Demystifying LLM Inference Optimization on Apple Silicon: Why Speculative Decoding Fails and Where the Real Bottlenecks Lie

AtomGradient

March 2026

Abstract

We present a systematic study of LLM inference optimization on Apple M2 Ultra using the MLX framework. Through extensive profiling and experimentation across 9 optimization directions, we reveal several counter-intuitive findings: (1) N-gram Prompt Lookup Speculative Decoding fails to accelerate inference on *both* pure Transformer (Qwen3-8B) and hybrid linear-attention architectures (Qwen3.5-9B), achieving 0.85–1.00× speedup due to memory-bandwidth-limited batch forward passes yielding only 1.79–2.57× actual speedup versus the theoretical 5×; (2) MLX already implements AOT shader compilation, single-dispatch attention kernels, and SIMD matrix operations—features previously assumed missing; (3) the prefill bottleneck is quantized GEMM in linear projections (57.6% of time), not the attention kernel (6.7%). We correct five misconceptions from prior optimization proposals and identify three high-value optimization paths: KV cache quantization threshold tuning (1-line change, 28% memory reduction), fused QKV projections (15–25% prefill improvement), and chunkwise parallel GatedDeltaNet prefill (3–10× prefill speedup).

1 Introduction

Apple Silicon's unified memory architecture offers a unique advantage for on-device LLM inference: CPU and GPU share high-bandwidth memory (M2 Ultra: 800 GB/s), eliminating the PCIe bottleneck of discrete-GPU systems. The MLX framework [1] exploits this design, achieving 85% memory bandwidth utilization during token generation—53% faster than llama.cpp [2] on the same hardware.

However, MLX's prompt processing (prefill) lags behind llama.cpp by 1.9–3.6×, suggesting significant optimization opportunities. A prior optimization proposal [7] identified 9 directions ranging from adaptive Flash Attention block sizes to Continuous Batching, projecting substantial speedups (e.g., "3–5× prefill from adaptive block sizes," "1.5–2× generation from speculative decoding").

In this work, we systematically evaluate each proposed optimization through code analysis, profiling, and end-to-end benchmarking. Our key contributions are:

1. **Speculative decoding evaluation on two architectures**: We implement N-gram Prompt Lookup Speculative Decoding with zero-cost cache checkpointing and test on both Qwen3.5-9B (hybrid GatedDeltaNet + Transformer) and Qwen3-8B (pure Transformer). Neither architecture benefits—the fundamental bottleneck is memory bandwidth, not compute.

2. **Correction of five misconceptions**: We demonstrate that MLX already implements AOT shader compilation, single-dispatch attention, and SIMD matrix multiply; that prefill bottlenecks are in quantized GEMM rather than attention; and that pure-Transformer batch speedup is far below theoretical predictions.

3. **Comprehensive prefill profiling**: We decompose prefill time by layer type, identifying that linear projections account for 57.6% of total time versus 6.7% for SDPA attention and 29.0% for GatedDeltaNet recurrence.

4. **Revised optimization roadmap**: Based on empirical evidence, we identify three actionable optimizations with validated feasibility assessments.

2 Background

2.1 Hardware Platform

Table 1: Apple M2 Ultra specifications.

Component	Specification
CPU	24 cores (16P + 8E)
GPU	76 Apple GPU cores, Metal 4
Unified Memory	192 GB, 800 GB/s bandwidth
SIMD Width	32 (simdgroup)
Per-core Registers	∼208 KB
Threadgroup Memory	∼32 KB

2.2 Model Architectures

Qwen3.5-9B uses a hybrid architecture with 32 layers:

- 24 layers (75%): GatedDeltaNet [4] (linear attention with gated delta rule)
- 8 layers (25%): Standard full attention (head_dim=256, GQA 4:1)

The GatedDeltaNet recurrence is:

$$S_t = g_t \cdot (I - \beta_t \cdot k_t k_t^\top) \cdot S_{t-1} + \beta_t \cdot k_t v_t^\top \tag{1}$$

where $S_t \in \mathbb{R}^{D_v \times D_k}$ is the state matrix, g_t is a per-head scalar gate, $\beta_t = \sigma(b_t)$ is the update rate, and k_t, v_t are L2-normalized key/value vectors.

Qwen3-8B is a standard Transformer with 36 layers of full attention (head_dim=128, GQA 4:1), all using KVCache.

2.3 Frameworks

MLX [1] (v0.31.0): Apple's ML framework with lazy evaluation, unified memory, and Steel GEMM/Attention kernels. Model loaded via `mlx-vlm` (0.3.12) for Qwen3.5 and `mlx-lm` (0.30.7) for Qwen3.

llama.cpp [2]: C++ inference engine with 103 Metal kernels, Flash Attention, and 15+ quantization formats. Uses GGUF model format (Q8_K_XL).

3 Speculative Decoding Experiments

3.1 Implementation

We implemented Prompt Lookup Speculative Decoding [3] with the following key design decisions:

Zero-cost cache checkpointing: GatedDeltaNet's `ArraysCache` does not support trim operations. We exploit the fact that GatedDeltaNet creates *new* array objects on each forward step (via `mx.concatenate` and `gated_delta_update`), so saving Python references to old arrays constitutes a valid checkpoint. Measured overhead: save=0.01 ms, restore=0.04 ms.

Async pipeline preservation: The standard generation path uses `mx.async_eval` to overlap GPU computation with CPU token output. Our speculative path only intervenes when an N-gram match is found, ensuring zero overhead on the common (no-match) path.

Batch sampling: Instead of evaluating each draft token individually, we compare all positions in a single `mx.eval(matches, draft_sampled)` call.

3.2 Batch Forward Pass Speedup

Table 2: Batch forward pass speedup vs. $N\times$ single-token forward. Measured after 20-token warmup with checkpoint/restore per iteration.

Batch	**Qwen3.5-9B-8bit (hybrid)**		**Qwen3-8B-4bit (pure Transformer)**	
Size	Time (ms)	Speedup	Time (ms)	Speedup
1	17.08	1.00×	9.96	1.00×
2	19.40	1.76×	14.32	1.39×
3	22.84	2.24×	18.73	1.60×
5	33.28	2.57×	27.87	1.79×
8	42.71	3.20×	41.31	1.93×

Counter-intuitive finding: The pure Transformer achieves *lower* batch speedup (1.79×) than the hybrid architecture (2.57×) for batch size 5. This is because:

1. The 4-bit Qwen3 model is smaller (∼5 GB vs. ∼10 GB), so single-token forward is already faster (10 ms vs. 17 ms), closer to the bandwidth ceiling.
2. Additional query tokens in a batch linearly increase KV cache attention bandwidth demand.
3. At 85% bandwidth utilization, there is almost no headroom for "free" batch computation.

For the hybrid architecture, we can decompose the forward pass into parallel (P) and sequential (R) components: $\text{Time}(N) \approx P + N \cdot R$. Fitting to measurements: $P = 13.03$ ms (76%), $R = 4.05$ ms (24%).

3.3 End-to-End Benchmark Results

Table 3: Speculative decoding benchmark results (ngram=3, draft=5, max_tokens=128/256).

Model	**Prompt**	**Std (t/s)**	**Spec (t/s)**	**Speedup**	**Accept%**
Qwen3.5-9B	short	67.9	67.8	1.00×	0%
	medium	65.3	64.9	0.99×	0%
	long	65.0	64.6	0.99×	50%
	summary	64.5	61.0	0.95×	52%
Qwen3-8B	short	114.3	110.4	0.97×	0%
	medium	114.1	104.1	0.91×	30%
	long	112.8	102.4	0.91×	30%
	summary	110.0	93.0	0.85×	55%

Speculative decoding is *slower* on the pure Transformer (0.85× worst case) than on the hybrid architecture (0.95× worst case), despite the hybrid having sequential GatedDeltaNet layers. The pure Transformer's higher baseline speed (114 vs. 65 t/s) means each speculative "miss" is more expensive in relative terms.

3.4 Cost Analysis

For batch(5) with 2 accepted tokens (typical scenario):

Table 4: Per-step cost breakdown for speculative verification.

Operation	**Qwen3.5 (ms)**	**Qwen3 (ms)**
Checkpoint	0.01	0.01
Batch forward (5 tok)	33.28	27.87
Sampling	0.52	0.50
Restore	0.04	0.04
Reprocess (2+1 tok)	18.42	∼13.0
Total	52.27	∼41.4
Tokens produced	3	3
ms/token (spec)	17.4	13.8
ms/token (std)	17.1	10.0

The speculative path produces 3 tokens in 52 ms (Qwen3.5) or 41 ms (Qwen3), yielding 17.4 or 13.8 ms/token—comparable to or worse than standard generation.

4 Prefill Performance Analysis

4.1 Throughput Comparison

Table 5: Prefill throughput at different prompt lengths.

Length	**Qwen3.5 (t/s)**	**Qwen3 (t/s)**	**Ratio**
64	528.8	571.7	1.08×
128	587.3	626.8	1.07×
256	642.9	703.8	1.09×
512	686.1	739.7	1.08×
1024	698.1	744.5	1.07×
2048	708.4	743.6	1.05×

Despite 75% of Qwen3.5's layers using sequential GatedDeltaNet, its prefill is only 5–9% slower than Qwen3's pure Transformer.

4.2 Layer-Level Time Decomposition

For Qwen3.5-9B at 512 tokens, we profiled each layer individually:

Table 6: Time decomposition by component (Qwen3.5-9B, 512 tokens).

Component	Time (ms)	% of Total
Linear projections (all layers)	∼430	**57.6%**
GatedDeltaNet recurrence kernel	∼216	29.0%
SDPA attention (8 full-attn layers)	∼50	6.7%
Other (embedding, norms, MLP gating)	∼50	6.7%
Total	∼747	100%

The dominant bottleneck is **quantized matrix multiplication in linear projections**, consuming 57.6% of prefill time. The attention kernel (6.7%) and GatedDeltaNet recurrence (29.0%) are secondary.

4.3 llama.cpp Performance Gap Analysis

The 1.9–3.6× prefill gap between MLX and llama.cpp is attributable to:

1. **Operator fusion** (∼40%): llama.cpp compiles a tighter compute graph with fewer intermediate buffers.
2. **In-kernel dequantization** (∼30%): llama.cpp fuses dequantization inside GEMM kernels, reducing memory round-trips.
3. **Block mask skipping** (∼15%): llama.cpp precomputes fully-masked KV blocks and skips them entirely; MLX evaluates masks per-element.
4. **Dispatch overhead** (∼15%): MLX's lazy evaluation model incurs higher per-operation scheduling cost for short sequences.

5 Correcting Prior Assumptions

Table 7: Five misconceptions from the prior optimization proposal, with corrections.

#	Prior Assumption	Actual Finding
1	MLX uses 256+ kernel dispatches for attention	MLX uses a single `dispatch_threadgroups` call, identical to llama.cpp
2	Metal shaders are JIT-compiled at runtime	MLX defaults to AOT compilation (`mlx.metallib`); JIT is an optional dev mode
3	SIMD matrix multiply is not utilized	MLX's `steel_attention` and `steel/gemm` already use `simdgroup_multiply_accumulate`
4	Pure-Transformer batch(5) should approach 5×	Actual speedup is only 1.79× due to memory bandwidth saturation
5	Speculative decoding should yield 1.5–2×	N-gram speculative decoding yields 0.85–1.00× (no improvement)

6 GatedDeltaNet Parallelization Analysis

6.1 Parallel Scan Feasibility

Equation 1 defines a **matrix-valued affine recurrence**. While scalar recurrences $s_t = a_t \cdot s_{t-1} + b_t$ can be parallelized via associative scan in $O(\log T)$ steps, the GatedDeltaNet recurrence requires $[D_k \times D_k]$ matrix multiplications at each scan step.

For Qwen3.5 with $D_k = 128$: each scan step involves 128×128 matrix multiplication ($O(d^3) = O(2 \times 10^6)$ FLOPs), and T steps require $O(T \cdot d^2)$ memory for intermediate states. At $T = 1024$ with 32 heads, this demands $\sim$32 GB—clearly impractical.

In contrast, Mamba/Mamba2's recurrence uses *diagonal* (element-wise scalar) decay, enabling standard parallel scan with $O(d)$ per step.

6.2 Chunkwise Parallel: A Viable Alternative

The DeltaNet [5] and GatedDeltaNet [4] papers propose a chunkwise parallel algorithm using the WY compact representation of Householder products:

1. Divide the sequence into chunks of size C (e.g., 64).
2. Within each chunk: use WY decomposition to parallelize the recurrence.
3. Between chunks: sequentially pass the final state.

Complexity: $O(L \cdot C \cdot d + L \cdot d^2)$ with $O(L/C)$ sequential steps—a significant improvement over $O(L)$ sequential for prefill. MLX already has a precedent: Mamba2's `ssm_attn` implements chunk-wise parallel SSD with chunk size 256.

Limitation: Chunkwise parallel does *not* help decode (where $T = 1$). This is a fundamental architectural constraint.

7 Revised Optimization Roadmap

Based on our findings, we propose three tiers of optimizations:

7.1 Immediate (Code Change $<$ 5 Lines)

1. **KV Cache quantization threshold**: Lower `DEFAULT_QUANTIZED_KV_START` from 5000 to 512. Expected memory reduction: 28% (8-bit) to 43% (4-bit) on KV cache. ArraysCache layers are automatically skipped. Code change: 1 line.
2. **Prefill chunk size**: Increase `prefill_step_size` from 2048 to 4096. Measured impact: +5–10% prefill throughput. Code change: 1 line.

7.2 Short-Term (1–2 Weeks)

1. **Fused QKV projections**: Merge separate Q, K, V linear projections into a single quantized matmul kernel. Since projections account for 57.6% of prefill time, reducing dispatch overhead and improving data reuse could yield 15–25% improvement.
2. **Block mask skipping in SDPA**: Add causal block skipping to the `steel_attention` kernel. For 512-token causal prefill, approximately half the KV blocks are fully masked and can be skipped entirely. Expected: 5–10% on attention-heavy workloads.

7.3 Medium-Term (2–4 Weeks)

1. **Chunkwise parallel GatedDeltaNet prefill**: Implement WY decomposition for chunkwise parallel processing of GatedDeltaNet layers during prefill. This is the highest-value optimization, potentially yielding 3–10× prefill speedup on sequences > 256 tokens. MLX's existing Mamba2 SSD implementation provides a code template.

7.4 Not Recommended

Table 8: Optimization directions evaluated and rejected.

Direction	Reason for Rejection
N-gram speculative decoding	Batch speedup insufficient (1.79–2.57× vs. 5× theory)
Unified attention kernel	MLX already uses single dispatch
Metal shader precompilation	MLX already uses AOT; only affects cold start
SIMD matrix multiply	Already used in steel_attention and steel/gemm
Continuous Batching	No benefit for single-user edge inference
Paged Attention	No memory fragmentation in single-user scenario
GDN decode parallelization	Mathematically infeasible (matrix-valued recurrence)

8 Discussion

8.1 The Memory Bandwidth Wall

Our central finding is that Apple Silicon LLM inference is **memory-bandwidth-bound**. MLX already achieves 85% of the M2 Ultra's 800 GB/s theoretical bandwidth during decode. This has profound implications:

- Batch forward passes cannot achieve near-theoretical speedup because each additional query token increases KV cache bandwidth demand.
- Speculative decoding's value proposition ("amortize weight-read cost over multiple tokens") breaks down when the bottleneck is reading KV cache, not model weights.
- Future optimizations must focus on reducing data movement (e.g., operator fusion, in-kernel dequantization) rather than increasing parallelism.

8.2 Architecture-Specific Insights

The Qwen3.5 hybrid architecture (75% GatedDeltaNet) presents unique challenges:

- GatedDeltaNet's recurrence is 24% of decode time but only 29% of prefill time—the per-layer cost (∼22.6 ms) is similar to full-attention layers (∼21.7 ms) because both are dominated by linear projections.
- The recurrence is *mathematically non-parallelizable* for decode ($T = 1$), setting a hard floor on decode latency that cannot be optimized away.
- Chunkwise parallel is viable only for prefill, where $T \gg 1$.

8.3 Limitations

1. Our speculative decoding evaluation uses N-gram prompt lookup only. A draft-model approach (e.g., Qwen3-0.5B) with higher acceptance rates might change the calculus, though the batch speedup limitation remains.

2. Profiling was conducted on M2 Ultra only. M3/M4 with NAX (Neural Architecture Extensions) may enable larger attention block sizes and different performance characteristics.

3. We did not implement the proposed optimizations beyond speculative decoding; feasibility assessments are based on code analysis and profiling data.

9 Conclusion

This study demonstrates that many commonly proposed LLM inference optimizations are either already implemented in MLX or fundamentally limited by memory bandwidth on Apple Silicon. The key lesson is that **empirical profiling must precede optimization**: our measurements overturned five assumptions from the prior proposal, potentially saving weeks of wasted engineering effort.

For practitioners deploying LLMs on Apple Silicon:

- MLX's token generation is already near-optimal (85% bandwidth utilization). Focus optimization effort on prefill.
- Speculative decoding with N-gram lookup is not beneficial. Consider draft-model approaches only if a suitable small model (<1B) is available with high acceptance rates.
- The most impactful near-term change is lowering the KV cache quantization threshold—a 1-line code change yielding 28% memory reduction.
- For hybrid GatedDeltaNet architectures, chunkwise parallel prefill is the highest-value research direction.

Acknowledgements

All experiments were conducted on a personal Apple M2 Ultra workstation. The MLX framework by Apple Machine Learning Research made this work possible. Profiling tools and speculative decoding implementation are open-sourced alongside this paper.

References

[1] A. Hannun et al., "MLX: Efficient and flexible machine learning on Apple silicon," 2023. https://github.com/ml-explore/mlx

[2] G. Gerganov, "llama.cpp," 2023. https://github.com/ggml-org/llama.cpp

[3] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from Transformers via speculative decoding," in *ICML*, 2023.

[4] J. Siems et al., "Gated Delta Networks: Improving Mamba2 with delta rule," in *ICLR*, 2025. arXiv:2412.06464.

[5] S. Yang et al., "Parallelizing linear Transformers with the delta rule over sequence length," in *NeurIPS*, 2024. arXiv:2406.06484.

[6] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," 2023. arXiv:2307.08691.

[7] Research_Apple_Silicon_LLM_Optimization.md, internal document, 2026.

[8] Flash Linear Attention Library, https://github.com/fla-org/flash-linear-attention

[9] T. Dao and A. Gu, "Transformers are SSMs: Generalized models and efficient algorithms with structured state space duality," in *ICML*, 2024.