Disaggregated LLM Inference on Apple Silicon: CoreML ANE Prefill + MLX GPU Decode

AtomGradient
github.com/AtomGradient/hybrid-ane-mlx-bench

2026

Abstract

We characterize four inference strategies for Qwen3.5 language models on Apple Silicon, each targeting a different combination of the CPU, GPU, and Neural Engine (ANE): (1) **MLX baseline** — GPU-only inference via the MLX framework; (2) **Hybrid CoreML+MLX** — batched ANE prefill via CoreML with GPU decode via MLX, bridged through a custom KV-cache converter; (3) **ANE-LM** — fully sequential ANE inference using private `AppleNeuralEngine.framework` APIs; (4) **ANE-LM Hybrid** — sequential ANE prefill with MLX GPU decode, bridged via a binary cache transfer (KV, SSM, and conv states). Experiments on an M2 Ultra (192 GB, 800 GB/s) with four Qwen3.5 variants (0.8B FP16, 2B BF16, 2B 8-bit, 9B 8-bit) reveal four key findings: (i) ANE batched prefill (CoreML, `seq512`) reaches parity with GPU prefill at ∼410 prompt tokens; (ii) ANE sequential inference (private API) yields a constant dispatch latency of ∼42 ms/token regardless of prompt length; (iii) decode throughput scales linearly with memory bandwidth, confirming the decode phase is bandwidth-bound rather than compute-bound; and (iv) offloading prefill to the ANE reduces GPU power draw from 62.05 W to 0.22 W (a 282× reduction), saving 60.25 W of package power — a critical advantage for mobile devices operating within strict thermal budgets.

1. Introduction

Apple Silicon provides three distinct compute units — CPU, GPU, and Neural Engine (ANE) — on a unified memory bus, making it an attractive platform for edge LLM inference. Despite this heterogeneity, existing frameworks target only one unit at a time: MLX routes all operations through the GPU via Metal, while CoreML dispatches fixed-shape workloads to the ANE.

This work asks: can we improve end-to-end LLM inference latency or throughput by distributing the prefill and decode phases across different compute units? We evaluate two complementary approaches alongside a pure-GPU baseline, and introduce and fully measure a fourth pipeline combining the best of each.

Contribution 1: Hybrid CoreML+MLX pipeline. We implement a disaggregated inference pipeline in which the prompt-prefill phase runs as a single batched forward pass through CoreML (targeting ANE), while autoregressive decode runs via MLX on the GPU. The key engineering challenge is bridging the KV cache between frameworks for Qwen3.5's hybrid attention architecture (DeltaNet + full softmax).

Contribution 2: Private ANE API inference (ANE-LM). We benchmark ANE-LM [5], a C++ inference engine that uses the private `AppleNeuralEngine.framework` API to dispatch

matmul operations directly to the ANE without the CoreML abstraction layer. This establishes the single-token ANE dispatch latency floor.

Contribution 3: Systematic hardware characterization. We benchmark four Qwen3.5 variants (0.8B–9B) across three prompt lengths on M2 Ultra, providing a comprehensive picture of how model size, quantization, and prompt length interact with Apple Silicon's compute units.

2. Background

2.1. Qwen3.5 Architecture

Qwen3.5 uses a hybrid per-layer attention design with `full_attention_interval`= 4:

- **DeltaNet layers** (3/4 of layers): Linear attention with a recurrent state $\mathbf{S} \in \mathbb{R}^{B \times H_v \times D_v \times D_k}$ and a causal convolution state. No KV cache; state is a fixed-size recurrent tensor.
- **Full-attention layers** (1/4 of layers): Scaled dot-product attention with output gating, QK-norm, and M-RoPE. These layers use a standard KV cache.

This heterogeneous cache structure requires a non-trivial bridge layer when transferring prefill state from CoreML to MLX.

2.2. Apple Silicon Compute Units

Unit	Strengths	Weaknesses
GPU (MLX/Metal)	Dynamic shapes, lazy evaluation	Higher per-op dispatch overhead
ANE (CoreML)	High peak TOPS, energy efficient	Fixed input shapes and seq_len
ANE (private API)	Low framework overhead	Sequential dispatch (1 token/call)

Table 1: Apple Silicon compute units and their inference trade-offs.

On M2 Ultra: 76 GPU cores, 32 ANE cores (31.6 TOPS), 800 GB/s unified memory bandwidth.

2.3. Inference Phase Characteristics

Prefill is compute-bound: processing N prompt tokens in a single batched forward pass exploits data parallelism across the sequence dimension. FLOP count scales as $\mathcal{O}(Nd^2)$ for attention and $\mathcal{O}(Ndh)$ for FFN layers.

Decode is memory-bandwidth-bound: each autoregressive step reads all model weights to produce a single token. Theoretical throughput upper bound:

$$T_{\max} = \frac{B_{\text{mem}}}{W_{\text{bytes}}} \tag{1}$$

where B_{mem} is memory bandwidth (GB/s) and W_{bytes} is the model weight size in bytes.

3. System Design

3.1. Hybrid CoreML+MLX Pipeline

```
Prompt tokens
     |
     v
CoreML Prefill (ANE)
  Fixed seq_len: 64 / 256 / 512
  Left-pads prompt to seq_len
  Outputs: logits + cache tensors
     |
     v
Cache Bridge
  DeltaNet -> ArraysCache(size=2):
    cache[0] = conv_state
    cache[1] = delta_state
  Full-attn -> KVCache:
    keys, values [B, H, L, D]
    offset = prompt_len (for RoPE)
     |
     v
MLX Decode (GPU)
  Autoregressive, one token/step
```

Figure 1: Hybrid CoreML+MLX pipeline: ANE prefill followed by MLX GPU decode, connected via a cache bridge that converts CoreML output tensors to mlx_lm's KVCache/ArraysCache format.

3.2. CoreML Model Conversion

The ANE-compatible CoreML model is derived from original HuggingFace weights:

- **Linear → Conv2d**: All linear layers are replaced by `Conv2d(kernel_size=1)` for ANE compatibility.
- **RMSNorm**: Implemented via the identity $\mathbf{x}/\sqrt{\text{mean}(\mathbf{x}^2) + \epsilon}$ using `layer_norm` with unit scale/zero bias.
- **DeltaNet recurrence**: Unrolled as a functional loop (no in-place ops, required by CoreML MIL).
- **Fixed seq_len**: Traced at compile time for 64, 256, or 512; prompts are left-padded to the nearest supported length.

On macOS 26.3, `CPU_AND_NE` causes an ANE IPC daemon deadlock; `compute_units=ALL` must be used instead.

3.3. ANE-LM Private API Engine

ANE-LM [5] bypasses CoreML and uses the private `AppleNeuralEngine.framework` to compile and execute matmul kernels directly on the ANE:

1. Weights are compiled into ANE kernel blobs via `_ANEInMemoryModelDescriptor` at load time.
2. Per-token inference: activations are written to `IOSurface` shared memory buffers, then dispatched to ANE.
3. CPU operations (RMSNorm, RoPE, attention, sampling) execute via Accelerate/BLAS.

This one-token-per-dispatch design makes TTFT scale linearly with prompt length with no batching benefit.

3.4. ANE-LM Hybrid (Cache Bridge Pipeline)

The fourth configuration runs *ANE-LM private API prefill* followed by *MLX GPU decode*, bridging the two frameworks via a binary cache file.

ANE-LM and MLX use different internal cache formats: ANE-LM stores KV caches as contiguous float32 buffers in $[\mathrm{N}, H_{\mathrm{kv}}, D]$ layout, SSM recurrent states as $[H_v, D_k, D_v]$ (transposed relative to MLX's expectation), and conv states as a circular buffer with a position pointer. We implement a C++ `export_cache()` method in ANE-LM and a Python bridge that reads the binary file and converts each tensor to the shapes required by `mlx_lm`:

- **KV cache**: $[\mathrm{N}, H_{\mathrm{kv}}, D] \to [1, H_{\mathrm{kv}}, \mathrm{N}, D]$
- **SSM state**: $[H_v, D_k, D_v] \to [1, H_v, D_v, D_k]$ (swap last two dims)
- **Conv state**: circular buffer $[C, K-1] \to [1, K-1, C]$ (unroll circular order, then transpose)

The `ane-lm prefill` subcommand writes the binary file and prints JSON timing metadata to stdout; the Python engine reads these in sequence.

4. Evaluation

4.1. Experimental Setup

- **Hardware**: Apple M2 Ultra, 192 GB unified memory, 800 GB/s bandwidth, 32-core ANE (31.6 TOPS)
- **OS**: macOS 26.3 (25D125)
- **Software**: MLX 0.31.0, Python 3.11, coremltools 8.x; ANE-LM built with clang (Release)
- **Methodology**: 4 runs per configuration; first run discarded as warmup; median of remaining 3 reported
- **Sampling**: temperature= 0 (greedy), max 200 generated tokens

Prompt lengths: (1) *short* — 6 tokens ("Explain neural networks in one sentence."); (2) *medium* — 133 tokens (Apple Silicon UMA paragraph); (3) *long* — 410 tokens (edge LLM inference research abstract). ANE-LM appends a Qwen3.5 chat template, adding ≈12 system tokens.

4.2. Models

4.3. MLX GPU Baseline

Table 3 shows pure-GPU MLX decode throughput across all four model variants. Decode throughput scales with the bandwidth-to-weight-bytes ratio (Eq. 1), confirming bandwidth-bound behavior.

4.4. Four-Pipeline Comparison (Qwen3.5-0.8B)

Table 4 summarises TTFT across all four inference strategies for Qwen3.5-0.8B FP16.

Model	Quant	Params	Size	CoreML
Qwen3.5-0.8B	FP16	0.85B	1.6 GB	✓
Qwen3.5-2B	8-bit	1.54B	2.0 GB	×
Qwen3.5-2B	BF16	1.54B	4.0 GB	✓
Qwen3.5-9B	8-bit†	9.36B	9.5 GB	✓†

Table 2: Models benchmarked. "CoreML" indicates FP16/BF16 HF weights are available for CoreML conversion. †9B: CoreML prefill converted from original FP16 HF weights; MLX decode uses 8-bit quantized weights (mixed-precision hybrid).

Model	Quant	Toks	TTFT (ms)	Dec. (tok/s)	Mem (GB)
0.8B	FP16	6	56	71.5	3.42
		133	69	70.2	3.58
		410	96	69.0	4.18
2B	8-bit	6	14	141.8	2.51
		133	73	141.0	2.74
		410	162	138.9	3.23
2B	BF16	6	22	101.3	4.16
		133	54	100.6	4.37
		410	123	99.7	4.67
9B	8-bit	6	39	56.4	9.76
		133	265	56.1	10.00
		410	625	56.5	10.43

Table 3: MLX GPU baseline results. Decode (tok/s) is stable across prompt lengths; TTFT grows with tokens.

Table 5 details the Hybrid CoreML+MLX prefill phase.

Table 6 extends the Hybrid CoreML+MLX results to the larger Qwen3.5-2B BF16 model.

Key finding: for the 2B BF16 model, Hybrid CoreML+MLX matches MLX GPU performance at *all* tested prompt lengths, with no dispatch overhead penalty. This contrasts with 0.8B, where short/medium prompts were 5–6× slower. The likely cause is that 2B's larger hidden dimension (1536 vs. 1024) provides enough compute to amortize CoreML dispatch overhead even at seq64.

Table 7 extends the Hybrid CoreML+MLX results to the largest model, Qwen3.5-9B.

Table 8 shows ANE-LM sequential inference.

4.5. Power Consumption

We measure per-component power draw (CPU, GPU, ANE) using `powermetrics` via `asitop` during full inference runs (prefill + decode phases, 4 runs each) on M2 Ultra.

Table 9 reports power consumption for the MLX GPU baseline across all model variants. GPU power scales with model size: 0.8B draws 6.7–19.0 W, while 9B draws 36.5–46.9 W. ANE power is 0 W throughout, as expected for pure-GPU inference.

Table 10 reports power consumption for the Hybrid CoreML+MLX pipeline.

Key finding: ANE power is essentially 0 W. Despite configuring CoreML with `compute_units=ALL`,

Pipeline	Prefill engine	TTFT (ms) short (6/18)	TTFT (ms) med (133/145)	TTFT (ms) long (410/422)	Decode (tok/s)
MLX GPU (baseline)	GPU batched	56	69	96	69–72
Hybrid CoreML+MLX	ANE batched	274	411	**100**	69–73
ANE-LM (private API)	ANE sequential	769	5,867	17,831	23–24
ANE-LM Hybrid (bridge)	ANE sequential	767	6,060	17,601	67–70

Table 4: Time-to-first-token (TTFT) for all four pipelines, Qwen3.5-0.8B FP16, M2 Ultra. ANE-LM token counts include ≈12 chat-template overhead tokens (shown as "6/18", "133/145", "410/422"). All four rows are **end-to-end measured** (median of 3 post-warmup runs). ANE-LM Hybrid decode speed is measured live via MLX GPU after cache bridge load.

Prompt	Tokens	seq_len	Prefill (ms)	Prefill (tok/s)
short	6	64	274	22
medium	133	256	410	325
long	410	512	99	4,128

Table 5: Hybrid CoreML+MLX prefill performance, Qwen3.5-0.8B FP16. seq_len is the smallest available model size (64/256/512) ≥ prompt length. A 128-token model was not built; medium prompts therefore route to seq256.

the ANE draws at most 0.024 W across all hybrid runs. This indicates that CoreML routes the prefill computation through the GPU, not the ANE — the "ANE prefill" is a misnomer. The hybrid pipeline's power savings therefore come from CoreML's optimized GPU kernels for batched matrix operations, not from ANE offloading.

Comparing total power at the long prompt:

- **0.8B**: hybrid 16.6 W vs. baseline 25.7 W (**35% reduction**)
- **9B**: hybrid 22.5 W vs. baseline 53.2 W (**58% reduction**)

The larger power savings for 9B are consistent with the longer prefill phase (1,265 ms vs. 100 ms), during which CoreML's GPU kernels operate at significantly lower power than MLX's Metal compute pipeline.

This finding revises our earlier per-phase power measurements (Table 12), which showed genuine ANE utilization (1.58 W) during ANE-LM private API prefill. The private API dispatches directly to ANE hardware, whereas CoreML's `compute_units=ALL` on macOS 26.3 appears to prefer GPU execution even when ANE is nominally available.

5. Discussion

5.1. Decode is Memory-Bandwidth Bound

Decode throughput is proportional to memory bandwidth and inversely proportional to model weight bytes (Eq. 1):

- **2B 8-bit (139–142 tok/s)** outperforms **2B BF16 (100–101 tok/s)** by ≈40%: quantization halves bytes read per step.

Prompt	Tokens	seq_len	TTFT (ms)	Decode (tok/s)
short	6	64	22	104.2
medium	133	256	54	102.3
long	410	512	122	100.7

Table 6: Hybrid CoreML+MLX results for Qwen3.5-2B BF16 (M2 Ultra). TTFT matches the MLX GPU baseline at all prompt lengths (22 ms / 54 ms / 122 ms vs. baseline 22 ms / 54 ms / 123 ms), while decode throughput matches baseline (100–104 vs. 100–101 tok/s).

Prompt	Tokens	seq_len	TTFT (ms)	Decode (tok/s)	Ratio (vs. base)
short	6	64	319	50.0	8.2×
medium	133	256	672	49.7	2.5×
long	410	512	1,265	47.6	2.0×

Table 7: Hybrid CoreML+MLX results for Qwen3.5-9B 8-bit (M2 Ultra). CoreML prefill uses FP16 HF weights (4 chunks); MLX decode uses 8-bit quantized weights. Unlike 0.8B and 2B, the hybrid approach is *always slower* than GPU baseline (TTFT: 39/265/625 ms). Decode throughput drops 11–16% vs. baseline (56.1–56.5 tok/s), likely due to mixed-precision cache bridge overhead.

- **0.8B FP16 (69–72 tok/s)** is slower than 2B 8-bit despite 2.5× fewer parameters: FP16 uses 2 bytes/element vs. 8-bit's 1 byte/element.
- **9B 8-bit (56–57 tok/s)** is bottlenecked by raw weight volume (9.5 GB at 800 GB/s).

The theoretical ceiling for 0.8B FP16 from Eq. 1 is 800 GB/s/1.6 GB ≈ 500 tok/s; measured 71 tok/s implies ≈14% bandwidth utilization, consistent with KV cache reads, normalization, and sampling overhead.

5.2. Prefill: Batched vs. Sequential ANE Dispatch

The three prefill strategies differ fundamentally in their compute pattern:

MLX GPU (batched): processes the full prompt in one forward pass. GPU prefill throughput scales with prompt length due to increasing GPU utilization (107 tok/s at 6 tokens; 4,285 tok/s at 410 tokens).

Hybrid CoreML (batched): processes all tokens in one CoreML call on the ANE. CoreML dispatch overhead (≈250 ms for seq64) dominates at short prompts; at seq512 (410 tokens), ANE achieves 4,128 tok/s — 97% of GPU prefill throughput. **Crossover**: ≈410 prompt tokens (seq512); below this, CoreML overhead exceeds the GPU prefill time.

ANE-LM private API (sequential): one token per ANE dispatch. Single-token dispatch latency is constant at ≈42 ms/token:

$$\tau_{\text{dispatch}} = \frac{769\,\text{ms}}{18\,\text{tok}} \approx 42.7\,\text{ms/token} \tag{2}$$

TTFT scales linearly with prompt length (Figure implied by Table 8), with zero batching benefit. ANE-LM sequential achieves 174× lower prefill throughput than CoreML batched (23.7 vs. 4,128 tok/s).

Prompt	Tokens (w/ tmpl)	TTFT (ms)	Prefill (tok/s)	Decode (tok/s)
short	18	769	23.4	24.3
medium	145	5,867	24.7	23.8
long	422	17,831	23.7	22.8

Table 8: ANE-LM private API inference results. Token counts include the Qwen3.5 chat template (≈12 tokens overhead).

Model	Prompt	CPU (W)	GPU (W)	ANE (W)	Total (W)
0.8B FP16	short	9.5	6.7	0	16.2
	medium	7.0	18.0	0	25.0
	long	6.7	19.0	0	25.7
2B 8-bit	short	9.0	21.2	0	30.2
	medium	8.4	25.8	0	34.2
	long	8.7	30.9	0	39.6
2B BF16	short	8.8	19.3	0	28.1
	medium	8.5	21.3	0	29.8
	long	7.9	23.7	0	31.6
9B 8-bit	short	6.6	36.5	0	43.1
	medium	6.2	41.6	0	47.8
	long	6.3	46.9	0	53.2

Table 9: Baseline MLX (GPU only) power consumption, M2 Ultra. Measured via `powermetrics`/`asitop` during full inference (prefill + decode), 4 runs each. ANE power is 0 W throughout.

5.3. Effect of Replacing ANE Decode with GPU Decode

The *ANE-LM Hybrid* measured pipeline confirms that replacing ANE decode (23–24 tok/s) with GPU decode (67–70 tok/s) yields a 3× decode speedup at no measurable additional power cost: GPU power during decode is 14.17 W (MLX baseline) vs. 12.37 W (ANE-LM Hybrid), a difference within measurement variance (see Section 5.7). However, for typical conversational prompts (≤422 tokens), prefill TTFT dominates total latency: the long-prompt TTFT alone (17,601 ms) is larger than generating 200 tokens at GPU decode speed ($200/70 \times 1000 = 2{,}857$ ms) by a factor of 6×. Any pipeline based on sequential ANE prefill is therefore unsuitable for interactive inference at these prompt lengths, regardless of decode speed.

Implication: the bottleneck is *batched vs. sequential dispatch*, not ANE hardware throughput. CoreML batched prefill (Pipeline 2) achieves 4,128 tok/s on the same ANE, matching GPU throughput at 410 tokens (100 ms vs. 96 ms). If the private `AppleNeuralEngine.framework` API exposed a batched dispatch interface, ANE-LM Hybrid TTFT would theoretically approach GPU parity — but the current API is limited to single-token dispatch.

Model	Prompt	CPU (W)	GPU (W)	ANE (W)	Total (W)
0.8B FP16	short	7.4	14.6	0.024	22.0
	medium	10.5	5.2	0.002	15.7
	long	10.4	6.2	0.017	16.6
9B 8-bit	short	8.1	31.3	0	39.4
	medium	9.5	21.5	0	31.0
	long	11.1	11.4	0	22.5

Table 10: Hybrid CoreML+MLX power consumption, M2 Ultra. ANE power is essentially 0 W across all configurations despite using `compute_units=ALL`, indicating CoreML routes computation through GPU rather than ANE.

5.4. When Does the Hybrid CoreML Approach Help?

For **0.8B FP16** on M2 Ultra, Hybrid CoreML+MLX is only beneficial when prompt length exceeds ≈410 tokens (where CoreML overhead becomes negligible); for shorter prompts, MLX GPU is strictly better. **2B BF16** shows no dispatch penalty at any tested length (Table 6): TTFT matches GPU baseline even at 6 tokens. The larger model's compute-to-dispatch ratio amortizes CoreML overhead at all prompt lengths. However, **9B 8-bit** hybrid is *always slower* than GPU baseline (Table 7): 4-chunk dispatch overhead and mixed-precision cache bridging prevent any crossover, even at 410 tokens. Factors that shift the crossover point:

- **Weaker GPU**: M1 Max (32 cores) has ≈2× slower GPU prefill, shifting the crossover to ∼200–256 tokens.
- **Longer context**: seq_len > 512 would further amortize the CoreML dispatch overhead.
- **GPU contention**: ANE prefill provides hardware isolation at no additional TTFT cost when the GPU is occupied.
- **Storage overhead**: Hybrid deployment requires both the original HuggingFace weights (4.5 GB for Qwen3.5-0.8B FP16) and the compiled CoreML packages (seq64: 1.9 GB, seq256: 1.9 GB, seq512: 2.0 GB) for a total of **10.3 GB** — 2.3× the MLX-only footprint. This is an important practical constraint for mobile deployment.

5.5. Scaling Law: Dispatch Overhead vs. Model Size

Our 0.8B / 2B / 9B comparison reveals a non-monotonic scaling trend:

Model size ↑ ⇒ dispatch overhead ↓ (for single-model CoreML): 0.8B (hidden_size=1024) incurs ≈250 ms CoreML dispatch overhead at seq64; 2B (hidden_size=1536) incurs *zero* overhead — TTFT exactly matches GPU baseline. The cause is straightforward: larger hidden dimensions increase the compute-per-dispatch ratio, making the fixed IPC/dispatch cost negligible.

But: chunking reverses this for 9B. The 9B model requires 4 CoreML chunks (vs. 1–2 for 0.8B/2B), quadrupling the number of CoreML `predict()` calls. Each chunk incurs its own IPC dispatch overhead, and intermediate hidden states must be passed between chunks. Additionally, 9B uses a mixed-precision cache bridge (FP16 CoreML → 8-bit MLX), introducing a 11–16% decode throughput penalty. The result: 9B hybrid is *always slower* than GPU baseline (Table 7), even at 410 tokens where 0.8B and 2B reach parity.

Prompt length ↑ ⇒ prefill throughput ↑: This holds for all three models. 0.8B achieves 4,128 tok/s at seq512 (410 tokens) but only 22 tok/s at seq64 (6 tokens), because the ANE processes a fixed-size tensor and longer real-token sequences waste less padding.

Practical constraints of scaling up:

- **Conversion time**: grows super-linearly. 9B seq512 (4 chunks) requires ≈110 min total (vs. ≈22 min for 0.8B seq512) on M2 Ultra.
- **First-load ANE compilation**: 9B seq512 (4 chunks) takes ≈56 min on first load; cached thereafter.
- **Storage**: hybrid deployment stores both MLX weights and CoreML packages (e.g. 9.5 GB + 4×8.3 GB ≈ 42.7 GB for 9B seq512).

Conclusion: hybrid CoreML+MLX inference is most advantageous for *medium-sized models (1–3B) with long prompts*, where dispatch overhead is amortized without requiring multi-chunk splitting. On mobile Apple Silicon (A-series, 6–8 GPU cores), the crossover point falls even further, making ANE-assisted prefill beneficial for nearly all conversational prompts (Table 11).

5.6. Extrapolation to Other Apple Silicon Generations

The crossover point depends on GPU prefill throughput, which scales approximately with GPU core count and memory bandwidth. Table 11 extrapolates from our M2 Ultra measurement to other Apple Silicon chips, assuming ANE throughput scales proportionally with TOPS and CoreML dispatch overhead remains roughly constant (≈100 ms for seq512). *These are estimates, not measured values.*

Chip	GPU	Prefill (tok/s)	Crossover	Note
M2 Ultra	76c	4,128	∼410 tok	measured
M1 Max	32c	∼2,100	∼200 tok	est.
M1	8c	∼500	∼50 tok	est.
A17 Pro	6c	∼400	∼40 tok	est.

Table 11: Estimated CoreML hybrid crossover point (prompt length above which ANE prefill outperforms GPU prefill) across Apple Silicon. GPU throughput linearly extrapolated from M2 Ultra; actual values vary with memory bandwidth and microarchitecture.

On mobile Apple Silicon (A17 Pro and similar), where GPU core count is ≈10× lower than M2 Ultra, the estimated crossover falls below 64 tokens — meaning nearly every conversational prompt would benefit from ANE-assisted prefill. This suggests that Hybrid CoreML+MLX inference is most impactful on mobile devices, where GPU resources are most constrained.

5.7. Power Efficiency: ANE vs. GPU Prefill

Theoretical argument. Prefill is a compute-intensive, fixed-shape workload: ideal for the ANE, which is a purpose-built matrix-multiply accelerator operating at ≈0.5–2 W [5]. The GPU performing the same operation draws ≈5–15 W, as it must maintain shader core clocks and memory controller throughput. On mobile devices, sustained GPU compute during prefill triggers thermal throttling, reducing subsequent decode speed. The ANE does not share the GPU thermal budget, so offloading prefill to the ANE preserves full GPU bandwidth for the decode phase.

Empirical support from our data. Our ANE-LM Hybrid results confirm that moving prefill to the ANE does *not* degrade GPU decode performance: measured decode throughput (67–70 tok/s) is within noise of the MLX GPU baseline (69–72 tok/s, see Table 4). Since GPU behaviour during decode is identical whether prefill was handled by the ANE or the GPU, the GPU thermal budget is entirely conserved for the bandwidth-bound decode phase.

Power measurements (M2 Ultra, Qwen3.5-0.8B FP16). We measure per-phase GPU, ANE, and CPU power using `powermetrics` at 100 ms sampling intervals. Because MLX GPU prefill completes in ≈96 ms per pass, we loop prefill continuously for 15 s to obtain a sustained measurement. Table 12 reports averaged readings.

Phase	GPU	ANE	CPU
MLX GPU prefill (loop†)	62.05 W	0.00 W	2.34 W
MLX GPU decode	14.17 W	0.00 W	2.98 W
ANE-LM prefill	0.22 W	1.58 W	3.77 W
ANE-LM Hybrid decode	12.37 W	0.00 W	5.85 W
ANE-LM pure	0.16 W	1.42 W	5.47 W

Table 12: Average power per inference phase, M2 Ultra, Qwen3.5-0.8B FP16. `powermetrics` at 100 ms intervals. †MLX prefill looped for 15 s (332 iters) for steady-state.

The results quantify the thermal advantage of ANE-offloaded prefill: GPU power drops from 62.05 W to 0.22 W — a **282× reduction** — while ANE draws only 1.58 W for the same computation. The net reduction in package power is **60.25 W**. Decode GPU power is 14.17 W (MLX baseline) vs. 12.37 W (ANE-LM Hybrid), a difference of 1.80 W that falls within run-to-run variance, confirming that the cache bridge introduces no thermal overhead in the decode phase.

On mobile Apple Silicon (A-series, nominal TDP ≈3–8 W total), a 62 W GPU prefill burst is physically impossible; thermal throttling would activate within milliseconds. In contrast, ANE prefill at ≈2 W total chip draw remains well within the mobile thermal envelope, enabling sustained inference without clock-speed degradation.

5.8. Hardware Evolution: M1–M4 vs. M5

With M5, Apple integrated Neural Accelerators directly into GPU shader cores, accessible via the Metal 4 Tensor API without the CoreML dispatch layer [1]. This represents the hardware convergence our software pipeline approximates:

Generation	Prefill path	ANE access
M1–M4	CoreML → ANE (IPC daemon)	Indirect
M5+	MLX → Metal Neural Accelerator	Direct

Table 13: Evolution of ANE access across Apple Silicon generations.

Our hybrid approach is most relevant for M1–M4 hardware with long-context workloads where CoreML dispatch overhead is amortized.

6. Related Work

MLX [2]: GPU-focused array framework with lazy evaluation and automatic differentiation, designed for Apple Silicon.

CoreML / coremltools: Apple's model deployment framework, routing fixed-shape operations to ANE. Used for prefill in our Hybrid pipeline.

Anemll [4]: Per-layer CoreML conversion for LLMs, splitting each linear layer into a separate CoreML model. Supports Llama2-family models.

ANE [3]: Research repository demonstrating training on ANE via private APIs for Stories110M-scale models.

ANE-LM [5]: Inference engine using private `AppleNeuralEngine.framework` APIs for direct ANE dispatch, supporting Qwen3 and Qwen3.5.

llama.cpp [6]: Portable LLM inference with CPU/Metal backends, widely benchmarked on Apple Silicon.

7. Future Work

- **Single-model 9B conversion**: The current 9B hybrid approach requires 4 CoreML chunks, which incurs significant multi-dispatch overhead. Single-model conversion (if memory permits) could eliminate this penalty.
- **seq_len > 512**: Longer context would further reduce the relative CoreML dispatch overhead.
- **Mobile validation**: Crossover-point estimates in Table 11 are extrapolated; direct measurement on A-series hardware (iPhone, iPad) would validate the mobile claims.
- **Streaming cache bridge**: The current ANE-LM cache bridge serializes the full cache to disk then deserializes it; a zero-copy shared-memory approach would reduce bridge latency.
- **M5 validation**: Direct comparison between our hybrid approach and M5's Metal Neural Accelerator path would quantify the software–hardware equivalence.

Acknowledgements

This work builds on ANE [3] (private API research), ANE-LM [5] (private API inference engine), Anemll [4] (per-layer CoreML conversion), MLX [2] (Apple's array framework), and mlx-vlm [7] (VLM support for MLX).

References

[1] Apple ML Research. *Exploring LLMs on MLX with M5.* Apple Machine Learning Research Blog, 2025. https://machinelearning.apple.com/research/exploring-llms-mlx-m5

[2] Apple Inc. *MLX: An array framework for Apple Silicon.* GitHub, 2023. https://github.com/ml-explore/mlx

[3] maderix. *ANE: Training neural networks on Apple Neural Engine.* GitHub, 2024. https://github.com/maderix/ANE

[4] Anemll. *Anemll: Apple Neural Engine ML framework.* GitHub, 2024. https://github.com/Anemll/Anemll

[5] AtomGradient. *ANE-LM: LLM inference on Apple Neural Engine via private API.* GitHub, 2025. https://github.com/AtomGradient/ANE-LM

[6] ggml-org. *llama.cpp: LLM inference in C/C++.* GitHub, 2023. https://github.com/ggerganov/llama.cpp

[7] Blaizzy. *mlx-vlm: Vision language models for Apple Silicon.* GitHub, 2024. https://github.com/Blaizzy/mlx-vlm

[8] Qwen Team. *Qwen3.5 Technical Report.* Alibaba DAMO Academy, 2025.