A Predictive, Per-Engine Energy Roofline of the Apple Neural Engine

SPENCER BRYNGELSON, The University of Texas at Austin, Austin, United States

The Apple Neural Engine (ANE) is in over a billion devices, but is characterized indirectly via CoreML. Using a CoreML-free fused-graph dispatch path, we measure the ANE, along with the Apple GPU and CPU on the same device, with per-engine energy accounting. The consumer-available roofline does not predict the ANE in practice. A critical correction is that the ANE sees two effective memory bandwidths: a slow activation path and a fast weight-stream path. Two roofline ridges follow, and the weight-stream ridge is below that of the GPU. The ANE single-operand compute peak decreases instead of plateauing. Its weight stream is byte-bound in fp16 but limited by the dequantizer's element rate in int8. As a result, int8 compression saves energy at unchanged throughput and an int4 lookup-table stream raises throughput and efficiency at a cost of 2.55 points of ImageNet top-1. The corrected roofline identifies the lower-energy ANE paths and bounds it from below. A constructed device map reproduces across four unique ANE generations. Results show that operators should be placed by arithmetic intensity and fused for optimal performance.

CCS Concepts: • **Hardware** → **Power estimation and optimization**; • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → *Neural networks.*

Additional Key Words and Phrases: Apple Neural Engine, roofline model, performance characterization, energy efficiency, hardware accelerators, on-device inference

1 Introduction

The Apple Neural Engine (ANE) is among the most widely deployed machine-learning accelerators, present in every recent iPhone, iPad, and Apple-silicon Mac. Despite this ubiquity, the engine's performance and energy profile has had little open measurement. The publicly quoted figure is a peak 38 TOPS (tera-operations per second) for the M4 Neural Engine. Community guidance, discussed in a community knowledge base [13] and implicit in Apple's transformers-on-the-engine material [1], states that the engine is worthwhile mainly for inference on large transformer models. Both the principal figure and the large-model guidance come from CoreML, which is the only sanctioned interface to the ANE. However, these figures instrument the hardware indirectly. CoreML decides, by an opaque placement policy, which operators run on the ANE versus the GPU or CPU, and it exposes neither a fair per-operator dispatch nor energy accounting [6]. The community's working assumptions about the ANE thus reflect a scheduler's defaults and optimization schemes, not the hardware envelope.

This work measures which workloads favor the ANE over the Apple GPU and CPU for the same device on speed and watts. No CoreML-free method exists to dispatch arbitrary operators with controlled batch, dtype, and warm state. We provide one with `aneforge` [5], a fused-graph compiler and runtime that lowers a user's operator graph to a single fused ANE program and submits it through Apple's `e5rt` execution runtime. Every timed operator runs on the ANE with controlled batch, data type, and warm state. This gives a fair per-operator dispatch and a controlled device comparison. The access mechanism is documented elsewhere [5, 6], though the instrument is described in full in the online supplement so the measurements can be independently judged. This includes two checks that every benchmarked operator runs on the ANE and not a fallback device. Against the ANE we run MLX, Apple's Metal-based array framework (fp16), for the GPU and numpy/Accelerate (fp32, via

Authors' Contact Information: Spencer Bryngelson, The University of Texas at Austin, Austin, Texas, United States; e-mail: shb@utexas.edu.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 1544-3973/2026/11-ART
https://doi.org/10.1145/3864456

Just Accepted

Apple's AMX matrix coprocessor) for the CPU, on an Apple M5 Pro (ANE architecture h17s; a 48 GB MacBook Pro, 18-core CPU and 20-core GPU). The setup repeats the three-way roofline comparison, neural processing unit (NPU, or ANE here) versus GPU versus CPU, that was established for a datacenter TPU [18], for an on-device NPU with energy measured throughout. The fp32 Accelerate/AMX CPU path is a contextual reference, not a like-for-like fp16 comparand. The main comparison is therefore the ANE against its companion GPU.

Prior characterizations reach the engine through CoreML or target other accelerators: smartphone-AI benchmarking [15, 25], the per-engine Edge TPU evaluation [26], edge and mobile NPU energy studies [17, 20, 23, 28, 31], and a small body of direct-access ANE work [19, 27, 33]. These works do not isolate the ANE in a per-engine energy roofline across operating regimes (see Section 11 for further discussion).

Here, we present a corrected roofline. The textbook single-roofline model mispredicts the ANE. The primary correction is that the ANE has two effective bandwidths: a slow activation path and a fast weight-stream path, so two ridges, not one. The weight-stream ridge that governs all compute-bound work lies below the GPU's, so an operator becomes compute-bound on the ANE at a lower arithmetic intensity than on the GPU. Its weight stream is byte-bound in fp16 but limited by the dequantizer's element rate in int8, so what weight compression buys depends on the encoding and on the part. Its single-fused-operand compute roof for general matrix–matrix multiplication (GEMM) falls instead of plateauing, so fusion, not a larger operand, keeps work on the fast tier. We combine these three corrections into a per-engine energy roofline. From a workload's arithmetic intensity it assigns each operator to the lower-energy engine and bounds its energy per operation, validated on held-out archetypes and a decode loop on the measured host (Section 5). The regime-dependent device map that prior NPU-versus-GPU work reports follows from this two-ridge structure. We test whether that map is a property of one part or of the engine family, and separate what reproduces across generations from what is specific to the measured host (Section 7).

The measurement method ties idle-subtracted total-package energy to the roofline per engine (Section 2). From it follow the three corrections (Sections 3.1, 3.3 and 9), a closed-form energy predictor and placement rule fit from two per-device constants (Section 5), and the device map's reproduction across four ANE generations (Sections 6 and 7).

Just Accepted

2 Methodology

Measurement tool and dtype. The ANE is driven by `aneforge`'s fused-graph dispatch (fp16): each operator graph is lowered to one fused MIL program and submitted directly to the engine, with no CoreML model and therefore no CoreML placement policy. Every timed operator is verified to run on the ANE rail. The per-rail power attribution below confirms this, and a route-and-capability gate test compiles both lowerings of every selectable operator on-device and validates each against a reference. The full instrument description is in the supplemental material. The GPU is driven by MLX v0.31.2 (Metal, fp16; v0.32.2 for the compiled MiniLM and ViT-B/16 baselines). The real-model rows also use torch-MPS (torch's Metal Performance Shaders backend) in fp16, with a single-session fp32 energy column. The CPU is driven by numpy/Accelerate (fp32, AMX), with torch as the reference where it is the canonical implementation. The GPU and ANE therefore run the same fp16 math; the CPU runs fp32 because Accelerate/AMX has no fast fp16 GEMM (it upcasts), which doubles its bytes-per-element and halves its arithmetic intensity, carried through the roofline arithmetic wherever it matters.

Speed. The measured latency is the minimum end-to-end latency over repetitions after warmup, a dedicated probe separate from the sustained power window; at the smallest shapes it measures each device's dispatch floor. Each repetition forces device completion (`mx.eval` for MLX, the compiled-net call for the ANE, `torch.mps.synchronize` for MPS) so that no work is left in flight. All measurements are at warm caches and steady state, with matched batch sizes. Throughput is GFLOP/s where a closed-form FLOP count exists, otherwise

elements/s. The latency includes each device's host and dispatch overhead; at sub-millisecond shapes the overhead is part of the cost a deployment pays and is not subtracted out.

Power. Power is read from `powermetrics` per ANE, GPU, or CPU rail. The state of programmatic Apple-silicon power measurement (IOReport versus `powermetrics`, and the absence of a validated estimator) is the subject of [22]; prior Apple-silicon efficiency studies have read `powermetrics` per-rail across the M1 to M4 generations [14]. We adopt total-package active power herein.

Following [11], a no-workload baseline is sampled once per run and subtracted (package 1196 mW in the device-map run). We take `powermetrics`' own `Combined Power (CPU + GPU + ANE)` line, not a hand-summed rail total, because an ANE workload still draws from host CPU power on dispatch (as does an MLX workload).

The measurement boundary. By *per-engine energy accounting* we mean that every operating point reports total-package power, latency, and accuracy for the engine that ran it. The claim is one of attribution, not of an exhaustive power domain. The domain is the CPU, GPU, and ANE rails as `powermetrics` reports them. It excludes the DRAM rail, quantified next, and rests on a modeled SoC estimator, not a wall-socket meter. For each measurement a per-rail breakdown is reported. Median loaded package power with the coefficient of variation (CV%) over per-sample readings are included, with CV above 35 % flagged low-confidence measurement. Each loop drives the workload for the entire powermetrics window and samples at 500 ms to integrate the ANE's sub-millisecond duty cycle; a naive fixed-time stop integrating at 100 ms produces a 60–140 % CV quantity, whereas this strategy gives 1–25 % CV (30 of 31 measurements below 17 %).

Cross-validation against an independent estimator. The total-package active-power metric is cross-validated against Apple's IOReport energy-model counters at five operating points spanning the attribution axis (see supplement, Table S1), where the estimators agree to a median of 97–99 %. Both remain modeled SoC estimators, not wall-socket meters; the relative device comparisons are unaffected by a shared offset that cancels via ratios.

The DRAM rail. The combined power excludes the DRAM rail, which the IOReport DRAM channel puts at 2.4–12.9 % of package across the loaded operating points. That share is device-dependent, not uniform: 4.1–5.3 % on an ANE convolution against 12.5–12.9 % on a saturating GPU GEMM. Including DRAM takes the GPU-over-ANE package-power ratio on that pair from 8.4 to 9.0. Since the GPU's DRAM share is the largest measured, excluding the rail is the conservative choice for the ANE-versus-GPU comparisons this paper draws (online supplement, Table S28).

Derived metrics: single-window. Perf/watt and mJ/inference (active W × sustained ms/iter) are computed within a single sustained measurement, never spliced across executions. The sustained per-iteration time is the wall time of the measurement loop divided by the inferences it completed inside the same window over which power is summarized. Each 6 s window holds 354 to 61 446 inferences, so the reported energy integrates completed work within one execution, and does not combine statistics from separate runs. At sub-millisecond timings that time exceeds the separate minimum-latency probe on every ANE path, and perf/watt therefore cannot be reconstructed from the latency column, by design. Across Table 5 the two statistics differ by a median factor of 1.05, and by more than 25 % only at the smallest shapes, where the gap measures per-call dispatch overhead. The online supplement (Table S21) reports both per class, with the sensitivity of each energy figure to the choice of power and latency statistic; both choices are conservative for the ANE.

Repeated measurements. Some results are replicated over independent cold starts, each a fresh process with its own idle baseline and randomized device order, and carry a 95 % confidence interval. These are the ANE and fp16 torch-MPS energies and ratios in Table 6, the six device-map classes marked in Table 5, and the ceiling and serving comparisons and crossovers of Table S17 in the online supplement. The compiled-MLX columns of Table 6 are repetitions within one session, also with 95 % intervals. The other seven device-map classes, the fp32 GPU energy

Just Accepted

column, the serving sweeps apart from their crossovers, the decode sweeps, and the cross-generation maps are each a single sustained measurement with a within-window CV and no across-run interval. Under repetition the DFT ratio moved from its single-session 0.82× to 1.35×, with an interval spanning parity (Section 6).

Accuracy. Every workload reports relative L2 error against an fp32/fp64 numpy reference, printed beside the speed and watt numbers so that a speed gain at worse accuracy is visible. The GPU is consistently more accurate in fp16 than the ANE, most so on large GEMMs. This error is computed on a synthetic input and is not a uniform proxy for real data: on their own domains ResNet-18 falls to 1.6×10^{-3}, MiniLM holds at 5.7×10^{-3}, and ViT-B/16 rises to 2.6×10^{-2}. Compiling the ViT encoder to each depth localizes that gap. Error grows by 7–9 % per block for both inputs, and their difference appears only at the classification head, where a peaked logit vector carries larger absolute rounding over a smaller norm. Relative L2 also does not say whether a divergence changes an answer, so all three real models are scored on their task. ResNet-18 reaches 69.76 % top-1 on the ANE, the GPU, and the fp32 CPU reference alike over all 50 000 ILSVRC2012 validation images. ViT-B/16 reaches 81.00 % against the reference's 81.06 %. Both reproduce the published torchvision accuracies. Neither ANE path shifts accuracy: its label changes run in both directions (ResNet 14 against 14, ViT 33 against 21, McNemar $p = 1.00$ and 0.13), though on ViT it changes ten times as many individual labels as the GPU does. MiniLM has no label set, so the decision measured there is which neighbor an embedding retrieves: the ANE returns the fp32 nearest neighbor for 99.2 % of queries and reproduces the pairwise similarity ranking exactly to four decimals. Weight compression produces the measurable task-level losses: per-channel int8 costs 0.11 percentage points of ResNet-18 top-1 and two points of MiniLM retrieval agreement, and int4 costs 2.55 points (online supplement, Tables S25 and S26).

Exclusions. Sixteen subprocess-bridge operators (sort, cost-volume, farthest-point sampling, radius search, and others) run at 25 ms to 2.5 s because each call spawns a one-shot probe process and pays the full process-spawn, graph-author, and ANE-load cost every call, a dispatch artifact, not silicon speed. Timing them against MLX would misrepresent the hardware, so they are excluded from the speed comparison and treated as a capability statement instead. The sixteen are listed in the online supplement, and the three bridge operators that carry a persistent worker (`sdpa`, `argmax`, `topk`) are timed on silicon and remain in the comparison.

3 Compute and bandwidth ceilings

The corrected roofline uses measured ceilings for each device. A representative shape need not saturate the device: a 512×512 GEMM reaches only about 4 % of the GPU's fp16 peak and therefore understates its capability.

3.1 Compute roof

The compute-saturation sweep scales two compute-bound primitives (square GEMM, $2N^3$ FLOPs, and a 3×3 same-pad convolution, the ANE's highest-throughput primitive) until each device plateaus, measuring throughput, idle-subtracted active package power, and fp16 relative error at every size. Table 1 gives the peaks.

Table 1. Compute and efficiency peaks (M5 Pro), GPU and ANE in fp16.

Primitive	Device	Dtype	Peak GFLOP/s	@ size	relerr there	Peak GFLOP/s/W
GEMM	**GPU**	fp16	**30 862**	$N = 6144$	3.6e-4	603 ($N = 8192$)
GEMM	**ANE**	fp16	10 191	$N = 2048$	1.2e-2	**866** ($N = 4096$)
conv	**GPU**	fp16	**31 790**	C=512	7.4e-3	1 392 (C=512)
conv	**ANE**	fp16	18 771	C=512	5.5e-4	1 372 (C=512)

The GPU has the higher raw throughput on both primitives compared to the other devices; GEMM 3.0× the ANE (30.9 versus 10.2 TFLOP/s) and conv 1.7× the ANE (31.8 versus 18.8 TFLOP/s), confirming that the large

Just Accepted

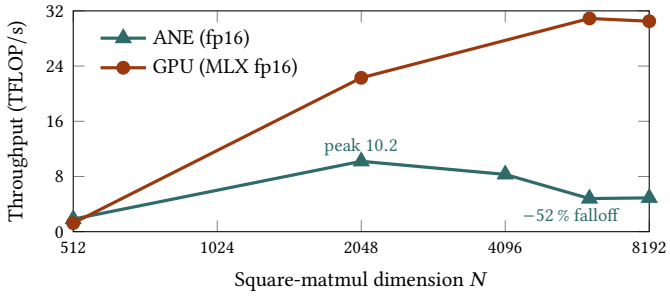

Fig. 1. Square-matmul throughput versus dimension N on the M5 Pro (ANE fp16, GPU MLX fp16). The ANE roof peaks then falls; the GPU's rises and plateaus.

square GEMM of large-language-model (LLM) inference is appropriate for the same SoC's GPU. Perf/watt divides between the primitives. At $N = 4096$ the ANE is 3.6× slower but draws 9.5 W against the GPU's 53.3 W, so its efficiency at that size is 1.55× the GPU's (866 against 558 GFLOP/s/W). Peak against peak, the ANE's efficiency is 1.4× higher (866 at $N = 4096$ against the GPU's 603 at $N = 8192$). Cold-start replication of the device map's GEMM-compute shape ($M = 256$, $K = N = 4096$) gives 1.45× (811 against 558 GFLOP/s/W, online supplement, Table S17); this is the most variable contested claim, but it remains Holm-significant. For conv the single-sweep gap is small (ANE 1372 against GPU 1392 GFLOP/s/W) and replication resolves it to a 5 % GPU edge. This bounds the efficiency claim: it holds for GEMM at peak and reverses to a small GPU edge at conv saturation. The utilization shows in the power curves: the GPU's rises from 9 W at $N = 512$ and plateaus at 49–53 W for $N \geq 4096$, while the ANE's peaks at 12.7 W at $N = 2048$ and then falls with its throughput.

The second roofline correction is determined via the throughput curve: the ANE's compute roof falls instead of plateauing. ANE GEMM peaks at $N = 2048$ (10.2 TFLOP/s), then degrades to 8.3 TFLOP/s at $N = 4096$, 4.8 for $N = 6144$, and 4.9 for $N = 8192$, a 52 % drop that reproduces across runs (Figure 1), remaining at roughly half its peak for $N \geq 6144$. The active power falls with the throughput (12.7 W to 6 W), so the engine is stalled, not busy at a reduced rate. Once a square operand exceeds the ANE's working-set window (about $N = 2048$ to 4096), on-chip tiling and weight streaming no longer feed the multiply–accumulate (MAC) array fully. The computation then serializes on off-chip streaming. The same peak-then-decline shape appears on an M2 Pro (h14), peaking at 7.5 TFLOP/s at $N = 2048$ and falling to 2.2 at $N = 8192$, so it is a property of the engine, not of one part. Accuracy degrades over the same range: relative error grows from 2.6×10^{-3} at $N = 512$ to 6.2×10^{-2} at $N = 8192$. Large-N ANE GEMM therefore loses both throughput and accuracy. That growth is consistent with per-product fp16 rounding over a long contraction, not a narrow accumulator: the MAC array carries a wide, fp32-class accumulator, and the error tracks $\sqrt{K}$ as independent rounding errors accumulate.

The falloff is specific to the large square single-matmul shape: ANE conv throughput rises monotonically to its 18.8 TFLOP/s peak with relative error constant at 5×10^{-4}, because conv's natural tiling keeps the array fed. This is why conv, not GEMM, is the ANE's highest-throughput primitive. A check robust to the falloff isolates the raw ceiling: deepening a fused chain of small matmuls grows compute at fixed I/O, and its slope reaches 19.6 TFLOP/s against the 10.2 single-matmul peak, so the gap is utilization, not a silicon limit.

3.2 Bandwidth roof

The compute-bound primitives (matmul, conv, and their variants) are the minority of `aneforge`'s 50 fused operators. The rest (activations, elementwise binaries, reductions, softmax, the norm family, pooling, and data-movement ops) are memory-bandwidth-bound at size. We scale each class from 256K to 64M elements and read

Just Accepted

its saturation plateau; the rise toward that plateau identifies the dispatch-bound small sizes. Table 2 shows the ceilings.

Table 2. Bandwidth ceilings (M5 Pro), GPU and ANE in fp16.

Archetype (bytes)	ANE GB/s	GPU GB/s	ANE GB/s/W	GPU GB/s/W
streaming relu / x·2 (2N)	24.1	**229.7**	1.99	**27.13**
reduction sum (1N)	22.4	**193.0**	2.24	**27.28**
softmax (2N)	15.1	**238.4**	1.78	**18.54**
layer_norm (2N)	4.4	**41.9**	0.90	**2.43**
gelu light-compute (2N)	23.2	48.6	2.03	3.70

The streaming row sets the measured GPU ceiling at 230 GB/s, with softmax reaching 238. This single-program rate is about 75 % of the part's 307 GB/s unified-memory specification and is the appropriate ceiling for a measured roofline. The ANE's achieved bandwidth through `aneforge`'s single-fused-program path is flat at 22–24 GB/s across streaming and reduction, reached already at the smallest sizes measured, about 10× below the memory ceiling the GPU attains. That flat-from-small signature is the finding: on pure elementwise and reduction graphs the ANE is bound by its program-streaming and dispatch path, not by the memory system, so the 24 GB/s is a path ceiling and not a silicon-memory one. The GPU therefore has both the higher raw bandwidth (230 against 24) and the higher bandwidth-per-watt on every archetype (27.1 against 1.99 GB/(s W) on streaming). The ANE's usual perf/watt advantage does not survive here, because it came from doing useful compute at low power and a pure memory stream is the one regime with no compute to do. The consequence is that standalone large softmax, elementwise, and reduction ops belong on the GPU; they are cheap only when fused into a larger compiled program.

3.3 Weight-stream bandwidth (GEMV-K)

This measurement establishes the first roofline correction: the ANE has two effective bandwidths, not one. The streaming roofline of Section 3.2 measures the standalone-activation path. A decode GEMV (matrix–vector multiply, $M = 1$) exercises a different path: it streams a $K \times N$ weight matrix once and does negligible compute (arithmetic intensity AI ≈ 1.0), so its effective bandwidth is that of the path the ANE uses to stream weights, not of its standalone-activation path. A single GEMV at $K = N = 4096$ gives an effective 112 GB/s, well above the 24 GB/s activation ceiling of Section 3.2. To resolve whether this is anomalous or a second bandwidth tier, we sweep $K = N \in \{512, 1024, 2048, 4096, 6144, 8192\}$ and read effective GB/s = (weight + I/O bytes, fp16) / minimum latency (the per-archetype byte models are in the online supplement, Table S16). Figure 2 reports the sweep (the per-K values are tabulated in the online supplement, Table S3); the same rise-to-plateau standard we applied to the 24 GB/s figure applies here.

The ANE's bandwidth increases 6 → 29 → 76 GB/s at small K, and so is dispatch-bound and the flat-from-small signature of Section 3.2 is inverted: here the per-program overhead is amortized as the weight matrix grows (Figure 2). The ANE reaches 113.5 GB/s for $K = 4096$, reproducing the 112 GB/s figure to within 2 %, then plateaus to 130–143 GB/s at larger K. The 112 figure is thus a conservative lower bound, and the weight-stream path is 5 to 6× the 24 GB/s standalone path.

The standalone-activation path plateaus at 23 GB/s with a stable large-size ceiling (a plateau coefficient of variation of 3.3 %) and does not rise with size, so it is a true ceiling and not an under-amortized operating point of the same DMA; the weight-stream tier is 5.4× higher. Fragmenting a fixed weight volume into many small programs also lowers the achieved weight-stream bandwidth, consistent with a separate weight-stream tier. An M2 Pro (h14, a desktop Mac mini) shows the same flat low activation ceiling at 14 GB/s and a weight-stream tier

Just Accepted

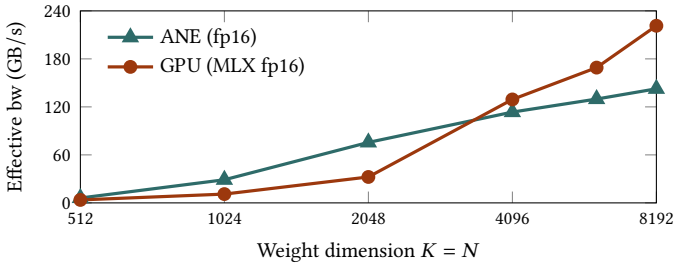

Fig. 2. Weight-stream effective bandwidth versus weight dimension $K = N$ for a decode GEMV ($M = 1$) on the M5 Pro. The ANE's bandwidth exceeds the GPU's up to $K = 2048$ (76 GB/s versus 32 GB/s) as it amortizes its dispatch floor, then plateaus near 130–143 GB/s; the GPU's unified memory is higher from $K = 4096$ and reaches 221 GB/s.

3.8× higher; an M4 (h16, a second Mac mini) shows the same flat ceiling near 10 GB/s with a weight-stream path that runs 2.4 to 6.6× above it.

Scope of the two-tier measurement. The two effective bandwidths are a property of the dispatch path as reached by `aneforge`, measured end to end from the host, and the measurement does not establish a dedicated weight DMA as the mechanism. In these programs the weight matrix is a compile-time constant of the compiled graph and the activation is the only runtime input, so a warm repetition reissues the same program against the same constant. The faster tier is also consistent with constant handling, on-chip tensor layout, compiler placement, or residency across repetitions. An external measurement cannot separate these from a distinct transfer engine, so we report a dedicated weight path as a candidate mechanism, not an established one. Every weight point exceeds the activation ceiling by at least 2.4×, so the two-tier separation holds across sizes; the online supplement notes one h16-only weight-path stride-aliasing dip at exact powers of two, which does not affect the separation. The activation path's saturation at 14 GB/s on a part with roughly 200 GB/s of unified-memory bandwidth indicates the ceiling is a dispatch-path limit, not a DRAM limit. The absolute ceilings track the part, not the generation: the two Pro parts (23 and 14 GB/s) sit above the smaller base M4 (10 GB/s), even though the M4 is the newer silicon.

The ANE's single-GEMV bandwidth exceeds the GPU's through $K = 2048$ (29 versus 11 at $K = 1024$, 76 versus 32 at $K = 2048$); from $K = 4096$ the GPU's unified memory is higher, rising 129 → 221 GB/s while the ANE saturates near 143. (The CPU's fp32 figures are highest only at the smallest sizes, where the tiny weight fits in cache and the comparison is call-overhead-bound, not bandwidth-bound; it falls to 50–60 GB/s once the matrix spills.)

The energy column shows the ANE streams weights at higher GB/s/W than the GPU at every K, 2.0× at $K = 4096$ (35.9 versus 18.0) and still 1.45× at $K = 8192$ (60.8 versus 41.9), past the bandwidth crossover. Even where the GPU moves more bytes per second, the ANE moves them at lower energy on the exact path decode uses, which is the roofline-level reason the ANE's decode advantage is energy (Section 9).

The weight-stream path also accepts compressed weights, decompressed during the DMA (the on-the-fly-decode pattern of EIE [12]); `aneforge` emits int4 lookup-table palettization and unstructured sparse forms behind one opt-in flag. On a single matmul this raises arithmetic intensity and speeds the stream (1.65 to 1.82× on large layers, online supplement Table S4), moving the int4 operating point to 104 % of the matmul roof at $B = 128$ (Table 4). End to end, int8's benefit is footprint and energy, not latency: the int8 weight stream is bound by the dequantizer's element rate, not by bytes, the third roofline correction, developed in Section 9 (halving the weight bytes yields energy savings on a part at its element-rate ceiling, and throughput only on a bandwidth-starved one, though a lookup-table encoding raises the ceiling itself).

Just Accepted

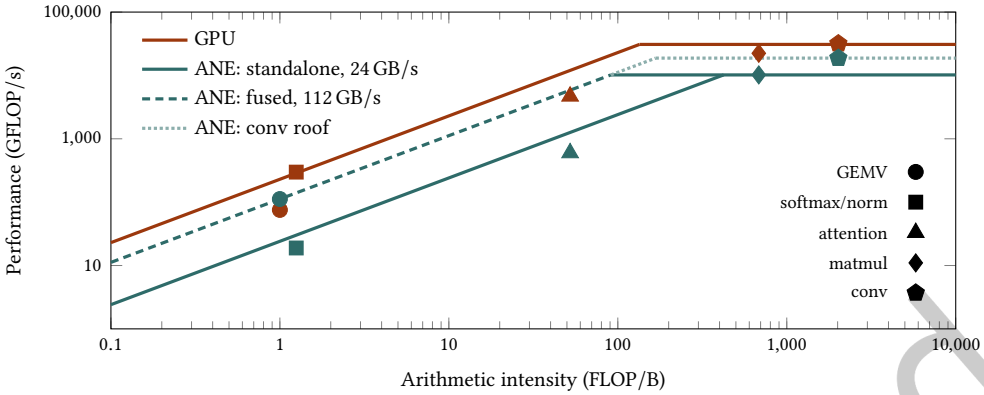

Fig. 3. Per-device roofline on the M5 Pro: five operator archetypes on the ANE (triangles) and the GPU (circles), placed by arithmetic intensity. The ANE shows two bandwidth slopes, a 24 GB/s standalone-activation path and a faster weight-stream path used by compiled programs; per-archetype, per-device placement is in Tables 3 and 4.

4 Per-device roofline synthesis

The corrected roofline turns on one inequality: the ANE's weight-stream ridge (about 71 FLOP/B) lies below the GPU's (134), so the path that carries every compute-bound ANE operator becomes compute-bound at a lower arithmetic intensity than the GPU does.

A roofline positions each operator by its arithmetic intensity (AI = FLOPs / bytes-moved) and predicts an achievable ceiling of `min(compute_roof, bandwidth_roof × AI)`; the ridge point (`compute_roof` / `bandwidth_roof`) divides memory-bound (left) from compute-bound (right). Table 3 gives each device's roofs and ridges (recomputed from the two ceiling JSONs, not re-measured); Table 4 then tabulates every measured archetype by its arithmetic intensity, reporting its achieved rate and the fraction of the applicable roof it reaches on each device. Only the high-AI matmul and conv archetypes exceed any device's ridge (compute-bound); everything below falls on the bandwidth slope.

Table 3. Per-device roof and ridge (recomputed from the saturation and bandwidth JSONs). The ANE has two bandwidths (Sections 3.2 and 3.3) and therefore two ridges: a standalone-activation path near 24 GB/s and a weight-stream path near 143 GB/s.

Device	Dtype	Compute roof (GFLOP/s)	Bandwidth roof (GB/s)	Ridge (FLOP/byte)
GPU	fp16	gemm **30862** / conv 31790	229.7	**134.4**
ANE (activation path)	fp16	gemm **10191** / conv 18771	24.1	**423.6**
ANE (weight-stream path)	fp16	gemm **10191** / conv 18771	~143	**~71**

On its standalone-activation path the ridge is about 424 FLOP/B (3.2× the GPU's 134), a consequence of that path's bandwidth being only 24 GB/s against 10 TFLOP/s of compute. This is why dispatching an elementwise, norm, or softmax op one at a time underutilizes the engine: such ops lie far to the left of 424 and never approach compute-bound.

Compute-bound work (GEMM, conv, GEMV, linear, anything that streams weights) instead uses the weight-stream path (Section 3.3), whose ridge is only about 71 FLOP/B, below the GPU's 134. On the path that carries the ANE's compute-bound work, an operator becomes compute-bound at a lower arithmetic intensity than on the

Just Accepted

Table 4. Operator archetypes placed on each device's roofline: arithmetic intensity (AI, the fp16 value), achieved performance, and the fraction of the applicable roof (`min(compute, bandwidth × AI)`) reached, M5 Pro. Only the high-AI matmul and conv rows exceed any ridge (compute-bound); the rest fall on the bandwidth slope. ‡The ANE GEMV is scored against the weight-path roof (78 %), not the 24 GB/s activation roof (a spurious 466 %), since a weight-streaming GEMV uses the weight path.

Operator archetype	AI	GPU GFLOP/s (%roof)	ANE GFLOP/s (%roof)
matmul $N = 512$	171	1235 (4 %)	1819 (18 %)
matmul $N = 2048$	683	22345 (72 %)	10191 (100 %)
matmul $N = 8192$	2731	30527 (99 %)	4882 (48 %)
conv ($C = 64$)	287	5343 (17 %)	5885 (31 %)
conv ($C = 512$)	2020	31790 (100 %)	18771 (100 %)
attention ($S = 512$)	52	4685 (39 %)	598 (48 %)
GEMV / decode	1.0	75 (33 %)	112 (78 %)‡
softmax	1.25	298 (104 %)	19 (63 %)
gelu	2.5	122 (21 %)	58 (96 %)
layer_norm	2.5	105 (18 %)	11 (18 %)
relu / copy	0.25	57 (100 %)	6 (100 %)
matmul int4 (4096^2, $B = 1$)	4	171 (19 %)	165 (29 %)
matmul int4 (4096^2, $B = 128$)	410	5218 (17 %)	10579 (104 %)

GPU, not higher. This marks where the ANE's energy advantage begins, not which device is faster: throughput follows the lower, falling compute roof (Section 3.1), so on large square GEMM the GPU is the faster device while the ANE still draws far less power, whereas on conv and the stencil the ANE is faster and lower-power. A single ridge at 424 conflates the activation and weight paths and places the weight-streaming points above the roof. With both ridges, the GEMV reaches 78 % of its applicable weight-path roof instead of the 466 % the activation ridge implies. The CPU (fp32 via Accelerate/AMX) is a contextual reference whose full figures are in the online supplement (Table S18); the like-for-like comparison is GPU versus ANE, which run identical fp16 math.

The fused 3-conv block at about 110 % of the single-conv roof exceeds a standalone-dispatch ceiling by amortizing per-dispatch overhead and I/O (the fusion discussion below). The GPU softmax at 104 % is FLOP-convention sensitivity (the flat per-element transcendental allowance of the online supplement).

The ANE roofline carries two caveats. It is not a single clean line: its compute roof differs by operator (conv about 18.8 versus square-matmul about 10.2 TFLOP/s, and matmul falls off past $N = 2048$ to about 4.9 TFLOP/s at $N = 8192$), so we draw both roofs. The 24 GB/s bandwidth and the ridge near 424 FLOP/B are also a path operating point of `aneforge`'s single-program streaming, not a silicon DMA limit (Section 3.2); the weight-stream path reaches 5 to 6× higher at its plateau (Section 3.3). In Figure 3 we draw the weight-stream line at the conservative 112 GB/s operating point corroborated at $K = 4096$ (ridge about 91 FLOP/B); the measured plateau reaches about 143 GB/s, which gives the ridge of about 71 used in Table 3, and the regime conclusions are identical under either convention.

The archetype arithmetic intensities (fp16) span four orders of magnitude, from 0.25 for elementwise ops to the thousands for conv, and scoring achieved throughput against `roof@AI` sorts them into three regimes. Large matmul and conv are the only compute-bound archetypes, so they are the only classes that approach the compute roof. Convolution gives the ANE's largest measured advantage, about 18.8 TFLOP/s and 1370 GFLOP/s/W. GEMV and decode (AI ≈ 1.0) lie below every ridge and are memory-bound on both devices, so decode throughput follows whichever device streams weights fastest. The operator-level GEMV result corroborates the second bandwidth tier but does not bound a fused token loop; Section 9 measures that loop end to end. Standalone elementwise,

Just Accepted

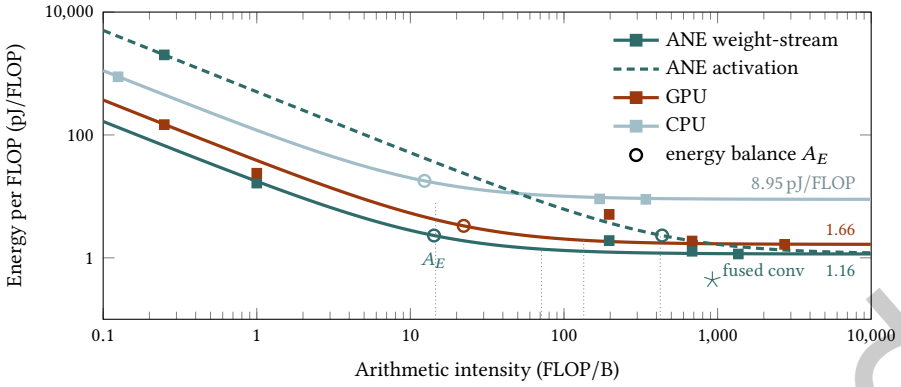

Fig. 4. Energy roofline (energy per FLOP versus arithmetic intensity) on the M5 Pro, fitted from the same active-package-power measurements as the perf/watt tables. Solid green is the ANE weight-stream path, dashed green the standalone-activation path; open circles mark each path's energy balance point AI_E and gray droplines the corresponding time ridge AI_T.

norm, and softmax ops reach only a fraction of even the ANE's low bandwidth roof (layer_norm 18 %, softmax 63 %), since its ridge near 424 demands about 3× the AI of the GPU to become compute-bound.

Fusion acts through two separable mechanisms. One is arithmetic intensity: fusing operators into one compiled program keeps intermediates on-chip and streams weights once, raising effective AI (a standalone conv at about 1076 rises to about 2854 in a fused 3-stack) and moving low-AI points rightward past the ridge, so elementwise, norm and softmax ops run at negligible marginal cost inside a fused program instead of standalone at low utilization. The other is overhead amortization: a fused graph of any depth lowers to a single `AneInference` dispatch, so depth amortizes the per-program floor and fixed I/O setup that a standalone dispatch pays in full, while each bridge-induced graph cut becomes a separate dispatch.

The three- and sixteen-conv stacks reach 110 % and 124 % of the single-conv saturation rate even though both lie beyond every ridge, where raising AI predicts no further gain. These values reflect amortized dispatch and I/O cost, not a violated hardware bound. We therefore call the measured single-dispatch curve an empirical operating envelope, not a hardware upper bound: it is the highest rate one dispatch reaches at that shape, carrying the per-dispatch cost and DRAM I/O that a fused program amortizes, and the fused-chain slope of Section 3.1 says the same for GEMM. An entry above 100 % in Table 4 therefore measures amortization, not a violated bound.

Fusion also produces the gain the roofline does predict from arithmetic intensity. Fusing a memory-bound neck block (a 1×1 convolution, group norm, and two gelus) moves its arithmetic intensity from 32.7–52.6 below the weight ridge to 119.9–131.2 above it. Throughput rises by 3.7× at $C = 256$ and 2.3× at $C = 512$.

A fused conv or attention graph is efficient on the ANE; the identical ops dispatched standalone are not.

5 An energy roofline and a predictor

The time roofline above bounds throughput; the energy roofline of Choi et al. [9] bounds energy per operation, $E(\mathrm{AI}) = e_{\mathrm{flop}} + e_{\mathrm{byte}}/\mathrm{AI}$, with an energy balance point $\mathrm{AI}_E = e_{\mathrm{byte}}/e_{\mathrm{flop}}$ that, like the time ridge, divides a memory-dominated from a compute-dominated regime but in joules, not seconds. Because every operating point in this paper already carries active package power, the two constants are fitted directly from the ceiling sweeps with no new measurement: e_{flop} is the inverse of each device's peak GFLOP/s/W and e_{byte} the inverse of its GB/s/W streaming plateau (Tables 1 and 2). Figure 4 draws the result for all three engines, with the ANE split across its two byte paths.

Just Accepted

The fitted constants explain the ANE's efficiency: a compute floor of 1.16 pJ/FLOP (GPU 1.66) and the cheapest byte in the system, 16.5 pJ/B on the weight path (GPU 36.3), while its standalone-activation byte is the most expensive at 503 pJ/B, 96 % of it host dispatch. The per-watt advantage thus comes from cheaper FLOPs, much cheaper weight bytes, and fusion keeping activation bytes off DRAM, not from the activation path. The energy balance point lies left of the time ridge on both fast paths ($\mathrm{AI}_E \approx 22$ versus $\mathrm{AI}_T \approx 134$ on the GPU, 14 versus 71 on the ANE weight path), Choi's regime where raising arithmetic intensity yields speed but no further energy savings; only the ANE activation path has the two coincident (435, 424).

The two per-device constants also make the energy roofline predictive. Given a workload's arithmetic intensity and dispatch path, $E_d(\mathrm{AI}) = e_{\mathrm{flop},d} + e_{\mathrm{byte},d}/\mathrm{AI}$ estimates its energy per FLOP on device d, and comparing E_{ANE} against E_{GPU} identifies the lower-energy engine. The dispatch path is a second model input and cannot be inferred from arithmetic intensity. An operator compiled into the fused graph reaches the weight-stream path at 16.5 pJ/B; the same operator issued standalone reaches the activation path at 503 pJ/B. The predictor takes that dispatch state as given, and does not derive it. The device-pair energy crossover is $\mathrm{AI}^\star = (e_{\mathrm{byte,ANE}} - e_{\mathrm{byte,GPU}})/(e_{\mathrm{flop,GPU}} - e_{\mathrm{flop,ANE}})$. On the ANE weight-stream path the ANE has both the lower compute floor and the cheaper byte, so $\mathrm{AI}^\star$ is negative: the ANE is the lower-energy device at every arithmetic intensity, with no crossover. The GPU has no separate weight path, so it carries a single streaming byte (36.3 pJ/B); that byte reproduces its measured weight-streaming energy (online supplement, Table S19), so the crossover compares each device's energy byte, not mismatched paths. On the standalone-activation path the expensive 503 pJ/B moves the crossover up to $\mathrm{AI}^\star \approx 933$, above which only fusing the op back onto the weight path keeps the ANE ahead. The predictor therefore has one lever: keep FLOP-denominated work on the weight-stream path (compiled and fused), where the ANE is the lower-energy device unconditionally, while a standalone activation-path op below AI 933 belongs on the GPU.

We test the predictor on workloads it was not fitted to, since the constants come only from the ceiling sweeps of Sections 3.1 and 3.2. The predictor makes two separable claims, on placement and on magnitude. On the weight path the crossover is negative, so the model makes one prediction, the ANE, for every weight-streaming workload; the device-map archetypes and the $B = 1$ decode loop bear it out (Tables 5 and 8), but this is a single sign, not a test per workload. The discriminating predictions are on the activation path, where the AI-933 crossover sends a standalone low-intensity op to the GPU. The standalone group norm (AI 2.5) is GPU-favored in Table 5, as predicted, and the below-ridge fusion block switches from the GPU when standalone (AI 32.7 to 52.6) to the ANE when fused (AI 119.9 to 131.2), the measured 3.7× and 2.3× gains confirming the ANE side.

The magnitude claim follows from the roofline being a lower bound on energy per FLOP for single-dispatch work: the predicted joules are accurate near the roof and a one-sided bound elsewhere. Tested against the held-out device-map classes (online supplement, Table S19), the predicted energy per FLOP is below measurement on every single-dispatch class, as a lower bound requires. Near the roof it lands within 7–28 % (median 14 % across both devices, on compute-bound GEMM, long-sequence attention, and the $B = 1$ decode loop). Away from the roof it under-predicts the dispatch-bound rows by large factors, whose package energy is dominated by the fixed per-call cost the roofline omits. The bound's domain excludes dispatch-amortized graphs: the fused 16-deep conv stack runs at 0.44 pJ/FLOP, beneath the fitted floor, because one dispatch covers sixteen layers (Section 4).

A dispatch term. The gap away from the roof is a fixed per-call cost, so we model it. Writing the roofline per call as $E_{\mathrm{call}} = e_{\mathrm{flop}}F + e_{\mathrm{byte}}B$ with $B = F/\mathrm{AI}$, and adding one constant,

$$E_{\mathrm{call}} = e_{\mathrm{flop}}F + e_{\mathrm{byte}}B + E_{\mathrm{disp}}, \qquad T_{\mathrm{call}} = \max\left(\frac{F}{P_{\mathrm{peak}}}, \frac{B}{BW_{\mathrm{peak}}}\right) + T_{\mathrm{disp}}, \tag{1}$$

where e_{flop}, e_{byte}, P_{peak} and BW_{peak} keep the values already fitted from the ceiling sweeps, so each equation gains exactly one parameter. Across the held-out classes the work F spans a factor of 1024 while the roofline's residual

Just Accepted

spans only 2.8, from 354–987 µJ on the ANE. The omitted cost therefore does not scale appreciably with work. The fitted ANE constants are $E_{\text{disp}} = 681\,\mu\text{J}$ and $T_{\text{disp}} = 136\,\mu\text{s}$ per call; the GPU values are 1771 µJ and 67 µs. Adding them cuts the median absolute error on energy per FLOP from 56.6 % to 13.7 %. Scoring each class with the constant refitted without it gives 16.5 %. The dispatch-floor classes carry the improvement: the smallest GEMM moves from −97.6 % to 30.1 % and the single conv from −97.0 % to −8.4 %, while the near-roof classes stay within 5 %.

An independent measurement confirms that the fitted T_{disp} is a dispatch floor, not a free parameter absorbing error. The per-program floor of about 70 µs (Section 9) comes from the minimum-latency probe, one dispatch under the most favorable conditions. Equation (1) is instead fitted to sustained execution, whose per-call cost is larger by the measured sustained-to-probe ratio of 1.99 at that shape. The two agree to 2.5 %: $70 \times 1.99 = 139\,\mu\text{s}$ expected against 136 µs fitted. We report the GPU's term as fitted only. Its independently measured 184 µs intercept comes from the stencil step-count sweep, a compiled 32-step program launch, not a single-operator dispatch, and therefore measures a different quantity.

The corrected roofline is thus a placement rule: from a workload's arithmetic intensity and its dispatch path, assign it to the lower-energy engine and fuse to keep it on the weight-stream path. Two of the three inputs reproduce across generations: the compute floor stays near 1 pJ/FLOP (1.16 on the M5 Pro, 0.96 on the M4, 1.3 on the M2 Pro) and the weight-path placement holds on every measured part (Section 7). The byte constant does not: the bandwidth-starved base M4 carries a higher effective weight byte (about 34 against the M5 Pro's 16.5 pJ/B), the byte-bound-versus-element-bound contrast of Section 9, so the predictor's energy magnitude is host-specific and reported on the fitted M5 Pro alone.

6 Single-stream device map

From each operator's arithmetic intensity, the corrected roofline predicts the single-stream device map measured here. The single-stream ($B = 1$) device map addresses the first practical choice: for one inference on one input, which engine a given operator class should run on. Table 5 reports latency, fp16 relative error, idle-subtracted active package power with CV%, and perf/watt for the ANE and GPU across thirteen synthetic workload classes; Table 6 reports three real models with energy per inference.

The single-stream map shows that the ANE draws less power on every in-graph-expressible workload tested (Figure 5), a 1.02 to 3.8× reduction in active package power, even on the workloads where the GPU is faster. On large square fp16 GEMM ($K = 4096$) the GPU is 2.8× faster and more accurate (3.6×10^{-4} versus the ANE's 2.7×10^{-2}), yet the ANE still does the work on 8.71 W against the GPU's 33.05 W. The ANE has the lower latency on convolution, convolution stacks (4.2× on the ResNet-like block), the stencil (2.8× against an optimized GPU stencil), short attention, mid-size GEMM, and the smaller real models (ResNet-18 5.0× and MiniLM 1.2× against compiled fp16 MLX). It has the higher perf/watt on 11 of 13 classes, by the largest margin on the ResNet-like conv stack (13×), then the stencil (4.7×) and the Jacobi solve (3.6×). The remaining two classes sit at or near parity, and repetition separates them. Over ten independent cold starts the GPU advantage on `group_norm` is small and reproducible (0.949, 95% CI [0.900, 0.997]). The DFT ratio is 1.35 with a CI of [0.96, 1.73] that spans parity, so that class carries no directional verdict (online supplement, formulation-fairness appendix). The exception among the real models is ViT-B/16, on which the GPU is faster (Table 6). Against torch-MPS fp16 the margin is small (15.9 versus the ANE's 18.3 ms) and the ANE's energy advantage is 9.4×. Graph-compiled MLX runs the model in 2.96 ms, 6.2× faster than the ANE, and the ANE's energy advantage drops to 1.5× (95% CI [1.3, 1.8]). Compiled MLX also halves MiniLM's advantage, from 6.7× to 3.4×, though the ANE remains the faster device there. The GPU has the lower latency on large square GEMM, long-sequence attention, and the small norm reductions, and is more accurate in fp16 everywhere.

Just Accepted

Table 5. Single-stream device map (M5 Pro, 6 s power window, idle package = 1196 mW); metrics defined in Section 2, with CV% in parentheses in the W columns; fp16 relative error per class is in the online supplement (Table S20). Best speed and perf/W per row in **bold**; GF/W is GFLOP/s per watt, the norm rows elements/s per watt. Shapes: GEMM $M{\cdot}K{\cdot}N$; attention sequence length S ($D = 768$, $H = 12$); conv rows are a single $16 \to 64$ layer and a stack of one $64 \to 256$ then fifteen $256 \to 256$ layers (18.4 GFLOP); stencil counts the dense 3×3 conv (a strict 5-point count scales every device by 5/9, leaving ratios unchanged). †The stencil GPU column is an optimized GPU stencil (shifted-slice arithmetic under `mx.compile`), not the dense 3×3 convolution the ANE runs, and both columns are N=5 means; the DFT row's single-session ordering is not a verdict (online supplement, formulation-fairness appendix). *Replicated over independent cold starts: the stencil (N=5), `group_norm` and the DFT (N=10, intervals in Figure 5), and the conv stack, ViT attention, and GEMM-compute rows (online supplement, Table S17). The seven unmarked rows are single-session.

Class	Workload	ANE lat	GPU lat	ANE W (CV)	GPU W (CV)	ANE perf/W	GPU perf/W
GEMM floor	M64·K256·N256	**0.092**	0.124	**2.93 (11 %)**	6.54 (5 %)	**15.7** GF/W	7.8 GF/W
GEMM bandwidth	M128·K1024·N1024	**0.086**	0.187	9.22 (7 %)	9.40 (6 %)	**274** GF/W	160 GF/W
GEMM compute*	M256·K4096·N4096	1.293	**0.461**	**8.71 (4 %)**	33.05 (8 %)	**751** GF/W	499 GF/W
conv	single 3×3 ($16 \to 64$)	**0.103**	0.173	**5.99 (12 %)**	7.61 (3 %)	**24.5** GF/W	14.7 GF/W
conv*	ResNet 3×3×16 ($64 \to 256$)	**0.757**	3.159	**10.12 (6 %)**	32.70 (2 %)	**2289** GF/W	175 GF/W
attention*	ViT block $S = 197$	**0.230**	0.269	**8.50 (17 %)**	16.01 (11 %)	**522** GF/W	197 GF/W
attention	long-seq $S = 512$	0.817	**0.390**	**6.89 (7 %)**	23.22 (11 %)	**561** GF/W	305 GF/W
norm	layer_norm (197,768)	0.195	**0.176**	**5.76 (12 %)**	9.03 (6 %)	**1.25e8**/W	9.24e7/W
norm	rms_norm (197,768)	0.373	**0.151**	**3.44 (6 %)**	7.75 (6 %)	**1.13e8**/W	1.12e8/W
norm*	group_norm ($1{,}256{,}64^2$)	0.350	**0.196**	**7.89 (7 %)**	10.82 (9 %)	3.63e8/W	**3.85e8**/W
sci*	5-pt stencil 256^2, 32 steps†	**0.137**	0.382	**7.54 (14 %)**	11.58 (1 %)	**37.1** GF/W	7.86 GF/W
sci	Jacobi n=512, 25 iters	**0.170**	0.285	**5.24 (13 %)**	9.98 (6 %)	**14.5** GF/W	3.98 GF/W
sci*	DFT-as-matmul N=1024	**0.176**	0.185	**4.06 (25 %)**	8.24 (5 %)	2.45 GF/W	**2.99** GF/W

Table 6. Real models (latency ms; energy mJ/inference; all GPU arms fp16 unless marked). `MPS` is torch-MPS (`.half()`) and `MLX-c` is graph-compiled MLX (`mx.compile`); MiniLM and ViT-B/16 run as MLX ports loading the same weights (online supplement, formulation-fairness appendix). The ANE and `MPS` energies are means over N=16 cold starts, and *adv. MPS* is their geometric-mean ratio with a 95 % interval, whose width is set mostly by the GPU arm (online supplement, Table S6). The fp32 `MPS` column† is single-session and carries no interval. The `MLX-c` energies and *adv. MLX-c* are paired repetitions interleaved with the ANE within one session (N=5 for ResNet-18, N=10 for the others), with a 95 % t-interval on the mean ratio. Each *adv. MLX-c* ratio pairs a compiled-MLX run with an ANE run from the same session, so it is not the quotient of the printed cold-start ANE energy (the in-session ANE energies are in the online supplement, formulation-fairness appendix). Latency is the device-map probe, except `MLX-c`, which is the sustained time of its energy window.

	Latency (ms)				Energy (mJ/inf)				ANE energy advantage	
Model	ANE	MPS	MLX-c	ANE relerr	ANE	MPS fp32†	MPS	MLX-c	adv. MPS	adv. MLX-c
ResNet-18	**0.330**	2.033	1.645	2.3e-3	**2.30**	39.1	21.3	32.6	8.9× [7.5, 10.7]	14.6× [12.1, 17.2]
MiniLM (1 sent)	**0.525**	1.920	0.650	5.5e-3	**2.74**	28.8	17.6	11.5	6.7× [5.9, 7.6]	3.4× [3.0, 3.9]
ViT-B/16	18.34	15.92	**2.959**	7.8e-3	**65.7**	1382.7	577.9	100.4	9.4× [7.9, 11.1]	1.5× [1.3, 1.8]

A precision-versus-speed rule follows from the measured fp16 accuracy. Use the ANE's fp16 speed and efficiency where the math is well-conditioned and not a large square GEMM (convolution, stencils, short attention, mid-size GEMM, encoders); there its relative error of 10^{-3} to 10^{-2} is the same order as GPU fp16 and it is usually both faster and lower-power. Hand large compute-bound square matmuls ($K \geq 4096$) and long sequences to the GPU. Use fp32 wherever cancellation can dominate (an adversarial, ill-conditioned reduction), where fp16 error is unbounded. The headline perf/watt is the conservative total-package number that charges the ANE for its own

Just Accepted

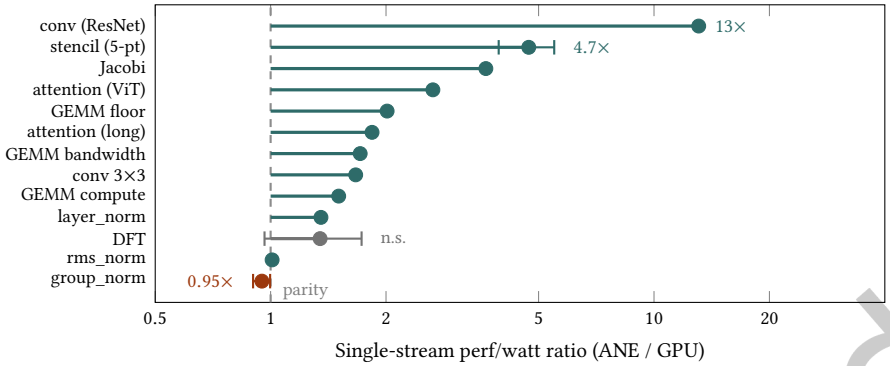

Fig. 5. Single-stream perf/watt advantage, ANE divided by GPU, per workload class on the M5 Pro (Table 5), sorted and drawn against a parity line. The ANE is more energy-efficient on 11 of 13 classes, by up to 13× on the ResNet-like convolution stack; the stencil is scored against an optimized GPU stencil, which places it at 4.7×. Error bars are 95% CIs over independent cold starts for the stencil, the DFT, and `group_norm`; the DFT interval spans parity, so that class supports no directional verdict (n.s.). The conv stack, ViT attention, and GEMM-compute classes are also replicated (perf/watt differences with CIs in online supplement Table S17), and Table 5 marks every replicated class.

host CPU dispatch, and the ANE is the more efficient device on it for 11 of the 13 device-map classes, so the efficiency is silicon; the ANE-rail-only silicon bound is 1.2 to 4× larger (online supplement, dual-bound table).

7 Cross-generation reproduction

The single-stream watt harness also runs on three further hosts: an M2 Pro (h14 ANE, a 16 GB Mac mini), an M1 Max (h13, a MacBook Pro), and an M4 (h16, a second 16 GB Mac mini, base part), so the reproduction spans four generations and both chassis classes. The online supplement gives the per-class maps (Tables S11–S13); h16's three real-model rows were not obtained, as ResNet-18 hung at compile on that part. Those maps cover the eleven synthetic classes common to all four hosts, two fewer than Table 5, since Jacobi and the DFT-as-matmul were added to the reference map afterwards and are M5 Pro only.

Every substantial-workload result reproduces, where substantial means the classes above the dispatch floor: conv stacks, attention, compute-scale GEMM, the stencil, and the real models where they ran. Every substantial class favors the same device on all four generations, and the ANE is the more efficient device in each case. The conv-stack perf/watt ratio stays between 12.9 and 14.5× across the four parts. Non-reproducing rows are confined to the dispatch-floor and tiny-norm classes, where host dispatch dominates package power. These shift toward the GPU on the older parts and back on the newest. Across the eleven classes, the faster device matches the M5 map on 7 and the more efficient device on 7, with 3 matching on both axes. On h16 the disagreements shrink instead of growing, the GPU keeping a perf/watt edge only on `group_norm`.

The bandwidth, serving, and decode sweeps reproduce in structure on the desktop parts. Both h14 and h16 show the same low-activation-ceiling asymmetry. On h14 the batched-serving throughput crossover falls near $B = 36$, with no throughput-per-watt crossover on any of the four workloads. On h16 the ANE leads in tokens-per-watt at every batch, by a wider margin than on the M5 Pro. It does not run on h14, whose 16 384 maximum tensor dimension the 32 000-token vocabulary head exceeds. A fused-GPU baseline (`mx.compile`) shows the large gaps are not MLX launch overhead: fusion helps only the small elementwise rows.

Just Accepted

Formulation is a separate confound, and one class is bound by it: expressing the 5-point stencil as a dense 3×3 convolution costs the GPU 3.3× in time and 2.9× in power, moving that row from 43.8× to 4.7× (online supplement, fused-GPU and formulation-fairness appendices).

8 Batched serving

The device map of Section 6 is the single-stream ($B = 1$) choice. Serving is different: a deployment batches requests. We sweep batch size $B \in \{1, 4, 16, 64, 256\}$ with true batching, one compiled program with a real batch dimension, not multi-stream, across four serving workloads, and report the two deployment-relevant crossovers: the batch at which GPU throughput first exceeds the ANE's, and the batch at which GPU throughput-per-watt does. Both devices run the same fp16 math at the same B, warm, steady-state; accuracy is flat in B.

The throughput crossover appears on three of the four workloads, and the throughput-per-watt crossover on none of them within the swept range. The online supplement (Tables S7–S10) gives the sweeps; Table 7 summarizes the crossovers.

Table 7. Crossover summary: the batch at which GPU throughput, and GPU throughput-per-watt, first exceed the ANE's, per serving workload. A **never** entry marks a workload where the ANE leads at every swept B, with the ANE/GPU multiplier range in parentheses.

Workload	Throughput crossover B^*	Throughput/W crossover
Vision (conv classifier)	**never** (3.1–5.7×)	**never** (6–10×)
Encoder (transformer block)	~**23–31** (replicate 31.1 ± 1.3)	**never** (1.9–5.4×)
Attention (self-attention block)	**small batch** (replicate 2.23 ± 0.02)	**never** (1.7–2.4×)
Batched GEMM	~**5** ($B = 4 \rightarrow 16$)	**never** (1.06× at $B = 64$; Section 10)

For vision both crossovers are absent: the ANE leads at every batch on throughput and perf/watt. For the transformer workloads the GPU's raw throughput crosses above the ANE's at small-to-moderate batch (attention near $B = 2$, encoder near $B = 31$; Section 10), while the ANE keeps the higher throughput-per-watt across the full swept range, and on bare large-batch GEMM it retains about 6 % (resolved under the replicate protocol of Section 10). The GPU is the throughput device for batched transformers; the ANE's serving role there is the low-batch and energy-capped regime. The throughput crossover depends on the workload, falling near $B = 2$ for a self-attention block and near $B = 31$ for a transformer encoder. Above its own crossover the GPU maximizes throughput. The ANE remains preferable whenever sequences-per-joule is the target, and for vision under both metrics.

9 End-to-end LLM decode

The serving sweeps above measure encoder, attention, and conv blocks. The remaining serving case, and the one where a single operating point most easily misleads, is autoregressive LLM decode, the per-token forward pass of a generative transformer. The synthetic GEMV operating point (Table 4: ANE 112 > CPU 81 > GPU 75 GFLOP/s at AI ≈ 1.0) suggests an ANE single-stream decode advantage. To test whether that op-level point transfers, we measured a full decode loop: one token is a complete 8-layer transformer forward at sequence position 1, with the key–value (KV) cache (length 256) fed as graph inputs. The configuration is a TinyLlama/GPT-small class model (`d_model=2048`, 8 layers, 16 heads (`d_head=128`), a 4× feed-forward network (FFN, `d_ff=8192`), vocab 32000, KV-len 256) at 1.22 GFLOP/token ($B = 1$) and 39.1 GFLOP/token ($B = 32$). The ANE attention is built from `bmm` and `softmax` primitives, not `af.sdpa` (which needs equal-shape query/key/value tensors and so does not apply to the $M = 1$ decode step). The GPU runs MLX fp16, the CPU numpy/Accelerate fp32 (the reference); ANE

Just Accepted

and GPU run the same fp16 math. Table 8 reports the result with the same per-engine energy accounting as the rest of the paper.

Table 8. End-to-end LLM decode, batch sweep (M5 Pro; 8-layer forward per token at seq pos 1; KV-len 256; ANE attention = bmm+softmax; idle pkg ≈ 1480 mW). Throughput (tok/s) and idle-subtracted total-package efficiency (tok/s/W) for the ANE (fp16 and per-channel int8 weight-stream) and GPU (MLX fp16). The fp32 relative-error reference is numpy/Accelerate. Best throughput and best efficiency per batch in **bold**. fp16-versus-fp32 relative error ≈ 2.5×10^{-2} (ANE fp16), 5.0×10^{-2} (ANE int8), 3.2×10^{-3} (GPU). The GPU's $B = 2$ efficiency dip is a power-state step (the GPU rail pins at 19.9 W for all $B \geq 2$ before batching amortizes it).

B	ANE tok/s	int8 tok/s	GPU tok/s	ANE-fp16 tok/s/W	ANE-int8 tok/s/W	GPU tok/s/W
1	115	123	**200**	35.9	**44.7**	19.3
2	204	204	**231**	50.6	**68.7**	9.9
4	341	342	**431**	**68.6**	64.8	18.5
8	517	515	**834**	88.2	**100.3**	36.9
16	704	698	**1444**	106.2	**113.7**	65.1
32	847	850	**2322**	88.0	105.4	**116.4**

The op-level GEMV ordering does not carry to end-to-end decode throughput. Measured on a real loop, the GPU has the higher throughput at every batch in fp16 and int8, 1.74× the ANE at $B = 1$ (200 versus 115 tok/s), widening to 2.7× at $B = 32$ (2322 versus 847), the reverse of the single-GEMV microbenchmark.

The per-token time budget explains both the deficit and the reversal. The decode forward compiles to a single fused program per token step, so the per-program dispatch floor [6] (about 70 µs on this part) is paid once per token, under one percent of the 8.7 ms fp16 token. The budget is instead dominated by the weight stream: the model as measured carries 602.4 M weight parameters across the eight decoder blocks and the vocabulary projection, 1.20 GB in fp16. The batch sweep separates the costs cleanly, since per-step latency is linear in batch with intercept 7.9 ms and slope 0.93 ms per stream (residuals under 1.5 % at all six batch points). The intercept is the batch-invariant weight stream, 1.20 GB at an effective 154 GB/s. This agrees with the standalone GEMV sweep after removing 56 amortized dispatch floors from the 7.6 ms total for its 57 constituent GEMVs. The slope is the per-stream traffic, chiefly the 16.8 MB KV cache fed through the slower activation path. Single-stream decode on the ANE is therefore weight-stream-bound; dispatch would dominate only if the stack were issued one program per operation, costing 57 floors and capping the loop near 80 tok/s.

The GPU streams the same way faster: at 200 tok/s it moves the same 1.20 GB in 5.0 ms, about 240 GB/s, near its achieved streaming ceiling. Decode is weight-stream-bound on both devices, as the roofline predicts, and the GPU's higher decode throughput follows from its faster weight stream. The GEMV microbenchmark corroborates the weight-stream tier (Sections 3.3 and 4), though standalone small-K GEMVs are dispatch-floor-dominated and do not bound the fused loop.

int8 weight streaming identifies which resource binds the stream. Decode is the AI ≈ 1.0 memory-bound regime, so per-channel int8 (half the weight bytes) is the one change that could move throughput, not energy alone. It barely does: int8 lifts $B = 1$ throughput by 7 % (115 → 123 tok/s) and then matches fp16 for every $B \geq 2$ (within 1 %). A purely byte-bandwidth-bound stream would roughly double under this compression, and the measured change is near null. Combined with the weight-stream-bound budget above, that places the limit at the rate weight elements pass the dequantize-and-multiply stage, about 77 Gelem/s, and not at DRAM bytes.

That ceiling belongs to the dequantize stage, not to the weight stream, and a third encoding separates the two. Streaming the same weights as an int4 lookup table quarters the weight blob (1149–287 MB) and raises $B = 1$ throughput to 203 tok/s, 1.65× the int8 rate, where an element-rate limit predicts no change. That is level with the GPU's fp16 rate of 200 tok/s; we did not measure an int4 GPU arm, so this is not a like-for-like

Just Accepted

throughput comparison. The encodings compile to different constant operators, `constexpr_affine_dequantize` for int8 and `constexpr_lut_to_dense` for int4, and the measured rates separate with them: fp16 streams at 140 GB/s and 70 Gelem/s, int8 at 74 GB/s and 74 Gelem/s (the 77 Gelem/s above, re-measured), and int4 at 61 GB/s and 122 Gelem/s. The fp16 stream is thus byte-bound at the 130–143 GB/s GEMV plateau, the int8 stream is element-rate-bound below it, and the lookup-table stream clears both. The ceiling is therefore a property of the weight encoding, not of precision: which constant operator the encoding lowers to sets the rate, and the three operators measured give three rates. The int4 program is inaccurate on this synthetic model, disagreeing with the fp64 reference on the greedy token at a rank correlation of 0.82 against 0.998 for int8, so its output alone does not establish that the encoding streams every weight. ResNet-18 does. Requested at the default budget the encoding falls back to int8 and reproduces the int8 row exactly (69.65 % top-1, 2.33 % label disagreement); engaged, it is a distinct and working classifier at 67.21 % top-1 over all 50 000 validation images (Section 2). A path that skipped weights would not classify at that rate. The synthetic model's error reflects randomly initialized weights, which give a 4-bit palette no cluster structure to exploit. The online supplement (Table S23) reports the four encodings with weight blobs read from the compiled artifact. This measurement is one configuration at $B = 1$ on the M5 Pro and is not checked across generations, which matters because int8 decode is already at a different bound on the base M4.

What compression yields instead is energy savings: halving the weight bytes cuts energy per token by 20 % at $B = 1$ (27.8 → 22.4 mJ, about 9 pJ per avoided byte, an LPDDR-scale figure) and improves the ANE's tok/s/W at most batches (by up to 36 %, though it dips about 6 % at $B = 4$), at roughly 2× the fp16 relative error (about 5×10^{-2}). int8 does not change the GPU's throughput advantage at any batch. This element-rate ceiling is host-specific: on the more bandwidth-starved base M4 (h16) int8 does raise ANE decode throughput (1.3 to 2×), which shows that part's decode is at the DRAM-byte bound, not at this element-rate limit (Section 7). The accuracy cost is bounded: int8 holds the greedy token 92 % of the time versus the fp64 reference.

The energy axis is a measured crossover, not an immediate switch. The ANE has the higher tok/s/W at every batch up to $B = 16$ in both precisions, int8 by 2.3× at $B = 1$ (44.7 versus 19.3), 2.7× at $B = 8$ (100.3 versus 36.9), and still 1.75× at $B = 16$ (113.7 versus 65.1), with fp16 close behind (35.9 at $B = 1$, 106.2 at $B = 16$). The GPU's efficiency crosses above only at $B = 32$ (116.4 versus 105.4 int8, 88.0 fp16), where the ANE's package draw grows with the host CPU feeding $B \times 16.8$ MB of KV cache per step. (The GPU's dip at $B = 2$ is a power-state step: its rail jumps from 6.3 to a pinned 19.9 W before batching amortizes the full-power clocks.) Sweeping the KV cache from 128 to 2048 positions at $B = 1$, per-step latency is linear in context on both devices, and the fit depends on where the cache lives. Holding it resident in the engine's input buffers, the ANE runs at 7.75 ms plus 1.34 µs per position against the GPU's 4.74 ms plus 0.73 µs. Both intercepts recover the batch-invariant weight stream on an axis it was not fitted against, and the KV term reaches the weight term only at $s_{kv} \approx 5800$ on the ANE and ≈ 6500 on the GPU, so single-stream decode is weight-stream-bound across any context either part will hold. The ANE's KV path is 1.8× slower per position, the activation path of Section 3.2 and not the weight path, and its efficiency advantage is 2.6–2.0 across the sweep, with no crossover.

Re-feeding the cache to the engine on every token instead, as a graph input instead of a resident buffer, is what the roofline predicts it to be and is worth separating from the silicon. The cost is 0.51 ms and 0.92 W at $s_{kv} = 256$, rising to 4.38 ms and 2.18 W at 2048, where the transfer is 128 MB per token. It also triples the fitted per-position slope, to 3.49 µs, and it erases the ANE's efficiency advantage by $s_{kv} \approx 1300$. The signature is a falling engine rail, 1878–1428 mW, beside a rising package, 3.75–5.86 W: the engine idles while the host marshals. Held resident, the rail is flat at about 1.93 W and the package falls to 3.68 W at the same point. This host-feeding term also produces the batch crossover above; it is an implementation cost, not a property of the part.

Batched decode gives the GPU the throughput axis immediately but the energy axis only past $B \approx 16$. At small batch, on-device decode uses 2 to 3× less energy on the ANE. That advantage ends near $B \approx 16$ on the batch axis, and across context length it persists only while the caller keeps the cache resident.

Just Accepted

These magnitudes are for the one model measured. Decode here is weight-stream-bound because the 1.20 GB weight stream is about 70× the per-token KV traffic at $s_{kv} = 256$, and the throughput ordering follows the two devices' weight-stream rates. A model whose weight bytes dominate its KV bytes should order the same way. The energy crossover depends on the ratio of per-stream KV and host-feeding cost to the weight stream, which changes with width, depth, attention layout, and vocabulary (the 32 000-token vocabulary projection alone is 65.5 M of the 602.4 M streamed parameters), and we make no claim for where it falls on other models.

10 Discussion

The corrected roofline has a corollary the four sweeps confirm: the ANE's performance-per-watt advantage is regime-dependent, set by where a workload's arithmetic intensity falls relative to the two ridges. It is largest in the compute-bound, low-to-moderate-utilization regime, where the ANE is both faster and more efficient (Table 5, and ResNet-18 and MiniLM in Table 6). Four regimes favor the GPU. It has the higher raw throughput once a device is saturated, where it also takes a small conv perf/watt edge (Section 3.1), and it leads the standalone memory-bound regime, where the ANE's activation path runs about 10× below the GPU's unified-memory rate (Section 3.2). It also leads batched-transformer serving (Section 8) and, in the measured decoder, decode throughput at every batch (Section 9). Vision serving is the exception. ViT-B/16 and single-stream decode are energy-only exceptions: the ANE uses less energy but has the higher latency. The decode result holds across the measured context range while the KV cache stays resident. The two-ridge structure accounts for this split, and HeteroLLM [7] reports the same split independently on a different NPU (Section 11).

This corrects the community guidance [1, 13]. The 38-TOPS shorthand (an M4 figure) reduces a regime-dependent engine to one peak number. The measurements also contradict the guidance that the ANE is worthwhile mainly for big models: its largest advantages are in low-batch, latency- and energy-capped on-device inference. The claim that the ANE is more efficient is true in the compute-bound regime, narrows to a small GPU edge on conv at peak, and is false in the standalone memory-bound regime.

Fusion is the program-level control on arithmetic intensity, and so on regime: fusing operators into one compiled program raises effective AI and moves the operating point past the ridge (Section 4). An arithmetic-intensity-aware scheduler should place work by arithmetic intensity and target utilization. Operators that would otherwise be memory-bound on the ANE should be fused into a larger graph where possible. The choice is also constraint-dependent. Under a tight power or energy cap the ANE is preferable for every serving workload measured here, though on bare large-batch GEMM by only about 6 %. Without such a cap the GPU is preferable past each workload's crossover. CoreML exposes model-level placement through `computeUnits`, which can assign vision, convolution, and small-encoder models to the ANE. It does not expose the per-operator fusion control that moves a memory-bound operator across the ridge, nor does it report the per-engine energy measured here.

Replication. Re-measured over $N = 5$ independent cold-start runs, conv at peak resolves to a 5 % GPU edge and batched GEMM at $B = 64$ to a 6 % ANE edge (both Holm-significant), the large ANE margins (conv stack, attention, GEMM-compute) confirm, and both transformer crossovers are stable ($B^* = 2.23 \pm 0.02$ and 31.1 ± 1.3). The full protocol and per-claim table are in the online supplement (Table S17).

11 Related work

The analysis descends from the roofline model [30], extended to energy by Choi et al. [9] and to cache-aware, hierarchical, and instruction-level variants [10, 16, 32], with an efficiency-oriented framing for ML accelerators [29]. The ANE exposes two effective bandwidths, so the single-line form does not describe it; the multi-tier roofline [32] is what the data re-derive for this NPU (Section 4). We build the roofs empirically from microbenchmarks, not datasheet peaks [21], and the weight-path-versus-activation-path split is the energy-per-data-movement dataflow taxonomy [8] seen from outside the black box. Pagoda [24] builds dual time-and-energy rooflines on

Just Accepted

Jetson, and RooflineBench [3] scores on-device LLMs by operational intensity but is throughput-only and does not isolate the ANE.

Smartphone-AI benchmarking was popularized by AI-Benchmark [15] and standardized, with a power methodology, by MLPerf Inference [25], though that line reaches the ANE only through CoreML and cannot isolate it. Seshadri et al. [26] report throughput, utilization, and energy for Google's Edge TPU. Edge-AI rooflines have targeted Jetson and Coral [17], mobile-NPU LLM inference has been studied at the systems level and in GPU-NPU hybrids [23, 31], and sustained-load mobile energy has been measured across NPU and GPU including Apple devices [20, 28], none isolating the ANE rail.

HeteroLLM [7] characterizes a Qualcomm Hexagon NPU with an Adreno GPU and finds the same regime split with absolute power, compute-bound prefill to the NPU and memory-bound decode to the GPU; that an independent study reaches this on a different NPU is consistent with the device map reflecting a general property of NPU-versus-GPU economics, not of one vendor. For the energy methodology, idle-subtracted net energy has precedent in NPU-MCU work [11], and programmatic energy measurement on Apple silicon has been discussed but not resolved into a validated estimator [22].

A small body of concurrent work establishes direct or reverse-engineered ANE access: Orion [19], the maderix M4-ANE writeups [27], and the Asahi-Linux ANE driver [33]; neither direct access nor the observation that the engine can be energy-efficient is new here. Orion reports ANE throughput for LLM training and inference, with an ANE-versus-CPU speed comparison only and no power or roofline measurements. It independently finds CPU decode outpacing ANE decode on an M4 Max (283 versus 170 tok/s). Our decode loop compares the ANE against the GPU, not the CPU, but finds the same direction, a faster companion engine, on a different chip and access path (Section 9), and attributes the deficit to the ANE's weight-stream rate, not to `aneforge`. NPUMoE [2] runs mixture-of-experts on the ANE through CoreML, as do Apple's transformer guidance [1] and the community knowledge base [13]. Pagoda [24] and HeteroLLM [7] are the closest prior characterizations. Neither isolates the ANE rail in a per-engine roofline with idle-subtracted, package-level energy across the single-stream, batched, compute-saturation, and bandwidth regimes.

12 Conclusion

We presented a per-engine energy roofline characterization of the Apple Neural Engine against the Apple GPU and CPU on an M5 Pro, with the single-stream watt map reproduced on an M1 Max, an M2 Pro, and an M4.

The ANE's efficiency advantage is regime-dependent. It is faster and lower-power for compute-bound work at low-to-moderate utilization, including conv, attention, mid-size GEMM, ResNet-18, and the MiniLM encoder; on ViT-B/16 it is only the lower-energy device, by 1.5× against compiled MLX. The GPU has the higher throughput at saturation and at transformer-serving scale, a small perf/watt edge on conv at peak, and both higher speed and higher efficiency for standalone memory-bound operators. The ANE, as reachable today, has two effective bandwidths (about 24 GB/s standalone-activation, 112–143 GB/s weight-stream) and two corresponding ridges, the fast one below the GPU's. Given arithmetic intensity and dispatch path, the energy roofline predicts the lower-energy engine and bounds its energy from below. Its median error is 14 % near the roof, and it stays conservative elsewhere within its stated domain. ML-systems schedulers should place work by arithmetic intensity and utilization, not by operator name.

Limitations of this work

The saturation, bandwidth, serving, and decode magnitudes are from the one Apple M5 Pro (h17s); their structure reproduces across the four generations (Section 7), but the absolute multipliers are host- and thermal-dependent and should not be quoted for other parts, and the cross-generation reproductions are single- or dual-run point estimates, and are not replicated under the cold-start protocol. The end-to-end decode results are for one model

Just Accepted

configuration (one size, one 32 000-token vocabulary, synthetic weights), and their magnitudes and energy crossover are not claimed for other models (Section 9). The activation-path energy crossover (AI≈ 933) is computed from the two fitted constants and lies well beyond the measured range, so it is an extrapolation, not a measurement. Both power estimators are modeled SoC telemetry, not an external meter, though they cross-validate (Section 2, online supplement Table S1), and the ANE bandwidth ceiling is a path limit of `aneforge`'s single-program streaming, not a claim about the silicon DMA. This is a plugged-in, actively cooled host exposing only an OS thermal-pressure signal, so a fanless or battery-constrained device may behave differently. The access path uses undocumented but unprotected user-space runtime interfaces [6]. This work characterizes performance only and crosses no content-protection or security boundary. Despite these limitations, the contested small-margin verdicts are settled under the cold-start replicate protocol with formal inference attached (Section 10).

Data and code availability

The code, fail-soft driver, scripts, and committed result files behind every reported value are public at https://github.com/sbryngelson/ANEForge under a permissive open-source license, archived as release `artifact-v1` at https://doi.org/10.5281/zenodo.24210581 [4].

Acknowledgments

During preparation of this manuscript the author used a large language model (Anthropic Claude) for prose and grammar polishing. The author reviewed and edited all such text and takes responsibility for the content.

References

[1] Apple Machine Learning Research. 2022. Deploying Transformers on the Apple Neural Engine. https://machinelearning.apple.com/research/neural-engine-transformers. Apple Machine Learning Research article.

[2] Afsara Benazir and Felix Xiaozhu Lin. 2026. Efficient Mixture-of-Experts LLM Inference with Apple Silicon NPUs. arXiv:2604.18788 [cs.LG]

[3] Zhen Bi, Xueshu Chen, Luoyang Sun, Yuhang Yao, Qing Shen, Jungang Lou, and Cheng Deng. 2026. RooflineBench: A Benchmarking Framework for On-Device LLMs via Roofline Analysis. arXiv:2602.11506 [cs.LG]

[4] Spencer H. Bryngelson. 2026. ANEForge: CoreML-free programming of the Apple Neural Engine. https://github.com/sbryngelson/ANEForge. doi:10.5281/zenodo.24210581 Software artifact, release `artifact-v1`.

[5] Spencer H. Bryngelson. 2026. ANEForge: Python for Direct Computation on the Apple Neural Engine. *Software: Practice and Experience* (2026). doi:10.1002/spe.70124

[6] Spencer H. Bryngelson. 2026. The Apple Neural Engine: A Complete Guide. *arXiv preprint arXiv:2606.22283* (2026). doi:10.48550/arXiv.2606.22283

[7] Le Chen, Dahu Feng, Erhu Feng, Yingrui Wang, Rong Zhao, Yubin Xia, Pinjie Xu, and Haibo Chen. 2025. Characterizing Mobile SoC for Accelerating Heterogeneous LLM Inference. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles (SOSP '25)*. Seoul, Republic of Korea. doi:10.1145/3731569.3764808

[8] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 367–379. doi:10.1109/ISCA.2016.40

[9] Jee Whan Choi, Daniel Bedard, Robert Fowler, and Richard Vuduc. 2013. A Roofline Model of Energy. In *2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS)*. 661–672. doi:10.1109/IPDPS.2013.77

[10] Nan Ding and Samuel Williams. 2019. An Instruction Roofline Model for GPUs. In *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 7–18. doi:10.1109/PMBS49563.2019.00007

[11] Anastasios Fanariotis, Theofanis Orphanoudakis, and Vasilis Fotopoulos. 2025. Evaluating the Energy Efficiency of NPU-Accelerated Machine Learning Inference on Embedded Microcontrollers. arXiv:2509.17533

[12] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 243–254. doi:10.1109/ISCA.2016.30

[13] Matthijs Hollemans. 2020. The Neural Engine — What Do We Know About It? https://github.com/hollance/neural-engine. Community-maintained repository.

Just Accepted

[14] Paul Hübner, Andong Hu, Ivy Peng, and Stefano Markidis. 2025. Apple vs. Oranges: Evaluating the Apple Silicon M-Series SoCs for HPC Performance and Efficiency. arXiv:2502.05317 [cs.AR]

[15] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. 2019. AI Benchmark: All About Deep Learning on Smartphones in 2019. arXiv:1910.06663

[16] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. 2014. Cache-Aware Roofline Model: Upgrading the Loft. *IEEE Computer Architecture Letters* 13, 1 (2014), 21–24. doi:10.1109/L-CA.2013.6

[17] Rakshith Jayanth, Neelesh Gupta, and Viktor Prasanna. 2024. Benchmarking Edge AI Platforms for High-Performance ML Inference. arXiv:2409.14803

[18] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. 1–12. doi:10.1145/3079856.3080246

[19] Ramchand Kumaresan. 2026. Orion: Characterizing and Programming Apple's Neural Engine for LLM Training and Inference. arXiv:2603.06728 [cs.LG] Concurrent preprint.

[20] Stefanos Laskaridis, Kleomenis Kateveas, Lorenzo Minto, and Hamed Haddadi. 2024. MELTing Point: Mobile Evaluation of Language Transformers. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom)*. doi:10.1145/3636534.3690668

[21] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. 2015. Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation (PMBS 2014) (Lecture Notes in Computer Science, Vol. 8966)*. 129–148. doi:10.1007/978-3-319-17248-4_7

[22] ML.ENERGY / Zeus. 2025. [RFC] Programmatic Energy Consumption Measurement on Apple Silicon (macOS). https://github.com/ml-energy/zeus/issues/159. GitHub issue #159.

[23] Seungjae Moon, Junseo Cha, Hyunjun Park, and Joo-Young Kim. 2025. Hybe: GPU-NPU Hybrid System for Efficient LLM Inference with Million-Token Context Window. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*. 808–820. doi:10.1145/3695053.3731051

[24] S. K. Prashanthi, Kunal Kumar Sahoo, Amartya Ranjan Saikia, Pranav Gupta, Atharva Vinay Joshi, Priyanshu Pansari, and Yogesh Simmhan. 2025. Pagoda: An Energy and Time Roofline Study for DNN Workloads on Edge Accelerators. arXiv:2509.20189 [cs.DC]

[25] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, et al. 2020. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459. doi:10.1109/ISCA45697.2020.00045

[26] Kiran Seshadri, Berkin Akin, James Laudon, Ravi Narayanaswami, and Amir Yazdanbakhsh. 2022. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. In *2022 IEEE International Symposium on Workload Characterization (IISWC)*. 79–91. doi:10.1109/IISWC55918.2022.00017

[27] Manjeet Singh. 2026. Inside the M4 Apple Neural Engine, Part 1: Reverse Engineering. https://maderix.substack.com/p/inside-the-m4-apple-neural-engine; repo: https://github.com/maderix/ANE. Blog series and repository.

[28] Pranay Tummalapalli, Sahil Arayakandy, Ritam Pal, and Kautuk Kundan. 2026. LLM Inference at the Edge: Mobile, NPU, and GPU Performance Efficiency Trade-offs Under Sustained Load. arXiv:2603.23640 [cs.DC]

[29] Marian Verhelst, Luca Benini, and Naveen Verma. 2025. How to Keep Pushing ML Accelerator Performance? Know Your Rooflines! *IEEE Journal of Solid-State Circuits* (2025). doi:10.1109/JSSC.2025.3553765

[30] Samuel Williams, Andrew Waterman, and David A. Patterson. 2009. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 52, 4 (2009), 65–76. doi:10.1145/1498765.1498785

[31] Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. 2025. Fast On-device LLM Inference with NPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 1*. doi:10.1145/3669940.3707239

[32] Charlene Yang, Thorsten Kurth, and Samuel Williams. 2020. Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system. *Concurrency and Computation: Practice and Experience* 32, 20 (2020), e5547. doi:10.1002/cpe.5547

Just Accepted

[33] Eileen Yoon. 2022. ane: Reverse-Engineered Linux Driver for the Apple Neural Engine (+anecc). https://github.com/eiln/ane; https://github.com/eiln/anecc. Asahi Linux driver and compiler repositories.

Received 24 June 2026; revised 14 October 2026; accepted 13 November 2026

Just Accepted