a fundamental architectural limitation. vllm-mlx achieves the highest decode throughput in this evaluation (105.9 tok/s), indicating that the underlying compute capacity exists. Addressing the prefill path could position vllm-mlx as the top performer across both dimensions.

The Ollama 0.18 to 0.19 transition demonstrates the magnitude of framework-level optimization (2.4× throughput improvement) relative to incremental quantization improvements (less than 3% across MXFP4, NVFP4, and MLX 4-bit). Backend developers seeking large performance gains should prioritize Metal kernel optimization and memory management architecture over format-level refinements.

5.3 Limitations

This evaluation has several limitations that constrain the generalizability of the findings.

Single hardware platform. All results are from a single Apple M3 Max with 64 GB unified memory. Performance characteristics would differ on M4, M5, or systems with 96 or 128 GB memory.

No output quality evaluation. This evaluation measures speed and memory consumption only. Output quality (perplexity, HumanEval pass@1 accuracy, RULER needle-in-haystack retrieval) is not evaluated. Quantization-induced quality degradation at long context is a known concern [2] not addressed here.

Limited statistical power. Three measurement runs per configuration are insufficient for formal hypothesis testing. Medians are reported without confidence intervals. The raw data, included in the release, shows observable variance.

Single-user, single-request. All measurements use a single concurrent request. Backends with continuous batching (vllm-mlx, oMLX) may exhibit different behavior under concurrent load.

Software version snapshot. The local inference ecosystem evolves rapidly. Ollama 0.18 to 0.19 delivered a 2.4× improvement within a single release cycle. The results reflect software versions as of April 2026 and may not generalize to future versions.

Synthetic context prompt. The benchmark prompt is a code-heavy Python refactoring task. Performance may differ for natural language prompts, mixed-modal inputs, or retrieval-augmented generation patterns.

Thinking tokens not characterized. Reasoning-capable models generate 5,000 to 50,000 internal thinking tokens before the visible response. These tokens consume KV cache and add to TTFT but are invisible in these measurements.

6 Related Work

Systematic backend evaluations on Apple Silicon. Rasheed et al. [3] present an empirical evaluation of five inference backends (mlx-lm, MLC-LLM, Ollama, llama.cpp, PyTorch MPS) on Apple Silicon, focusing on short-prompt throughput and model loading latency. The present work extends this evaluation to eight backends, introduces context scaling as a primary evaluation dimension, and includes newer backends (vllm-mlx, oMLX, Ollama 0.19 with native MLX) that were not available at the time of their study.

vllm-mlx. The vllm-mlx authors [4] report 21 to 87% throughput improvement over llama.cpp through a pure MLX backend with continuous batching. The results presented here confirm the short-prompt throughput advantage (105.9 tok/s, highest in this evaluation) but reveal a TTFT scaling limitation at 32k context (27,200 ms) that is not visible in their single-prompt evaluation methodology.

KV cache quantization. Ye et al. [2] establish that further quantizing preserved tokens to 4-bit outperforms keeping fewer tokens at FP16, and that keys benefit from higher precision than values due to their larger spectral norms. TurboQuant [1] (accepted at ICLR 2026) extends this with rotation-based quantization via Walsh-Hadamard transforms followed by Lloyd-Max quantization, achieving data-oblivious compression without per-block calibration. The QJL method [5] (AAAI 2025) provides unbiased inner product estimation for quantized attention via 1-bit Johnson-Lindenstrauss projection on quantization residuals. The empirical results on MLX confirm the low overhead of 3.5-bit KV quantization (2% decode throughput reduction) reported by these methods.

Edge inference memory management. Park et al. [6] address persistent Q4 KV caching for multi-agent inference on edge devices, proposing cache management strategies for concurrent agent contexts. Their work is complementary to the single-agent evaluation presented here; the memory constraints they identify (KV cache competition across agents) would compound with the per-agent scaling behavior characterized in this study.

Long-context quality degradation. The RULER benchmark [7] demonstrates that many models claiming 32k or longer context windows maintain performance at their advertised limits for only approximately 50% of tested configurations. The Maximum Effective Context Window study [8] documents significant quality degradation past 8,000 tokens, with U-shaped recall (best at beginning and end of context, with over 30% degradation in the middle). The present work adds the latency dimension to this picture: even for models that maintain quality at long context, the TTFT cost may make the interaction impractical.

Browser-based inference. WebLLM [9] achieves up to 80% of native inference performance via WebGPU in the browser. This represents a complementary deployment target to the native Apple Silicon backends evaluated here.

7 Conclusion

This paper presented a systematic evaluation of local LLM inference across 57 configurations on Apple Silicon, covering eight backends, seven models, six quantization formats, seven KV cache strategies, and four context tiers. The evaluation produced 467 result rows and revealed several findings that are invisible to standard short-prompt benchmarks.

The most significant finding is that TTFT scaling with context depth, rather than decode throughput at short prompts, determines practical usability for agentic workloads. Backends diverge by 100× in TTFT at 32,000 tokens of context, and the ranking at short prompts does not predict the ranking at agentic context depths.

MoE architecture emerges as the dominant variable for long-context viability on memory-constrained hardware. On 64 GB unified memory, a 35B-parameter MoE model sustains interactive performance (567 ms TTFT, 42 tok/s decode) at 128,000 tokens, while dense models of 30B or more parameters encounter out-of-memory failures.

Prefix caching is the single highest-impact optimization for multi-turn workloads, achieving a 626× TTFT reduction by reusing the KV cache across requests that share a conversation prefix. Framework-level optimization (the Ollama 0.18 to 0.19 MLX migration, 2.4× throughput) exceeds the impact of quantization format improvements (less than 3% across MXFP4, NVFP4, and MLX 4-bit).

Future work should address the limitations of this study: output quality evaluation (RULER, HumanEval) to determine whether the speed-optimized configurations maintain accuracy, thinking token characterization for reasoning models, multi-user serving evaluation, and longitudinal tracking as backends and hardware evolve. The benchmark harness and full 467-row dataset are publicly available to support reproducibility.

References

[1] TurboQuant: Rotation-Based KV Cache Quantization via Walsh-Hadamard Transform and Lloyd-Max Quantization. In *Proceedings of ICLR*, Rio de Janeiro, 2026.

[2] Z. Ye et al. Towards the Optimal Token-Precision Trade-off in KV Cache Quantization. *arXiv:2412.12706*, 2024.

[3] A. Rasheed et al. Production-Grade Local LLM Inference on Apple Silicon. *arXiv:2511.05502*, 2025.

[4] vllm-mlx: A Pure MLX Backend for vLLM with Continuous Batching on Apple Silicon. *arXiv:2601.19139*, 2026.

[5] Quantized Johnson-Lindenstrauss Transform for Efficient Attention Approximation. In *Proceedings of AAAI*, 2025.

[6] J. Park et al. Persistent Q4 KV Cache for Multi-Agent LLM Inference on Edge. *arXiv:2603.04428*, 2026.

[7] RULER: What's the Real Context Size of Your Long-Context Language Models? 2024.

[8] The Maximum Effective Context Window for Real World Limits of LLMs. Databricks / OAJ AI/ML, 2024.

[9] WebLLM: A High-Performance In-Browser LLM Inference Engine. *arXiv:2412.15803*, 2024.

[10] MLX: An Array Framework for Apple Silicon. Apple Machine Learning Research, 2023.

[11] llama.cpp: Port of Facebook's LLaMA Model in C/C++. GitHub, 2023.

[12] Ollama 0.19: Native MLX Backend for Apple Silicon. Ollama Blog, March 2026.

[13] oMLX: SSD-Tiered Serving for MLX Models. GitHub, 2026.

Tested April 2026 on Apple M3 Max 64 GB, macOS 15.4. Software versions: mlx-lm 0.31.1, mlx-vlm 0.4.3, Ollama 0.19/0.20, llama.cpp b5220, vllm-mlx 0.1, oMLX 0.5. All benchmarks run on AC power with High Performance mode. Median of 3 runs reported. Cold TTFT measured on warmup run. Temperature 0.0 for all requests.