HETEROENGINE: CONCURRENT TRAINING ON APPLE SILICON'S GPU AND NEURAL ENGINE

Anonymous authors
Paper under double-blind review

ABSTRACT

Apple Silicon uses a system-on-chip (SoC) architecture in which the CPU, GPU, and Apple Neural Engine (ANE) share unified memory. The two accelerators, the GPU and the ANE, have opposite characteristics: the GPU offers high throughput but consumes more energy, whereas the ANE consumes less energy but offers lower throughput. Moreover, because the two engines share unified memory, they can access the same data and weights without the inter-device transfers required by discrete accelerators with separate device memory; this favors using them together. Standard training frameworks, however, backpropagate on the GPU alone, leaving the ANE idle throughout training. We therefore aim to use both engines concurrently, each compensating for the other's weakness—the GPU's high energy consumption and the ANE's low throughput—and thereby to exceed GPU-only training in both throughput and energy efficiency. In practice, Apple's public software exposes the ANE only through Core ML, a framework for executing trained models that provides no automatic differentiation and thus cannot compute gradients. Prior ANE training work consequently relied on Apple's unpublished internal interfaces. We present HeteroEngine, which derives the backward pass by hand and implements it as a Core ML program, thereby training on the GPU and the ANE concurrently through Apple's official APIs alone, without any such private interface. HeteroEngine partitions each training batch into microbatches and distributes them between the GPU and the ANE. Starting from the same model parameters, both engines concurrently perform the forward and backward passes of the entire model on their assigned microbatches. HeteroEngine then averages the gradients across all microbatches and applies a single optimizer update. When training GPT-2 124M on an M4 Pro, HeteroEngine achieves $1.419\times$ the throughput of GPU-only training at 30.9% less energy per token; relative to ANE-only training, it achieves more than twice the throughput at higher energy per token. Whereas ANE-only training yields a small but statistically significant increase in perplexity, HeteroEngine matches or improves on GPU-only quality: at equal token counts its perplexity is equivalent to GPU-only perplexity on Shakespeare and WikiText-103; on WikiText-103 it is slightly lower after 18.06M tokens at equal token counts and substantially lower at equal training time; and on a Mac mini M2 the same program passes the same test and reproduces the direction of both gains. HeteroEngine thus uses the otherwise idle ANE alongside the GPU, raising both throughput and energy efficiency without loss of quality, and, built on Apple's official APIs alone, runs on a standard macOS installation.

1 INTRODUCTION

Apple Silicon uses a system-on-chip (SoC) architecture in which the CPU, GPU, and Apple Neural Engine (ANE) share unified memory. The two accelerators, the GPU and the ANE, have opposite characteristics. The GPU is a general-purpose parallel processor that offers high throughput but consumes more energy; the ANE is an accelerator dedicated to neural-network computation that consumes less energy but offers lower throughput. Moreover, because the two engines share unified memory, they can access the same data and weights without the inter-device transfers required by discrete accelerators with separate device memory, so both engines can be used together without adding a device. Standard training frameworks, however, backpropagate on the GPU alone, leaving the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

ANE idle throughout training. We therefore aim to use both engines concurrently, each compensating for the other's weakness—the GPU's high energy consumption and the ANE's low throughput—and thereby to exceed GPU-only training in both throughput and energy efficiency. Shared memory alone, though, provides no shared training interface.

MLX (Hannun et al., 2023) and PyTorch MPS (PyTorch, 2022) expose GPU backpropagation. In Apple's public software, by contrast, the only route to the ANE is Core ML. Public Core ML supports model execution and limited training; its ML Program format has mutable state but no training loop or automatic differentiation (Apple Inc., 2024b;c). A system that trains on the ANE must therefore supply the backward pass itself, install updated weights, and retrieve complete gradients on every update. Each of these carries a cost. Weights embedded as program constants require recompilation through this public path whenever they change, while ordinary tensor inputs incur substantial ingestion costs. Even an executable backward pass needs separate numerical validation, since matching the scalar loss does not guarantee an accurate gradient. Prior ANE training work consequently relied on interfaces that Apple has not published: Orion (Kumaresan, 2026) and maderix/ANE (maderix, 2026) use private ANE client interfaces, and ANEForge (Bryngelson, 2026b) uses the private e5rt runtime.

HeteroEngine meets these requirements through Apple's official APIs alone. We derive the backward pass by hand and implement it so that one Core ML program computes the complete forward and backward passes of the entire model for one microbatch. The trainable matrices reside in mutable `MLState` tensors, avoiding recompilation and ingestion costs, and caller-owned output backings write gradients directly into GPU-accessible optimizer buffers. Each update assigns Q microbatches to the ANE and K to the GPU, with $M = Q + K$; both engines start from casts of the same fp32 parameters—fp16 on the ANE, bfloat16 on the GPU. Their gradients are averaged and applied in one Adam update (Kingma & Ba, 2015), so the two engines compute gradients concurrently while the update boundary remains synchronous.

We make three contributions:

1. We show that training on the ANE does not require private interfaces. HeteroEngine computes gradients for all 72 dense transformer-block matrices of GPT-2 124M (Radford et al., 2019) (84.93M trainable parameters) on the ANE through Apple's official APIs alone and trains concurrently with the GPU; its concurrent execution reproduces a sequential reference bit for bit, and host timestamps verify that the two engines' work overlaps (§3, §4).
2. We present a validation procedure for backward passes derived by hand for an accelerator without automatic differentiation. Because an unchanged scalar loss does not guarantee an accurate gradient, we compare gradients directly with an fp32 reference and localize errors with GPU controls that mirror the partition. The procedure located an error in our first softmax backward that the scalar loss did not reveal. With the established shifted-softmax formula, gradient error falls by more than an order of magnitude, and all five tested ANE allocations pass a prespecified quality bound (§5, §6.2).
3. We show empirically that using the GPU and the ANE concurrently raises both throughput and energy efficiency over GPU-only training while preserving training quality. On an M4 Pro, against a GPU control selected in a separate batch-size grid, throughput rises $1.419\times$ and CPU+GPU+ANE energy per token falls by 30.9%; on Shakespeare and WikiText-103, quality at equal token counts is equivalent and, on WikiText-103, perplexity at equal training time is lower; and on a Mac mini M2 the same program passes the same quality test and a preregistered panel reproduces the direction of both gains (§6, Appendix C.1).

2 Background and Related Work

Training on the ANE. The initial maderix/ANE release (maderix, 2026) and Orion (Kumaresan, 2026) run the forward pass and activation gradients on the ANE while the CPU computes weight gradients and Adam; maderix/ANE recompiles its programs when the baked weights change, and Orion patches the compiled weight files and reloads the program. ANEForge (Bryngelson, 2026b) runs the forward pass, backward pass, and Adam update on the ANE, with resident optimizer state and a half-precision-stable softmax–cross-entropy backward. All three use private interfaces. HeteroEngine keeps the weights in public Core ML state, computes all 72 weight gradients on the

ANE, and combines them with concurrently computed GPU gradients before one Adam update on the GPU.

Heterogeneous training and execution. QSync selects operator precisions for synchronous training across heterogeneous GPUs, balancing accuracy, memory, and throughput (Zhao et al., 2024). Mandheling distributes mixed-precision training operators across CPU and DSP (Xu et al., 2022). FusionML row-splits linear-layer matrix multiplications between CPU and GPU for prefill and finds that Core ML dispatch to the ANE pays off only for batch-scale work (Mohite, 2026). HeteroEngine instead splits at microbatch granularity, so each ANE call carries a complete forward and backward pass. KAITIAN combines NVIDIA GPUs and Cambricon MLUs and reports 83.4% accuracy against 85.3–85.5% for homogeneous runs, without seed-level uncertainty (Lin et al., 2025); we test quality with paired controls and a prespecified equivalence margin.

Reducing training computation. PockEngine sparsifies the training graph (Zhu et al., 2023), FwdLLM replaces backpropagation with perturbed forward passes (Xu et al., 2024), and LoRA and ReLoRA train low-rank updates (Hu et al., 2022; Lialin et al., 2024). HeteroEngine instead computes full gradients for all 72 block matrices and spreads that work across the GPU and ANE.

Numerical accuracy in reduced precision. Mixed-precision training combines low-precision arithmetic, fp32 master weights, and loss scaling (Micikevicius et al., 2018). These measures address range and update resolution but do not make algebraically equivalent expressions equally accurate. Blanchard et al. (2021) analyze log-sum-exp and softmax evaluation and recommend shifted normalization over the division-free formulation that reconstructs probabilities from log-sum-exp. We apply that established result to the ANE head backward and measure its effect on gradients and training quality.

3 System Architecture

HeteroEngine trains the 72 dense matrices of GPT-2 124M's twelve transformer blocks ($W_q, W_k, W_v, W_o, W_1, W_2$; 84,934,656 parameters). Token and position embeddings, biases, and LayerNorm parameters stay frozen. Public Core ML was built to execute trained models, so training these matrices through it faces the obstacles in Table 1; this section describes the design that addresses each.

One parameter version, one update. At update t, both engines use execution weights cast from the same fp32 master parameters, fp16 on the ANE and bfloat16 on the GPU (Figure 1). Each assigned microbatch contains $S = 256$ tokens, and both engines compute gradients of the mean token loss. The ANE exports $\widetilde{g}^{\mathrm{ANE}}$ scaled by $\lambda = 1024$, while GPU gradients g^{GPU} are unscaled. The update gradient is

$$g_t = \frac{1}{M}\left(\frac{1}{\lambda}\sum_{i=1}^{Q}\widetilde{g}_{t,i}^{\mathrm{ANE}} + \sum_{j=1}^{K} g_{t,j}^{\mathrm{GPU}}\right). \tag{1}$$

The reduction removes the ANE loss scale once and averages the microbatch gradients once. One Adam update then advances the masters and both moments.

Mutable weights in an inference program. A single MIL program computes the forward pass, cross-entropy loss, and hand-derived backward pass for all 72 matrices; one public Core ML prediction

Table 1: **Obstacles to training through public Core ML and the corresponding design.**

Obstacle	HeteroEngine design
No autodiff for ML Programs	Hand-derived backward in one MIL program
Constant weights need recompiling	Matrices in mutable `MLState`, rewritten by the host
The compiler limits the state count	72 matrices packed into 24 states
Gradients return in new output arrays	Output backings write them into MLX buffers
Two engines, one parameter version	Fixed-order join, then one Adam update (§4)

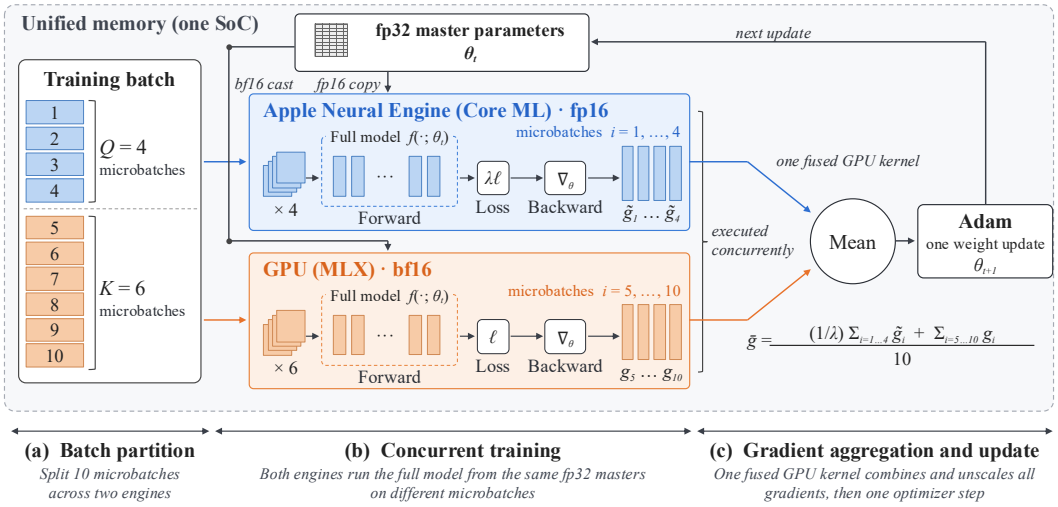

Figure 1: **One synchronous update across two engines** (Q=4, K=6): (a) batch partition, (b) concurrent forward and backward passes from the same fp32 masters, (c) gradient averaging, Eq. (1), and one Adam update.

executes a complete microbatch. Embedded weight constants would require recompilation after every update, and ordinary tensor inputs incur substantial ingestion costs, so the matrices live in mutable `MLState` tensors that the host rewrites between calls (Appendix C). We use ANE-compatible tensor layouts (Apple Machine Learning Research, 2022; Bryngelson, 2026a). Because the compiler limits the number of states, each block packs W_q, W_k, W_v, W_o, W_1 into a $[6144, 768]$ slab and stores W_2 in a $[768, 3072]$ tensor; in-graph slices and transposes reuse these 24 states in both passes, and the state constraints also hold on a second chip (Appendix C.1). The program reads state but leaves its mutation to the host, avoiding the fallback observed with in-graph updates. It runs on `CPUAndNeuralEngine` execution units, excluding GPU fallback; its compiled plan marks all 1,586 operations ANE-preferred, and power rails check ANE activity separately.

Native dispatch and gradient exchange. The driver calls public Core ML APIs through PyObjC. Caller-provided output backings (Apple Inc., 2026b) wrap MLX-owned unified-memory buffers, so each ANE prediction writes its 72 gradients into a separate optimizer buffer bank without passing them through the coremltools return path. On the earlier head program (§5), which shares this driver, native dispatch reduces prediction latency from 106.9 to 48.2 ms and state installation from 21 to 3.6 ms (Appendix C). A fused GPU kernel combines and unscales the banked gradients, updates the fp32 parameters and Adam moments, and produces the next fp16 weight image, which the host copies into the state view before the next ANE prediction. At $K = 0$, the GPU still performs reduction and optimization: "all-ANE" covers the forward and backward passes.

4 Execution Model and Validation

The main thread enqueues GPU work asynchronously and signals a persistent Core ML worker to install the current weight image and issue Q serial predictions. No state write overlaps an ANE read. Both legs finish before the fixed-order gradient reduction and Adam update, so every microbatch uses parameter version t. ANE assignments rotate through the predefined batch sequence.

Let W denote the state-write time, P the ANE microbatch time, $T_{\mathrm{GPU}}(K)$ the GPU leg's time, and U the post-join reduction and optimizer time. With noncontending engines in steady state,

$$T(Q, K) = \max(W + QP,\, T_{\mathrm{GPU}}(K)) + U. \tag{2}$$

Throughput is $SM/T(Q, K)$; we measure $T_{\mathrm{GPU}}(K)$ directly because batching need not scale linearly with K (Appendix A gives an idealized capacity bound).

A ten-update $Q = 2, K = 2$ check produces identical hashes of fp32 parameters, both Adam moments, and the fp16 weight image under concurrent execution, sequential execution, and an independent reference driver. This checks the implemented update against its sequential reference; arithmetic accuracy is evaluated separately.

Host timestamps also establish overlap. Across three $Q = 4, K = 6$ grid runs, the median ANE prediction phase (204.5–208.8 ms) and GPU completion wait (205.8–211.8 ms) both fall within complete updates averaging 229.1–233.7 ms, so the two legs overlap inside each update (Appendix A).

5 Gradient Fidelity

Loss scaling protects small gradients but cannot repair an inaccurate probability calculation. For logits $\ell = Eh$, let $m = \max_v \ell_v$ and $Z = \sum_v \exp(\ell_v - m)$. A backward that reconstructs probabilities from the fp16 log-normalizer $a = \mathrm{fp16}(m + \log Z)$ obtains $\widehat{p}_v = \exp(\ell_v - a)$. If a has rounding error ϵ, then, holding other operations exact, $\sum_v \widehat{p}_v = e^{-\epsilon}$. One scalar rounding error perturbs every vocabulary probability.

HeteroEngine instead forms scaled probabilities by the established shifted-softmax formula (Blanchard et al., 2021):

$$p_v^{\mathrm{sc}} = \exp(\ell_v - m)\left(\frac{C}{Z}\right), \qquad C = 4096. \tag{3}$$

The implementation uses a common maximum and denominator across four vocabulary chunks. With frozen output matrix E and target row E_y, the loss-scaled adjoint is

$$\widetilde{\nabla_h L} = \frac{\lambda}{SC}\left(E^\top p^{\mathrm{sc}} - CE_y\right) = \frac{\lambda}{S}\left(E^\top p - E_y\right) \tag{4}$$

in exact arithmetic, where $p_v = \exp(\ell_v - m)/Z$. The head scale C cancels here; the loss scale λ is removed once in Equation 1.

Changing only this backward expression leaves scalar loss, fp16 execution, loss scaling, and Adam unchanged. Against a common fp32 GPU reference, pooled relative gradient error falls from 0.3639 to 0.0189 at initialization and from 0.3076 to 0.0219 at a shared 300-update checkpoint. Each probe uses 16 matched inputs and 4,096 coordinates per matrix (Appendix B). A probability-mass error δ adds $\delta E^\top p$ to $E^\top p - E_y$; its component $\delta\mu$ along the mean output embedding μ shifts every position's hidden-state gradient in a common direction. The corrected backward also closes the training-quality gap: under the primary Shakespeare protocol, the hybrid's perplexity difference from GPU training falls from $+3.02\%$ to $+0.06\%$, and every seed improves (Figure 6 in Appendix B; §6.2).

A fixed-weight comparison then tests the corrected hybrid against GPU-only controls under the actual $6 + 4$ partition. At two 1,008-update WikiText checkpoints and six shared input sets, median full-gradient relative L_2 error is 10.33% for GPU batch 10, 10.49% for GPU-only $6 + 4$, and 7.26% for GPU 6 plus ANE 4; the hybrid is closer to the fp32 reference in all 12 combinations (median paired reduction against GPU-only $6 + 4$: 28.85%). Partitioning alone changes error beyond the prespecified $\pm 10\%$ band in three combinations (Appendix B). A stabilized GPU fp16 path reduces median error further, to 1.12%, so the ANE's advantage is over the tested bf16 paths, not over GPU execution in general (Appendix B).

6 Experiments

6.1 Setup and comparisons

We use an M4 Pro MacBook Pro (Mac16,7; 14-core CPU, 20-core GPU, 16-core ANE, 24 GB unified memory), running macOS 26.6.2, MLX 0.32.0, and PyObjC 12.2.2. Resource measurements use AC power with the unchanged High Power setting (`powermode 2`). All arms retain the trainable scope in §3 and start from the same archived fp16 model export, upcasting trainable masters to fp32. Unless specified otherwise, each microbatch has 256 tokens and Adam uses learning rate 5×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$; training uses neither weight decay nor dropout (Appendix C).

Table 2: **GPU-only control for each question.**

Question	GPU-only control	Held equal
Quality at equal tokens (§6.2, 6.4)	Same M as the hybrid	Tokens and Adam updates
Throughput, energy (§6.3, App. B)	M=24 (grid best); M=10; fp16 M=20	Measurement protocol
Fixed training time (§6.4)	M=10 and M=24	450 s of update time
Gradient accuracy (§5)	GPU-only 6+4; stabilized fp16	Checkpoints and inputs

Table 3: **Training quality by allocation.** Paired differences from the same-M GPU control after 300 Shakespeare updates, five seeds. The $Q = 4, K = 6$ row is the primary panel; the other allocations form the allocation sweep. The three $M = 4$ allocations share the same five GPU control runs. CI denotes pointwise paired 90% intervals.

Allocation	M	f	PPL difference, CI (%)	1% equivalence
$Q = 1, K = 7$	8	$1/8$	$+0.001\ [-0.164, +0.167]$	Pass
$Q = 1, K = 3$	4	$1/4$	$-0.004\ [-0.250, +0.242]$	Pass
$Q = 4, K = 6$	10	$2/5$	$+0.061\ [-0.217, +0.340]$	Pass
$Q = 2, K = 2$	4	$1/2$	$+0.190\ [-0.145, +0.526]$	Pass
$Q = 4, K = 0$	4	1	$+0.606\ [+0.372, +0.840]$	Pass

By default, GPU forward and backward passes use bfloat16, with fp32 masters and logits promoted to fp32 before the loss; the common evaluator also uses this arithmetic.

We encode the Tiny Shakespeare sample distributed with `char-rnn` (Karpathy, 2015) using GPT-2 BPE, obtaining 338,025 tokens: the first 304,222 for training and the final 33,803 held out. The primary quality panel uses 300 updates at $M = 10$, five fresh seeds 101–105, constant learning rate, and matched training-batch sequences; seeds vary the sampled training windows, not the pretrained parameters. A common GPU evaluator scores all final states on the same 30 held-out windows, and training seeds are the units of the paired analysis.

For seed s, define $d_s = \mathrm{CE}_{H,s} - \mathrm{CE}_{G,s}$. We report $\bar{d}$ and the paired relative perplexity gap, $100(\exp(\bar{d}) - 1)$. The prespecified equivalence test requires both ends of the paired 90% Student-t interval to lie inside $\pm \ln(1.01)$. Timing ratios use separately specified intervals. Appendix A gives the full protocol definitions.

Each comparison uses the GPU-only control that matches its question (Table 2). The resource and WikiText-103 GPU-only controls omit Core ML state, ANE gradient banks and workers, and per-update fp16 shadow conversion; a 300-update state-hash check confirms that this leaves the control's parameters and optimizer states unchanged (Appendix A). The GPU baseline evaluates its microbatches together using batched vector–Jacobian products (VJPs); complete GPU-leg tests on MLX 0.32.0 found no speed advantage from fused SDPA (Appendix C). All ANE comparisons use this GPU graph.

6.2 Training quality across ANE allocations

At $Q = 4, K = 6$, the five-seed panel gives a $+0.061\%$ perplexity difference from GPU $M = 10$ (90% CI $[-0.217\%, +0.340\%]$), passing the prespecified 1% equivalence test.

The allocation sweep uses the same five seeds at $f = Q/M \in \{1/8, 1/4, 1/2, 1\}$, each against a GPU control at the same M. Together with $f = 2/5$, all five allocations pass the prespecified pointwise 1% equivalence test (Table 3). All-ANE retains a $+0.606\%$ residual whose 90% interval $[+0.372\%, +0.840\%]$ excludes zero but lies within the margin. On a Mac mini M2, the same HeteroEngine program, recompiled on the device, gives a $+0.0022\%$ difference from GPU training at Q=2, K=2 (300 Shakespeare updates, seeds 33–37; 90% CI $[-0.097\%, +0.102\%]$), passing the same test (Appendix C.1).

Table 4: **Operating points on the M4 Pro.** Throughput and CPU+GPU+ANE energy come from the same five metered measurement blocks. The GPU $M = 24$ comparator was selected in the preceding throughput grid. Energy entries are means ± standard deviations across blocks.

Configuration	Tokens/update	tok/s	J/1k tokens
GPU $M = 24$	6144	7,240	4.184 ± 0.030
GPU $M = 10$	2560	6,947	4.124 ± 0.026
GPU $M = 4$	1024	6,176	4.214 ± 0.037
Hybrid $Q = 4, K = 6$	2560	10,271	2.890 ± 0.011
ANE $Q = 4, K = 0$	1024	4,486	1.502 ± 0.085

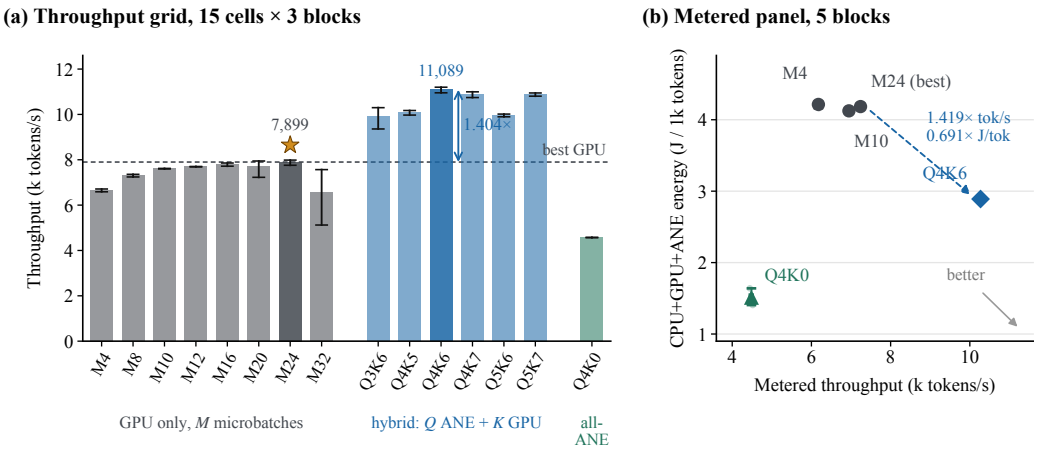

Figure 2: **Throughput and CPU+GPU+ANE energy.** (a) All 15 grid cells; star: selected GPU. (b) Metered panel; throughput is 7–8% below the grid for every arm. Whiskers: observed block ranges.

6.3 Throughput and Energy

The throughput grid spans eight GPU batch sizes, six hybrid allocations, and all-ANE execution in three measurement blocks. Each cell runs 120 updates, discarding the first 20 as warmup; configuration order is fixed before each block. GPU $M = 24$ is the fastest GPU cell at 7,899 tok/s, compared with 11,089 tok/s for $Q = 4, K = 6$.

A separate five-block metered panel fixes these configurations before collection. The hybrid delivers $1.419\times$ throughput relative to GPU $M = 24$ (95% CI $[1.415, 1.422]$), with 30.9% less CPU+GPU+ANE energy per token (ratio 0.691, 95% CI $[0.685, 0.697]$). Against batch-matched GPU $M = 10$, throughput is $1.478\times$ (95% CI $[1.473, 1.484]$; Table 4, Figure 2).

Energy per token integrates the Combined CPU+GPU+ANE power from `powermetrics` over the full timed training bracket, including update and synchronization time, and divides by completed tokens; wall-plug and DRAM energy are outside its scope. All-ANE uses still less energy, 1.502 ± 0.085 J/1k tokens, at 4,486 tok/s: $0.359\times$ the selected GPU's energy per token and $0.620\times$ its throughput. Ratios use paired log contrasts across the five blocks, not individual power samples (Appendix A).

In one sustained-load trace per arm (44.81 and 33.76 minutes after warmup), the hybrid delivers $1.390\times$ throughput with 28.7% less rail energy per token, and throughput drifts by only -0.24% and $+0.27\%$ from the first to the final five minutes (Appendix A, Figure 5). The hybrid/GPU $M = 24$ ratio is $1.404\times$ in the unmetered grid; a sampler on/off crossover estimates the sampler's effect on this ratio at $+0.97\%$ (90% CI $[-0.11\%, +2.06\%]$; Appendix A). On the M2, a preregistered one-seed panel shows both gains against batch-matched GPU controls: $1.49\times$ and $1.59\times$ throughput with energy-per-token ratios 0.82 and 0.78 at Q=4, K=6 and Q=2, K=2. The M2's smaller energy gain

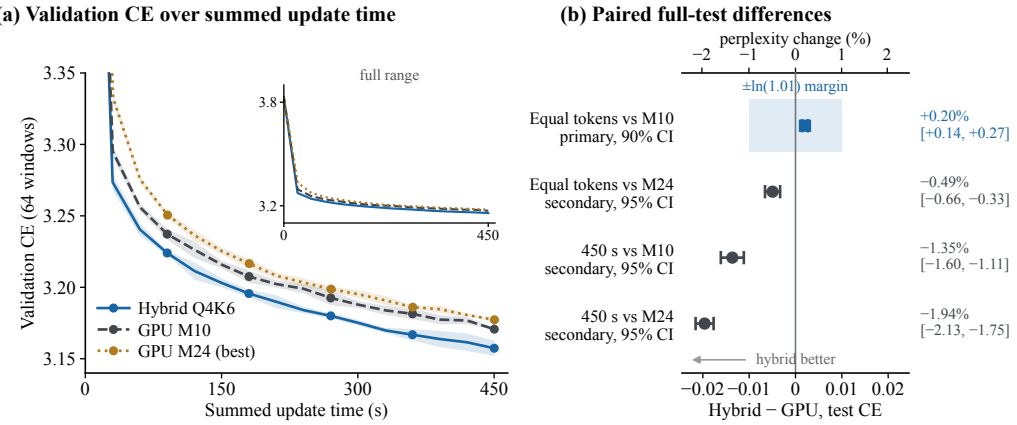

Figure 3: **WikiText-103 quality at matched tokens and fixed time** (bf16 evaluation). (a) Validation CE, pointwise 95% bands; inset: full range. (b) Paired test contrasts: 90% interval and equivalence band for the primary contrast, 95% intervals for secondary contrasts; labels: perplexity change [CI].

is mostly due to ANE-rail energy per token, which is 0.44 of the GPU rail's on the M2 and 0.17 on the M4 Pro (post hoc; Appendix C.1).

6.4 Matched-token and fixed-time training on WikiText-103

Shakespeare's small training split is repeatedly revisited within the 300-update panel, so we test longer training on a fixed 20M-token prefix of the official WikiText-103 raw training split (Merity et al., 2017), encoded with GPT-2 BPE; the model, trainable matrices, and learning rate are unchanged (Appendix A).

On seeds 201–205, we compare $Q = 4, K = 6$, pure GPU $M = 10$, and the previously selected GPU $M = 24$. Each run saves a checkpoint after 2,580,480 training tokens and stops at the first completed update that crosses 450 seconds of accumulated update time, which includes synchronization, Adam, and compilation within updates but excludes offline Core ML compilation, setup, evaluation, and checkpoint writes. A common GPU evaluator scores all 287,488 targets in the 1,123 complete official test blocks. The primary comparison uses GPU $M = 10$, matching both tokens and the 1,008 Adam updates. Figure 3(a) shows five-seed validation means on 64 common windows at mean actual crossing times for 30-second targets.

At 2.58M tokens and 1,008 updates, the hybrid's perplexity difference from GPU $M = 10$ is $+0.2041\%$ (90% CI $[+0.1411\%, +0.2672\%]$), passing the prespecified 1% equivalence test, although all five paired differences are positive. Against GPU $M = 24$ at the same token count, its perplexity is 0.49% lower (95% CI $[0.33\%, 0.66\%]$ reduction).

Fixed training time. Under the 450-second budget, the hybrid has 1.943% lower test perplexity than GPU $M = 24$ (95% CI $[1.752\%, 2.133\%]$ reduction) and 1.354% lower than GPU $M = 10$ (95% CI $[1.109\%, 1.599\%]$ reduction). The hybrid processes $1.405\times$ as many tokens per second as GPU $M = 24$ (95% CI $[1.395, 1.415]$). The $M = 24$ comparisons allow different batch sizes and optimizer-update cadences, and the $M = 10$ control matches both tokens and updates only at the equal-token checkpoint above.

Longer matched-token training and evaluation precision. On new seeds 301–305, we extend $Q = 4, K = 6$ and GPU $M = 10$ to 7,056 updates, or 18,063,360 sampled tokens, with the same constant learning rate and matched within-seed token sequences. Both arms save checkpoints at 1,008, 2,016, 4,032, and 7,056 updates, and every checkpoint is scored on all 1,123 test blocks by common bf16 and fp32 evaluators; the fp32 evaluation was added in a follow-up plan (Appendix A). The final gap passes the prespecified 1% equivalence test under both evaluators (90% CI $[-0.490\%, -0.413\%]$ with bf16; $[-0.228\%, -0.179\%]$ with fp32). At that endpoint the hybrid

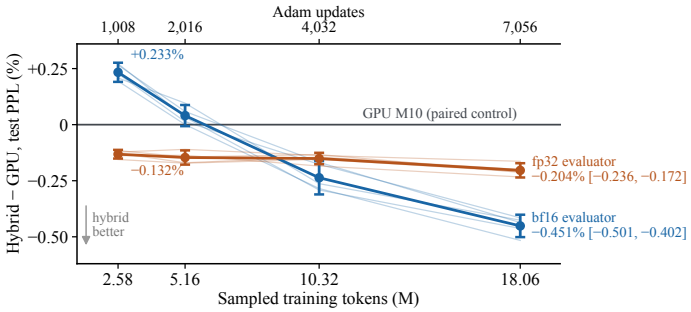

Figure 4: **Long-training quality under common bf16 and fp32 evaluation.** Thin lines: seeds; points and bars: paired mean contrasts and 95% CIs as perplexity change; labels: first and last checkpoints.

has lower perplexity under both: -0.451% with bf16 (95% CI $[-0.501\%, -0.402\%]$) and -0.204% with fp32 ($[-0.236\%, -0.172\%]$). The initial gap, however, is $+0.233\%$ with bf16 and -0.132% with fp32. Thus the final advantage holds under both tested evaluators, but an early deficit followed by a crossover occurs only with bf16 (Figure 4; Appendix A).

7 Discussion

The fastest and lowest-energy allocations differ: overlapping GPU and ANE work maximizes throughput, while all-ANE execution minimizes measured energy per token among the tested configurations. The hybrid reaches 88% of the summed GPU $M = 24$ and all-ANE throughputs; the gap is mainly due to the unoverlapped update U of Equation 2 and slower GPU microbatches (Appendix A). On the M4 Pro, the saving comes from higher throughput at similar CPU+GPU+ANE power (29.7 and 30.3 W, from Table 4).

The ANE path improves gradient accuracy only relative to the tested bf16 paths; a stabilized GPU fp16 path is more accurate still, and short-run quality rankings change with evaluation precision (Appendix B). The long-run endpoint favors the hybrid under both evaluators, but changing engines also changes kernels, operation order, and backward implementation, so we do not attribute that advantage to fp16 precision alone.

The study covers one M4 Pro, one M2 (quality at $Q{=}2, K{=}2$ and a one-seed energy panel), the 72 block matrices of §3, and 256-token sequences; the earlier M2 execution in Appendix C.1 used the log-normalizer head variant. Quality panels have five training seeds each. A later panel gives the hybrid $1.44\times$ throughput and $0.69\times$ energy per token against the stabilized fp16 GPU path (more accurate gradients, slower than bf16), and $1.37\times$ and $0.71\times$ against bf16 (Appendix B). The prespecified 1% margin is an acceptance criterion, not equality: all-ANE retains a $+0.606\%$ residual whose 90% interval excludes zero. Matched-token testing reaches 18.06M sampled tokens and fixed-time testing 450 seconds; neither establishes quality for larger models or full pretraining. Equation 2 ties the hybrid's gain to the number of microbatches Q the ANE completes while the GPU runs. The M6 pairs two Neural Engines, announced to double peak ANE compute (Apple Inc., 2026a); if Core ML exposes that throughput, the ANE could take twice as many microbatches per update, and we expect both gains over GPU-only training to grow. We have not measured this.

8 Conclusion

HeteroEngine trains concurrently on the GPU and ANE through Apple's official APIs, delivering $1.419\times$ throughput with 30.9% less SoC rail energy per token than the selected bf16 GPU control. Quality meets the prespecified bound across five ANE allocations and through 18.06M WikiText-103 tokens. At the longer matched-token endpoint, perplexity is lower under both common evaluators, and under a fixed training-time budget the throughput gain yields lower test perplexity.

REPRODUCIBILITY STATEMENT

The artifact index maps every reported campaign to its program builders, native drivers, frozen plans, raw logs, paired endpoint arrays, and analysis code. Records 317–334 document the reported experiments and numerical controls; records 283–316 cover the log-normalizer head variant and the earlier M2 execution. Appendix A specifies inputs, evaluation, statistical units, and scoped execution times. Appendix B defines the numerical probes, and Appendix C specifies the implementation and its checks. Source, compiled-model, parameter, token-array, and evaluator fingerprints distinguish versions. The training path uses public MLX and Core ML APIs on stock macOS. All reported training runs use the M4 Pro except the M2 runs in Appendix C.1.

AI USE STATEMENT

Large language models were used in five capacities in this work: (i) *method implementation*—the training harnesses, native driver, and measurement scripts were written with LLM coding assistance under the authors' direction; (ii) *interpretation of results*—as a numerical-analysis aid in discussing measured effects; (iii) *figure generation*, via authored plotting scripts; (iv) *translation*, the authors' working language not being English; and (v) *literature search and bibliography verification*. The system design, the experimental designs and preregistrations, and all measurements are the authors' own; every quantitative claim was verified by the authors against the released measurement records, and the authors take full responsibility for the content of this paper.

REFERENCES

Apple Inc. Apple unveils M2, taking the breakthrough performance and capabilities of M1 even further, 2022. URL https://www.apple.com/newsroom/2022/06/apple-unveils-m2-with-breakthrough-performance-and-capabilities/. Press release, June 6, 2022; accessed September 23, 2026.

Apple Inc. Apple introduces M4 Pro and M4 Max, 2024a. URL https://www.apple.com/newsroom/2024/10/apple-introduces-m4-pro-and-m4-max/. Press release, October 30, 2024; accessed September 23, 2026.

Apple Inc. Comparing ML programs and neural networks, 2024b. URL https://apple.github.io/coremltools/docs-guides/source/comparing-ml-programs-and-neural-networks.html. Guide to Core ML Tools, accessed September 6, 2026.

Apple Inc. Stateful models, 2024c. URL https://apple.github.io/coremltools/docs-guides/source/stateful-models.html. Guide to Core ML Tools, accessed September 6, 2026.

Apple Inc. Apple introduces M6 and M5 Ultra for a big leap in performance and AI compute, 2026a. URL https://www.apple.com/newsroom/2026/08/apple-introduces-m6-and-m5-ultra-for-a-big-leap-in-performance-and-ai-compute/. Press release, August 25, 2026; accessed September 22, 2026.

Apple Inc. MLPredictionOptions.outputBackings, 2026b. URL https://developer.apple.com/documentation/coreml/mlpredictionoptions/outputbackings. Core ML API reference, accessed September 6, 2026.

Apple Machine Learning Research. Deploying transformers on the Apple Neural Engine, 2022. URL https://machinelearning.apple.com/research/neural-engine-transformers.

Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330, 2021. doi: 10.1093/imanum/draa038. URL https://academic.oup.com/imajna/article/41/4/2311/5893596.

Spencer H. Bryngelson. Apple neural engine: Architecture, programming, and performance, 2026a. arXiv:2606.22283.

Spencer H. Bryngelson. ANEForge: Python for direct computation on the Apple Neural Engine, 2026b. arXiv:2606.17090.

Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on Apple silicon, 2023. URL https://github.com/ml-explore/mlx. Version 0.32.0.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *ICLR*, 2022.

Andrej Karpathy. Tiny Shakespeare: Sample corpus in char-rnn. GitHub repository, data/tinyshakespeare/input.txt, 2015. URL https://github.com/karpathy/char-rnn/blob/370cbcd448eb7daf32f21a6be560b70e0b33c4e3/data/tinyshakespeare/input.txt. Commit 370cbcd448eb7daf32f21a6be560b70e0b33c4e3; accessed 2026-09-06.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Ramchand Kumaresan. Orion: Characterizing and programming Apple's Neural Engine for LLM training and inference, 2026. arXiv:2603.06728.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReLoRA: High-rank training through low-rank updates. In *ICLR*, 2024.

Jieke Lin, Wanyu Wang, Longxiang Yin, and Yinhe Han. KAITIAN: A unified communication framework for enabling efficient collaboration across heterogeneous accelerators in embodied AI systems, 2025. URL https://arxiv.org/abs/2505.10183.

maderix. ANE training: Backpropagation on Apple Neural Engine, 2026. URL https://github.com/maderix/ANE/tree/f213c8db68e0a328ac300544cc12b9c41bac9f05. Initial software release, February 28, commit f213c8d.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.07843. Introduces the WikiText corpus; preprint arXiv:1609.07843, 2016.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *ICLR*, 2018.

Om Mohite. FusionML: Prefill, not decode—mechanism and boundaries of CPU+GPU co-execution on unified-memory Apple silicon, 2026. arXiv:2607.22785.

PyTorch. Introducing accelerated PyTorch training on Mac, 2022. URL https://pytorch.org/blog/introducing-accelerated-pytorch-training-on-mac/.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. GPT-2 small, 124M parameters (originally reported as 117M).

Daliang Xu, Mengwei Xu, Qipeng Wang, Shangguang Wang, Yun Ma, Kang Huang, Gang Huang, Xin Jin, and Xuanzhe Liu. Mandheling: Mixed-precision on-device DNN training with DSP offloading. In *MobiCom*, 2022.

Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. FwdLLM: Efficient federated finetuning of large language models with perturbed inferences. In *USENIX ATC*, 2024.

Juntao Zhao, Borui Wan, Yanghua Peng, Haibin Lin, Yibo Zhu, and Chuan Wu. QSync: Quantization-minimized synchronous distributed training across hybrid devices. In *IEEE IPDPS*, 2024.

Ligeng Zhu, Lanxiang Hu, Ji Lin, Wei-Chen Wang, Wei-Ming Chen, Chuang Gan, and Song Han. PockEngine: Sparse and efficient fine-tuning in a pocket. In *MICRO*, 2023.

A Experimental Protocols and Provenance

All training panels in this appendix ran to completion under plans fixed before execution; no seeds were added, replaced, or selected on the basis of outcomes. Analyses specified after results were observed are identified as such.

Head variants. The log-normalizer head variant reconstructs probabilities from an fp16 log-normalizer. The HeteroEngine head uses direct shifted normalization in Equation 3. The two programs share the same trainable scope, frozen parameters, arithmetic type, and loss scale. Their compiler reports contain 1,593 and 1,586 ANE-preferred operations, respectively. The artifact distinguishes source and compiled MIL hashes; an identical constant-weight blob alone does not establish an identical graph.

Shakespeare input identity. The archived 1,115,394-byte text is byte-identical to `data/tinyshakespeare/input.txt` at the cited 2015 `char-rnn` commit. Encoding the entire text once with GPT-2 BPE, without stripping or inserting end-of-text tokens, reproduces all 338,025 archived int32 IDs and their complete file hash. The 90/10 split is applied after encoding. The artifact records text, token-array, and upstream file fingerprints.

Primary quality panel and head comparison (record 317). The panel uses seeds 101–105, $Q = 4, K = 6$, 300 updates, constant learning rate 5×10^{-5}, and three arms: hybrid with the log-normalizer head, hybrid with the HeteroEngine head, and GPU-only. Arm order rotates by seed. Evaluation uses the common GPU implementation and the same 30 held-out offsets. The primary CE contrast subtracts GPU-only from HeteroEngine, with paired 90% Student-t intervals and a $\pm \ln(1.01)$ margin. Per-seed contrasts appear in Table 5.

Table 5: Per-seed CE contrasts in the head comparison. Log-norm. denotes the hybrid arm with the log-normalizer head variant; HeteroEngine denotes the hybrid arm with the shifted-normalization head.

Seed	Log-norm. − GPU	HeteroEngine − GPU	HeteroEngine − Log-norm.
101	$+0.03168$	$+0.00344$	-0.02824
102	$+0.02813$	-0.00159	-0.02971
103	$+0.03942$	-0.00091	-0.04033
104	$+0.03063$	$+0.00410$	-0.02653
105	$+0.01905$	-0.00199	-0.02104

Relative to the log-normalizer arm, HeteroEngine has a CE contrast of -0.02917, with a 90% interval of $[-0.03589, -0.02245]$. The two head variants produce scalar ANE losses that are identical at matched state and input; final quality is evaluated on the GPU after the completed updates.

State and evaluation checks. For ten $Q = 2, K = 2$ updates at seed 33, the native driver, a sequential reference, and an independent reference driver match all four hashes: fp32 parameters, Adam first moment, Adam second moment, and fp16 state image. ANE diagnostics install the current fp16 image before prediction. Because diagnostic predictions write gradient bank zero, they invalidate the GPU-only path's zero-bank cache; a 300-update GPU-only run with intermediate diagnostics matches the evaluation-free reference in all four state hashes. These checks separate state freshness and diagnostic side effects from ANE–GPU arithmetic differences. Record 318 additionally passes five ten-update driver-pair checks across hybrid and pure-GPU paths at $M = 10$ and 16, and its 300-update pure-GPU $M = 10$ run at seed 101 matches the quality-panel GPU control in parameters, both moments, and the terminal fp16 image. That image is constructed after the timed bracket; the pure-GPU loop does not maintain an ANE shadow.

Resource protocol (record 318). The throughput grid contains GPU $M \in \{4, 8, 10, 12, 16, 20, 24, 32\}$; hybrid $Q{:}K \in \{3{:}6, 4{:}5, 4{:}6, 4{:}7, 5{:}6, 5{:}7\}$; and all-ANE $Q = 4, K = 0$. Each of three blocks runs every cell for 120 updates, with 20 warmup updates and seed 7. Selection uses mean throughput over complete blocks. The joint block bootstrap enumerates all 27 ordered resamples of the three

complete blocks and reselects the fastest GPU M in each: $M = 24$ wins 20 resamples and $M = 20$ wins seven. With only three blocks, its percentile intervals are descriptive; the independent metered panel supplies the reported speed and energy intervals.

The metered panel fixes GPU $M = 4, 10, 24$, hybrid $Q = 4, K = 6$, and all-ANE $Q = 4, K = 0$ before collection. Five cyclically ordered blocks place each configuration in each order position once. Every run uses seed 7 and 124 updates; the timed bracket covers updates 5–124. One finite-count `powermetrics` sampler per run records CPU, GPU, and ANE rails at 500 ms intervals and exits naturally before the next run. Readiness, ownership, coverage, exit status, and unchanged AC power and High Power Mode are checked for each run. Output data are written outside the synchronized project folder during metering. No idle subtraction, activity threshold, or power-sample exclusion is used. The parser takes the first rail tuple in each sample and integrates piecewise-constant sample power over its overlap with the full training bracket. Mean energy, its sample SD, and throughput come from these same five blocks. Paired log ratios use two-sided 95% Student-t intervals with four degrees of freedom; power samples do not increase the replicate count. Unmetered and metered throughput are reported separately.

Energy-integration uncertainty. Sample elapsed times are accumulated on a monotonic axis, and the observed wall–monotonic offset envelope bounds the alignment of each sample with the training bracket; receipt times bound sample completion. The prespecified acceptance criterion requires the resulting energy-uncertainty width to be at most 1% of the point estimate. All 25 runs pass, with a maximum width of 0.4952%. This bound concerns clock alignment and timestamp rounding, not sensor calibration or power variation within a sample. The artifact archives the parser and its alignment checks.

Sustained-load protocol. The hybrid and the selected GPU control each start from the pretrained parameters at seed 7 and constant learning rate 5×10^{-5}, without intermediate evaluation. Before either run, we fix $N = 20 + \lceil 1900/d_{\min} \rceil$, where $d_{\min}$ is each arm's shortest post-warmup update in the throughput grid. This gives 11,118 updates for $Q = 4, K = 6$ and 2,527 for GPU $M = 24$. The hybrid runs first and the GPU second, with one trace per arm. A single finite-count sampler requests 1,000 ms intervals and exits naturally before the next run.

The sustained window begins after update 20. A valid trace requires both its recorded wall duration and its clock-envelope lower bound to be at least 1,800 seconds, with every stored training loss and final common-GPU evaluation finite. These checks add no per-update gradient scans. The complete window, the first and final 300 seconds, and every complete nonoverlapping 300-second bin must pass the same 1% energy-alignment uncertainty gate. Tokens are credited when updates finish in $(t_{\text{start}}, t_{\text{end}}]$; updates crossing boundaries are not split. A final incomplete bin remains in the complete-window integral but is not treated as another five-minute observation. The first/final-window throughput change measures drift within each trace. Time bins do not provide independent replications or establish thermal equilibrium.

Both traces pass every duration, loss-finiteness, and energy-alignment gate. The maximum alignment interval width across all checked windows is 0.0795% of the point estimate. Independent integration reproduces both traces. Over the complete windows, the hybrid and GPU achieve 10,567 and 7,604 tok/s, respectively, at 27.435 and 27.675 W of Combined power and 2.596 and 3.639 J/1k tokens. These long runs revisit the small training split and overfit: final common-GPU CE is 6.786 and 6.635. Their unequal fixed token budgets support sustained resource measurements, not a quality comparison. Figure 5 plots throughput, Combined power, and energy per token in complete, nonoverlapping five-minute bins after warmup.

Idealized capacity bound. For an idealized linear GPU cost F per microbatch, divisible work, and zero state-write and update overhead, balancing the legs gives $T^* = MPF/(P + F)$ and capacity $S(1/P + 1/F)$. This bound assumes the specified per-microbatch engine costs; operator-dependent placement requires a different cost model. Measured throughput grids include the discrete work and runtime costs omitted by the idealized model.

Overlap observations. Across the three $Q = 4, K = 6$ grid runs, the median per-update ANE prediction phase, covering all Q predictions, is 204.5–208.8 ms, state installation is 3.2–3.3 ms, GPU completion wait is 205.8–211.8 ms, and reduction plus Adam and fp16 export is 17.8–17.9 ms.

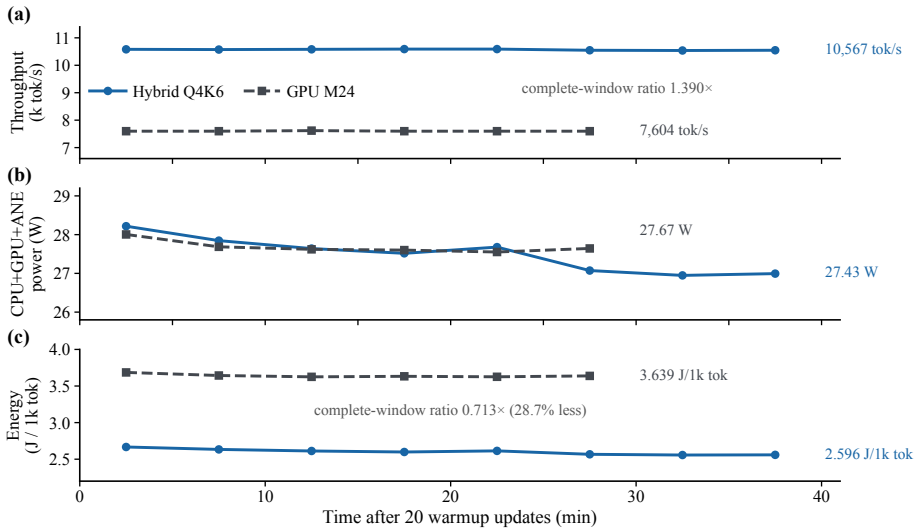

Figure 5: **Throughput, power, and energy during sustained training.** Complete nonoverlapping five-minute bins after warmup; right labels: complete-window values.

The average complete-update time is 229.1–233.7 ms. The two engine intervals overlap within the same update. These are host-side observations: phase medians discard the first two updates, and the printed update average covers updates 5–120. The primary grid throughputs instead discard 20 warmup updates. The raw logs preserve these definitions; no sum of phase medians is treated as an exact serialized counterfactual. In the metered panel, the hybrid's 10,271 tok/s is 87.6% (per block 87.2–88.0%) of the summed GPU-only $M = 24$ and all-ANE throughputs (11,726 tok/s). Its 249.3 ms update overlaps a 226.3 ms GPU leg with a 221.0 ms ANE leg (state write and four predictions) and then spends 19.3 ms on reduction, Adam, and fp16 export (host-timer medians averaged over the five blocks). At the same GPU clock, a GPU microbatch takes 37.7 ms in the hybrid's six-microbatch group against 34.4 ms in GPU-only $M = 24$; batch size differs between the two, so we do not attribute this difference to contention alone.

Allocation sweep (record 319). The sweep uses seeds 101–105 and 300 updates at a constant learning rate. New allocations are $Q{:}K = 1{:}7, 1{:}3, 2{:}2, 4{:}0$, with GPU controls at $M = 8$ and $M = 4$. Each contrast uses its matching control and a pointwise paired 90% interval. The $Q = 4, K = 6$ result is reused from record 317 and is not a newly collected seed panel. Every allocation passes its pointwise equivalence gate. The $Q = 1, K = 3$, $Q = 2, K = 2$, and $Q = 4, K = 0$ contrasts share the same GPU $M = 4$ control, so they are not independent panels. Because M differs across allocations, the grid cannot be treated as a pure causal dose–response experiment. The artifact reports every paired seed and control endpoint.

WikiText preprocessing and long-training plan (record 320). The source dataset is `Salesforce/wikitext`, configuration `wikitext-103-raw-v1`, revision `f776294184f13b8ff2337b3841cf9269a6216d1e`. The training input is a fixed 20,000,000-token prefix from the first training Parquet shard, `train-00000-of-00002.parquet`. Original rows are joined with two newlines within consecutive 512-row Parquet batches; each batch is GPT-2-tokenized independently, the token arrays are concatenated, and the prefix is retained. No end-of-text token is appended. These tokenizer reset boundaries are part of the input definition. Validation and test rows are each joined and encoded in one pass, producing 251,048 and 287,644 tokens. The manifest retains hashes for source Parquet, text preparation, and all token arrays. This subset is not the full WikiText-103 training corpus. The five training seeds are 201–205, with arm order drawn once using seed 32006. Every arm starts from the same pretrained parameters and uses constant learning rate 5×10^{-5}. The common token budget is 2,580,480: 1,008 updates for the hybrid and GPU $M = 10$, or 420 for GPU $M = 24$. These checkpoints compare equal token budgets; the $M = 24$ arm takes fewer Adam updates. Each run continues until its accumulated complete-update time first crosses 450

seconds. The update clock starts before batch selection and stops after Adam completes. It includes compilation performed inside an update and excludes offline Core ML compilation, model setup, validation, and checkpoint writes. The artifact records the actual elapsed time, completed updates, tokens, and crossing overshoot.

Validation runs at initialization and every 30 accumulated training seconds on 64 fixed nonoverlapping 256-target blocks, selected with seed 999 from the official validation split. After initialization, each plotted point uses the first completed update crossing its scheduled time, retaining the actual elapsed time. Validation is not used to select hyperparameters or stopping points. At the equal-token and final checkpoints, GPU evaluation covers all 1,123 complete 256-target test blocks: 287,488 targets. The first input token supplies context and the final 155 incomplete targets are omitted. Perplexity is based on GPT-2 BPE tokens, not the benchmark's word-level convention.

The primary equal-token contrast is hybrid minus GPU $M = 10$ CE, using a paired 90% Student-t interval over the five seeds and a $\pm\ln(1.01)$ equivalence margin. Equal-time contrasts and the secondary same-token $M = 24$ comparison use pointwise paired 95% intervals, without adjustment for multiple comparisons. Training seeds, not test blocks, are the replication unit. Both training checkpoints store the full fp32 trainable parameter vector and its fingerprints. A separate evaluator reproduces every final test-block value exactly in the two device preflight checks. Arm order follows the saved random draw. Every run stops at its first complete-update crossing, with overshoots of 0.003–0.519 seconds. Independent recalculation from the saved per-block test values reproduces all primary and secondary intervals.

Sampler-package crossover (record 321). Three blocks each contain four conditions: hybrid $Q = 4, K = 6$ and GPU $M = 24$, each with the complete `powermetrics` sampler and collector enabled or disabled. All 12 fresh processes use the unchanged qualified driver, compiled programs, initial weights, seed 7, and constant learning rate 5×10^{-5}. Each runs 124 updates; throughput divides 307,200 hybrid or 737,280 GPU tokens by the same printed wall-clock bracket covering updates 5–124. Every condition starts training after a fixed five-second prelaunch delay and occupies a fixed slot: 120 seconds for the hybrid and 200 seconds for the GPU. Enabled samplers request 500 ms intervals with finite counts of 210 and 370, respectively, and exit naturally. Disabled conditions launch neither sampler nor collector. The prelaunch and slot timing tolerance is fixed at 0.5 seconds before collection. The treatment includes sampling, output collection, and clock recording; it does not isolate CPU contention.

The fixed order places each condition in three different positions. On-first and off-first pairs are balanced across cells, but each cell has a 1/2 ordering imbalance across its three blocks; order balance is thus partial. The six terminal states within each cell match exactly in fp32 weights, both Adam moments, and the fp16 state image. Runtime, source and input fingerprints, timing integrity, and all stored losses pass their checks.

Table 6 reports geometric means and pointwise paired log-t intervals across three blocks, with two degrees of freedom. The relative-advantage interaction is $D = (R_{\mathrm{H,on}}/R_{\mathrm{G,on}})/(R_{\mathrm{H,off}}/R_{\mathrm{G,off}})$, where R denotes throughput. It is 1.009696, with a 90% interval of $[0.998870, 1.020639]$. The prespecified criterion requires that interval to lie inside $[0.98, 1.02]$; its upper end reaches 1.020639, so the equivalence test is inconclusive rather than passed. The sampler package's estimated effect on the relative advantage is about 1%. These are measurement blocks at one training seed, not independent training seeds. No energy is inferred for the unmetered conditions.

Table 6: **Sampler-package crossover.** Paired throughput ratios over three complete blocks. The interaction divides the hybrid/GPU ratio with the sampler by that without it.

Contrast	Ratio	CI level	Confidence interval
Hybrid on/off	1.009032	95%	$[0.994596, 1.023678]$
GPU on/off	0.999343	95%	$[0.997702, 1.000986]$
Relative-advantage interaction	1.009696	90%	$[0.998870, 1.020639]$

Long matched-token extension with paired evaluators (records 322 and 328–329). The training plan was fixed before execution and specifies five fresh seeds 301–305, two arms ($Q = 4, K = 6$ and

pure GPU $M = 10$), and 7,056 updates. It retains the record-320 model, compiled ANE program, driver, dataset, and constant learning rate, checked against their archived fingerprints. Training windows are sampled with replacement from the fixed 20M-token prefix; 18.06M sampled tokens do not imply 18.06M distinct tokens. Arm order is fixed in the plan, with the hybrid first for two seeds and the GPU first for three. No sampler runs during this extension.

All ten runs complete. Both arms save full parameter checkpoints at updates 1,008, 2,016, 4,032, and 7,056, and each of the 40 checkpoints is scored on the same 1,123 test blocks (287,488 test targets) by common bf16 and fp32 graphs. Across the 40 paired evaluations, parameter, checkpoint-file, input, target-offset, and update-count checks pass. The initial execution (record 322) specified bf16 endpoint equivalence as its outcome. After its results and the short-run evaluator-sensitivity analysis (Appendix B) were available, a follow-up plan (record 328) specified paired bf16/fp32 endpoint advantage, equivalence, and evaluator interaction, and we reran the same five seeds; the rerun reproduces all 40 parameter hashes and bf16 per-window evaluation arrays exactly and adds no independent seeds. Record 328 supplies the reported evaluations and retains all 40 parameter checkpoints; Adam moments were not stored.

The primary equivalence test requires the paired 90% CE interval at update 7,056 to lie inside $\pm \ln(1.01)$. It passes under both evaluators: the final bf16 mean is -0.00452488 CE, with a 90% interval $[-0.00491011, -0.00413965]$, and the final fp32 90% PPL interval is $[-0.228\%, -0.179\%]$. A secondary replication check at update 1,008 also passes: its bf16 CE interval contains the earlier five-seed mean gap of 0.00204 CE ($+0.204\%$, seeds 201–205). All pointwise 90% intervals at every checkpoint lie inside the equivalence band under both evaluators. The follow-up plan's endpoint-advantage criterion (both evaluator-specific paired 95% upper CE bounds below zero) is met; because it was specified after the initial results on these same seeds were observed, it is a confirmation on a rerun rather than an independent replication. Table 7 reports the paired 95% intervals for both evaluators. At the final endpoint both exclude zero in favor of the hybrid; at update 1,008 the bf16 interval lies above zero while the fp32 interval lies below it. The paired evaluator interaction at the final endpoint, $d_s^{\mathrm{fp32}} - d_s^{\mathrm{bf16}}$, is $+0.002486$ CE (95% CI $[+0.001880, +0.003091]$), so the two evaluators measure different effect sizes on the same parameters. All intervals are pointwise over the five seeds; test blocks and checkpoints are not additional replicates. These comparisons concern evaluation precision; neither training arm uses fp32 forward and backward computation.

Table 7: **Long matched-token quality trajectory under two common evaluators.** Hybrid minus GPU $M = 10$, five paired training seeds; hybrid PPL change in percent with paired 95% intervals, each entry $100(\exp(d) - 1)$ applied to the mean CE contrast or its interval endpoints. The same parameter checkpoints underlie both columns. These are pointwise intervals; repeated checkpoints are not independent replications. The last row is the primary endpoint.

Updates	Tokens (M)	Common bf16: gap [95% CI]	Common fp32: gap [95% CI]
1,008	2.58048	$+0.233$ $[+0.191, +0.276]$	-0.132 $[-0.151, -0.112]$
2,016	5.16096	$+0.040$ $[-0.007, +0.088]$	-0.147 $[-0.178, -0.115]$
4,032	10.32192	-0.237 $[-0.311, -0.162]$	-0.151 $[-0.176, -0.126]$
7,056	18.06336	-0.451 $[-0.501, -0.402]$	-0.204 $[-0.236, -0.172]$

Eight of the 80 checkpoint evaluations were rerun serially from local input copies after a transient cloud-file outage, with matching source, weight, data, and checkpoint hashes; no training run was repeated.

Measured execution time. The ten training panels contain 165 completed training processes; reused controls and the reused $Q = 4, K = 6$ endpoints add no training runs. Fixed-weight diagnostics (records 323 and 325) are described in Appendix B. Table 8 reports recorded wall times with their measurement boundaries; these are elapsed times on one shared GPU+ANE host, not separate device-hours.

The long-extension process times exclude the separate checkpoint test evaluations (3,766.3 wall seconds in total). The earlier process times include setup and final evaluation; the record-320 WikiText process also includes intermediate validation and checkpoint writes, but not its separate checkpoint-test helper. The sampler crossover's twelve fixed slots total 1,920 seconds; its 866.9

Table 8: Recorded execution time for the ten completed training panels.

Panel	Runs	Wall time (s)	Recorded interval
Head comparison	15	1,366.4	Launch through artifact storage
ANE allocations	30	1,983.1	Launch through artifact storage
Throughput grid	45	2,514.4	Training process
Metered panel	25	1,268.0	Training process
Sustained load	2	4,755.0	Training process
WikiText-103	15	7,535.4	Training process
Sampler crossover	12	866.9	Training process
WikiText long extension	10	21,889.8	Training process
GPU fp16 quality	5	2,028.1	Training process
Earlier fp16 timing panel	6	484.9	Training process

seconds of training process time fall within those slots. The fp16 quality panel also records 115.1 seconds for its five bf16 checkpoint evaluations, and record 327 preserves the 15 fp32 evaluation processes separately. Record 322, the earlier execution of the long-extension plan, contains ten identical training processes (20,867.2 seconds) that are not counted here. The M2 panels (records 330 and 332) and the fp16-control panel (record 334) are also not counted; their run logs and times are in the artifact. These intervals include setup and host work, not only device-active time. Development, qualification, diagnostic, compilation, and data-preparation costs are not completely recorded; we do not estimate total project compute from this ledger.

B Numerical Diagnostics

Plain-model gradient probes. Both ANE programs are evaluated at the same master parameter checkpoint and token offsets, using the same fp16 parameter casts. We sample 4,096 coordinates per input from every trained matrix: 294,912 of 84,934,656 coordinates (0.3472%). Equal samples per matrix oversample attention matrices relative to larger feed-forward matrices; reported pooled norms concern these sampled coordinates. The artifact contains all 72 matrix summaries. Within each configuration in Table 9, both ANE programs use the same GPU reference, coordinate indices, and token offsets.

Table 9: Gradient diagnostics at shared checkpoints against the indicated GPU reference. Log-norm. denotes the log-normalizer head variant; HeteroEngine uses shifted normalization.

Shared checkpoint / reference	Pooled relative L_2		Mean input cosine	
	Log-norm.	HeteroEngine	Log-norm.	HeteroEngine
Initialization / fp32 GPU	0.36388	0.01891	0.92509	0.99982
Update 300 / fp32 GPU	0.30762	0.02190	0.95567	0.99976
Update 300 / bf16 GPU	0.32343	0.10015	0.95020	0.99500

For sampled ANE and GPU reference gradients a_i and r_i, pooled relative L_2 is

$$\sqrt{\textstyle\sum_i \|a_i - r_i\|_2^2 / \sum_i \|r_i\|_2^2},$$

with sums over the 16 inputs. Cosines are averaged over the 16 inputs; each uses the concatenated sampled coordinates. The trained checkpoint comes from HeteroEngine training at $Q = 2, K = 2$, seed 33. Against the common fp32 reference, all 1,152 matrix–input pairs improve in both relative L_2 and cosine at each checkpoint. The bf16 reference remains a sensitivity comparison: all pairs improve in relative L_2, and 1,151 improve in cosine at update 300. Initialization upcasts the archived fp16 weights into the fp32 master vector; this does not recover their pre-export precision. The trained checkpoint instead supplies its saved fp32 master vector. The ANE uses its fp16 cast. Embedding sums are formed in fp32, then cast to each path's input dtype. The comparison therefore includes parameter, input, and arithmetic precision effects. The additional fp32-reference probe reproduces

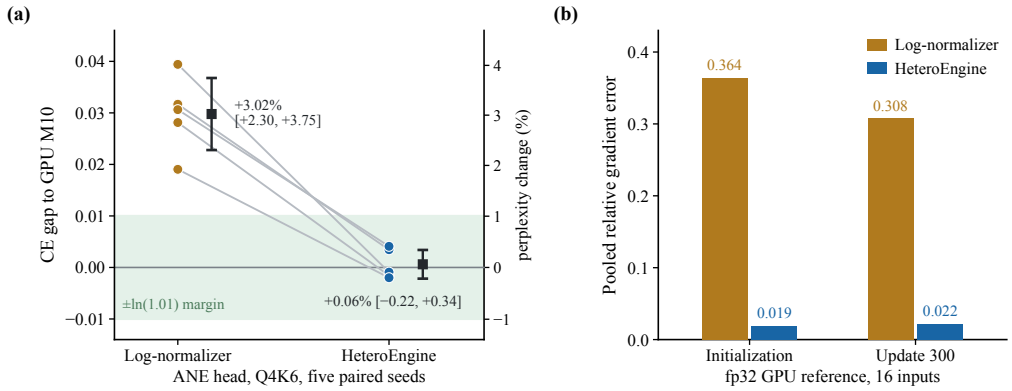

Figure 6: **Shifted normalization improves gradients and training.** (a) CE gap (left axis) and perplexity change (right axis) vs. GPU M=10 for both heads at Q=4, K=6 after 300 Shakespeare updates; lines: seeds; black markers: means and paired 90% CIs; shading: equivalence margin; labels: perplexity change [CI]. (b) Pooled relative gradient error against the fp32 GPU reference.

all 16 archived HeteroEngine ANE gradients and scalar losses bit-for-bit; only the GPU reference changes. For eight overlapping initialization inputs, the ANE arrays also match the earlier bf16-reference probe byte-for-byte, so the fp32 comparison changes the reference rather than the ANE outputs. Figure 6(b) plots pooled gradient error against the common fp32 reference. Panel (a) shows the five paired training CE gaps at $M = 10$: lines connect seeds, black markers and bars denote means and paired 90% intervals, and shading marks the equivalence margin.

Same-partition gradient comparison (records 323–324). Two fixed fp32 master vectors come from record 320, seed 201, update 1,008: one hybrid endpoint and one GPU $M = 10$ endpoint. Six sets of ten training-window offsets are drawn with RNG seed 323 and shared between the checkpoints. Four paths average over the same ten windows: A, GPU bf16 batch 10; B, GPU bf16 batch 6 plus four GPU singles; C, GPU bf16 batch 6 plus four ANE fp16 singles; and D, ten GPU fp32 singles accumulated in float64. The first four offsets form the singles group. The reference average is cast to fp32 before comparison. No parameters or optimizer states are updated.

For each condition, relative error is $\|g - D\|_2/\|D\|_2$ over all 84,934,656 coordinates. Table 10 summarizes the twelve conditions, which share inputs and originate from one training seed. They are not twelve independent training replications.

Table 10: **Gradient error with and without ANE replacement.** Median full-vector relative L_2 error across two checkpoints and six shared input sets. The last two rows compare only the four single-window gradients with the corresponding four-window fp32 reference.

Path	Median error (%)
A: GPU batch 10	10.327
B: GPU batch 6 + four GPU singles	10.492
C: GPU batch 6 + four ANE singles	7.259
Four GPU singles	9.001
Four ANE singles	2.518

C has lower error than both A and B in every condition; the median per-condition C/B ratio is 0.711502, or 28.85% lower error, and the four ANE singles are more accurate than the four GPU singles in every condition. Partitioning alone is not neutral: the B/A ratio lies outside the prespecified $[0.9, 1.1]$ band in three of the twelve conditions (1.640094, 1.128690, and 0.657598). The probe therefore establishes an ANE-path contribution under this partition; it does not separate datatype, kernel, and operation-order effects or test the long-training optimizer trajectory.

Decomposing two of the exceptional sets into their eight individual window evaluations shows large GPU bf16 errors on two windows, 146.58% and 41.10%, versus 9.15% and 6.02% on the ANE; this selected sample does not estimate an outlier frequency across all windows. The maximum individual ANE–GPU bf16 loss difference is 0.0327113. Per-matrix relative-error rankings do not locate the first erroneous operation or establish attention cancellation as the cause.

The artifact preserves full-vector scalar metrics and 4,096 sampled coordinates per matrix; full reference vectors were deleted after use. Independent analysis reproduces the sampled-array comparisons and aggregates the full-vector logs. This diagnostic is separate from the 165 training processes in Table 8.

Stabilized GPU fp16 control, evaluator sensitivity, and throughput (records 325–327 and 334). An exploratory GPU control keeps matrix operations and activations in fp16, computes LayerNorm intermediates in fp32 before casting its output back to fp16, and applies a loss scale of 1024 with fp32 unscaling before Adam. It retains the fp32 masters and moments, tanh-GELU formula, attention, learning rate, and trainable matrices. It neither clips gradients nor skips updates. A plain dtype substitution does not pass the gradient check; protecting LayerNorm and adding loss scaling are required for accurate gradients. Input tracing and separate variance recalculation point to a LayerNorm range problem, but do not directly measure every intermediate inside the compiled kernel.

On the same two checkpoints and six input sets as the same-partition probe above, this path has a median full-gradient relative L_2 error of 1.116% (maximum 2.389%), versus 10.327% for GPU bf16 and 7.259% for the hybrid. It has lower error in all twelve conditions. The full fp32 reference hashes match the earlier diagnostic; these are repeated fixed-input probes, not independent training trials.

The candidate was selected before training endpoints were observed. Five fp16 runs use seeds 201–205, 1,008 updates, $M = 10$, and 2,580,480 sampled tokens; bf16 GPU and hybrid controls reuse the archived record-320 runs. All 5,040 updates complete without triggering the nonfinite-gradient guard. The primary common bf16 evaluator finds fp16 perplexity $+0.288\%$ above the bf16 GPU control and $+0.083\%$ above the hybrid; both pass the existing 1% equivalence test.

After the first three primary bf16 endpoints were observed, an exploratory follow-up specified fp32 evaluation of all fifteen checkpoints; parameter and test-input hashes match across evaluators. Table 11 reports both sets of results. The fp16–bf16 ranking reverses with evaluation precision; fp16 versus hybrid under fp32 has a 95% interval spanning zero. The bf16 evaluator remains the primary one, and neither result replaces the other. This sensitivity is why the long-training panel reports both evaluators (Appendix A).

Table 11: **GPU fp16 quality under two common evaluators.** PPL changes and paired 95% intervals across five seeds, in percent, after 1,008 matched updates. Negative values favor the fp16-trained GPU model.

Evaluator	fp16 vs. bf16 GPU	fp16 vs. hybrid
bf16 (primary)	$+0.288$ $[+0.216, +0.359]$	$+0.083$ $[+0.027, +0.140]$
fp32 (exploratory)	-0.192 $[-0.252, -0.131]$	-0.020 $[-0.051, +0.012]$

A preregistered follow-up panel (record 334) compares the hybrid with this fp16 path on the same M4 Pro. An unmetered grid first swept the fp16 path over $M \in \{4, 8, 10, 12, 16, 20, 24, 32\}$ in three rotated passes; $M{=}24$ had the highest median throughput and $M{=}20$ lay 0.97% below it, inside the prespecified 1% tie band, so the rule favoring the smaller batch selected $M{=}20$. Run at the same bf16 $M{=}24$ configuration, the bf16 variant of the fp16 driver reached 0.990 of the throughput of the driver used for the hybrid and the bf16 control, inside the prespecified $[0.97, 1.03]$ band, so driver differences account for at most about 1% of the comparison. A five-block metered panel then measured the hybrid ($Q{=}4, K{=}6$), bf16 GPU $M{=}24$, and fp16 GPU $M{=}20$ and $M{=}10$ at seed 7, using the sampler and timed bracket of Appendix A (one finite 500 ms sampler per run, updates 5–124, absolute CPU+GPU+ANE energy without idle subtraction). The panel applied its own preregistered gates (exit status, sampler hygiene, coverage, rail identity, manifest, bracket stability, and a load-average gate); the fifth block repeats the first block's order. All 20 runs passed these gates. Three earlier launches stopped before or during the unmetered grid because of launcher, disk-space,

and memory-pressure faults, and the grid was restarted from scratch; no metered run was repeated or excluded.

The hybrid delivered $1.440\times$ the throughput of fp16 $M{=}20$ (95% CI $[1.422, 1.457]$) at $0.692\times$ its CPU+GPU+ANE energy per token ($[0.679, 0.704]$), and $1.514\times$ and $0.696\times$ against fp16 $M{=}10$. At $M{=}20$ the fp16 path was 4.6% slower than bf16 $M{=}24$ and used 3.0% more energy per token. In the same panel the hybrid's advantage over bf16 $M{=}24$ was $1.373\times$ ($[1.356, 1.391]$) and $0.712\times$ ($[0.699, 0.726]$), smaller than in the primary panel ($1.419\times$, $0.691\times$), whose values lie outside these intervals. Both the bf16 GPU and hybrid runs were faster and drew less power than in the primary panel (8,108 versus 7,240 and 11,135 versus 10,271 tok/s; 26.0 versus 30.3 and 25.4 versus 29.7 W). The machine state also differed (a rebuilt Python environment with the same package versions, other applications open, and several gigabytes of swap in use); we did not isolate the cause. We keep the primary panel as the headline and report this panel separately. It measures speed and energy only: fp16 quality comes from the five-seed WikiText panel above (Table 11), measured at $M{=}10$ rather than the selected $M{=}20$. The panel supersedes an earlier three-block $M{=}24$ timing comparison (fp16/bf16 0.968, 95% CI $[0.776, 1.207]$).

Head observations and observer effect. For two diagnostic inputs, we export scaled probabilities, the hidden-state adjoint, and normalization intermediates. The instrumented graphs produce a 0.11–0.18% relative L_2 difference in sampled gradients from ordinary prediction, although the scalar loss is unchanged. Consequently, these are observations of separate diagnostic graphs, not exact traces of the production graph. With the raw exported fp16 scaled probabilities and float64 CPU sums, the probability-mass RMS error is 1.847% and 1.790% for the log-normalizer diagnostic graph, versus 0.0557% and 0.0534% for the HeteroEngine diagnostic graph.

Let $\epsilon = a - (m + \log Z)$ be computed in float64 from exported intermediates. The log-normalizer graph's mass error closely follows $\exp(-\epsilon) - 1$: correlations exceed 0.9996 on both inputs and the unfitted prediction has $R^2 > 0.9993$. The HeteroEngine head's mass error is essentially uncorrelated with this quantity on these inputs. These observations support the log-normalizer mechanism in the diagnostic graphs; they do not assign a percentage of the final training gap to one primitive, or separate all normalization rounding from underflow.

Why input dependence does not imply irreducible noise. For a computed probability mass of $1 + \delta$, decomposing output embeddings into their row mean μ and centered part exposes an extra $\delta\mu$ in $E^\top p - E_y$, or $(\lambda/S)\delta\mu$ in the loss-scaled hidden-state adjoint. The hidden activations, logits, and rounding error depend deterministically on the input. A large batch-varying share of the error therefore coexists with a correctable arithmetic expression. For the log-normalizer head at the pretrained point, write the paired ANE–GPU microbatch error δ as a mean $\bar{\delta}$ plus a fluctuation, so that $\mathbb{E}\|\delta\|^2 = \|\bar{\delta}\|^2 + \operatorname{tr}\Sigma$. Over 64 paired batches the input-varying term $\operatorname{tr}\Sigma$ accounts for 96.8% and the input-independent term $\|\bar{\delta}\|^2$ 3.2% (record 292). This input dependence does not rule out systematic arithmetic error.

Localizing the log-normalizer penalty (record 297). Ten-seed panels (seeds 13–22, 300 updates, common GPU evaluator) compare the log-normalizer hybrid with three GPU controls: a batched control that evaluates all M microbatches together, a sequential control that evaluates them individually, and a GPU-mirror control that preserves the hybrid's partition into Q individual microbatches and one batched K-microbatch group but uses GPU arithmetic for both. At $M{=}4$, hybrid$-$batched is $+0.0453$ CE (90% CI $[+0.0407, +0.0499]$, 10/10 seeds positive) and hybrid$-$mirror, the ANE-arithmetic contrast, is $+0.0448$ $[+0.0401, +0.0495]$, whereas mirror$-$batched ($+0.0005$ $[-0.0009, +0.0018]$) and batched$-$sequential ($+0.0001$ $[-0.0006, +0.0008]$) are equivalent within $\pm\ln(1.01)$. At $M{=}5$ the corresponding contrasts are $+0.0372$ $[+0.0327, +0.0416]$, $+0.0362$ $[+0.0317, +0.0407]$, and $+0.0010$ $[-0.0002, +0.0021]$. Reduction grouping and batched arithmetic are neutral; the log-normalizer penalty is localized to ANE arithmetic, consistent with the head-level diagnosis above.

C Implementation Evidence

Implementation specification. The pretrained weights come from the Hugging Face `gpt2` checkpoint, loaded with `GPT2LMHeadModel.from_pretrained`, cast to fp16, and archived once; the

archive is identified by its SHA-256, and the checkpoint revision was not recorded. Both paths implement GPT-2 small: 12 layers, width 768, 12 attention heads of dimension 64, feed-forward width 3,072, a 50,257-token vocabulary, learned positions, LayerNorm with $\epsilon = 10^{-5}$, the tanh approximation of GELU, and an additive causal mask. Neither path applies dropout, unlike the 0.1 dropout of the reference GPT-2 configuration. The output layer is tied to the frozen token embedding. Each microbatch reads a 257-token window from the training split, giving 256 inputs and 256 next-token targets, and the loss is their mean cross-entropy. The loss scale λ is a constant of the compiled program. The fused optimizer applies Adam with bias correction and no weight decay or gradient clipping.

The trainable state is a flat vector of $P = 84{,}934{,}656$ entries ordered by block and matrix. Its stored tensors are fp32 masters and two Adam moments (3×324 MiB); one fp16 weight image (162 MiB); 24 fp16 `MLState` tensors whose declared shapes also total 162 MiB; and Q fp16 gradient banks of 162 MiB each (648 MiB at $Q{=}4$). The GPU also forms a per-update fp32 gradient sum over its K microbatches (324 MiB) and keeps bf16 copies of the frozen parameters (75 MiB). These are sizes of stored tensors, not measured peak memory; Core ML's internal allocations were not measured.

Native output backings. We compare backed and default-allocated native outputs on four fixed inputs, one per worker bank, using the qualified native class and the compiled HeteroEngine program. All 72 gradient matrices are bit-identical in every pair: 288 full-matrix comparisons. Every returned array has both the requested native object identity and data pointer, with the same layout as its MLX-owned buffer. NaN-filled banks are fully overwritten with finite gradients; the other banks and all 24 weight states remain unchanged, and scalar losses are bit-identical. These checks establish buffer identity and output equality at the public API boundary.

The contract, binding, and optimization measurements below concern the log-normalizer head variant or the explicitly identified contract probes. Their latencies are distinct from the HeteroEngine resource measurements.

The contract probes use one M4 Pro running macOS 26.5.2 and coremltools 9.0. With baked weight constants, the probe achieves 13.4 effective fp16 TFLOPS at 7.8 W. Passing weights as ordinary inputs incurs approximately 10 ms/MB of ingestion, compared with 0.02 ms/MB to copy the same bytes. For 67 MB of weights, state-resident prediction takes 4.2–4.6 ms, compared with 2.56 ms for baked constants and 666 ms for streamed inputs. Host `write_state` updates values at approximately 0.3 ms/MB through the Python binding and 0.0125 ms/MB through the native API. The ANE compiler rejects in-graph mutation (`coreml_update_state`); CPU-only and CPU+GPU configurations compile and execute that program. In the state-count probes, 52 tensors compile and 54 fail even with half the byte payload, while 24 states carrying 255 MB compile. Reducing the payload therefore does not remove the observed state-count boundary.

Native binding overhead. Table 12 reports Python-binding and native Core ML latency on matched models, states, and inputs. Native and Python-binding drivers produce identical gradients on matched state and input, and identical per-microbatch training-loss arrays over 300 updates. Native dispatch reduces both prediction and state-update latency.

Table 12: Binding latency on matched models, states, and inputs (log-normalizer head program; testbed denotes a four-layer, 16.8M-parameter model used for kernel-level A/B tests).

Operation	Python binding	Native API	Ratio
GPT-2 predict (fwd+bwd, 170 MB dW out)	106.9 ms	48.2 ms	2.2×
GPT-2 state write (170 MB fp16 image)	21 ms	3.6 ms	6×
Testbed predict (34.6 MB out)	16.4 ms	6.4 ms	2.6×
Testbed state write (32 MiB)	4.2 ms	0.40 ms	10×

Other implementation choices. Table 13 records the A/B measurements supporting each optimization. Baselines differ—testbed or GPT-2, Python or native as marked—so the effects are not additive. §6 reports end-to-end throughput and energy. On the four-layer, 16.8M-parameter testbed, the native ANE path takes 10.65 ms at 11.3 W against the GPU's 18.0 ms at 19.4 W.

Three further candidates were evaluated and not adopted. A flat-parameter VJP materialized an MLX scatter and took 1,305 ms, roughly 27× slower than the per-tensor path. `mx.fast` SDPA gave no speedup and changed the loss ($3.943 \to 3.996$). A batched ANE graph ($B{=}2$) gave a gradient cosine of 1.000 and losses of 4.3164 versus 4.3145, yet saved only about 1 ms with native output backings. The MLX 0.32.0 follow-up tests both additive and causal SDPA masks (record 298); neither improves the complete GPU leg. The HeteroEngine ANE head does not change this GPU computation.

Table 13: Adopted optimizations and their measured effect.

Optimization	Core idea	Measured effect
In-graph transposes	`transpose_x` on state tensors; no transposed copies	write payload halved
Fused flat optimizer	one compiled kernel: unscale + combine + Adam + fp16 image	$13.4 \to 4.8$ ms (testbed)
Zero-copy views	`np.array(copy=False)` over GPU buffers, both directions	-3.8 ms/step
Phase-shifted write + async enqueue	worker owns write→predict; GPU graphs enqueued first	enables the concurrent schedule; Python-path dual round 202 ms

C.1 Execution on a second chip (M2)

The probe replication below uses the 1,593-operation log-normalizer head variant. The HeteroEngine head was subsequently run on the same M2 at $Q{=}2, K{=}2$; see the paragraph on record 330 below.

The second device is a Mac mini M2 (Mac14,3; 8-core CPU, 16-core ANE, 16 GB unified memory, macOS 26.4.1), with the same coremltools 9.0, probe scripts, and state scripts as the M4 Pro rows (records 303–305; raw logs in the artifact). The M4 Pro column was re-measured with the identical tool on the same day. Table 14 reports the paired probe results.

Table 14: Execution constraints on the M4 Pro and M2, measured with identical probes.

Quantity	M4 Pro	M2
State-resident 67 MB predict (p50)	4.3 ms	8.2 ms
Compute-plan placement (state program)	49/49 ANE	49/49 ANE
`write_state` refit	0.31 ms/MB	0.30 ms/MB
Same 67 MB streamed as inputs	646 ms	refused by ANE runtime
Ingestion slope (same tool)	9.7 ms/MB	8.7 ms/MB
State-count boundary	52 pass; 54 fail	54 pass; 56 fail
Single-layer probe placement	ANE from 32 rows	ANE at every probed size
Default `.all` vs. forced-CPU (single-layer probe)	79 vs. 0.96 ms	57 vs. 1.45 ms

Training-system pair on the M2 (record 312). The workspace was rebuilt on the M2 with the same pinned toolchain; the compiled model's weight blob is byte-identical (SHA-256) to the M4 Pro's, its compute plan is likewise 1,593/1,593 ANE-preferred, and the dual-versus-sequential state-hash gate passes. Paired endpoints at $Q{=}2, K{=}2$ (300 updates, common GPU evaluator, seeds 33–37) are $+.0465$, $+.0402$, $+.0567$, $+.0565$, $+.0336$, with mean $+0.0467$ (sd 0.0101, 90% CI $[+0.0370, +0.0564]$, 5/5 positive), against $+0.0441$ (sd 0.0094, five seeds, same configuration and head; record 292) on the M4 Pro. The log-normalizer penalty thus reproduces on the second chip with a similar observed mean. The initial ANE-evaluated CE differs slightly between chips (4.3718 versus 4.3736), so paired contrasts use the common GPU evaluator within each device. A ten-update M2 gate run took 276 ms per round; this is a validation run rather than a replicated throughput estimate.

Table 15: **Post hoc energy decomposition of the batch-matched pair** ($Q{=}4, K{=}6$ against GPU $M{=}10$), computed separately from each chip's panel. Energies are `powermetrics` rail estimates; times are host-timer medians.

Quantity	M4 Pro	M2
Throughput ratio	1.478	1.493
Energy-per-token ratio E	0.701	0.821
r_A (ANE / GPU energy per token)	0.172	0.442
g_G (hybrid / GPU-only GPU energy per token)	1.038	1.024
ANE energy per microbatch (J)	0.174	0.388
GPU-only GPU energy per microbatch (J)	1.011	0.878
ANE time per microbatch (ms)	54.2	72.5
GPU-only time per microbatch (ms)	35.0	88.3
ANE occupancy per update (upper bound)	0.89	0.48

HeteroEngine head on the M2 (record 330). The corrected head was then run on the same M2, reusing the seeds, data tape and evaluator of record 312. The workspace was rebuilt and the model recompiled on the M2; its `model.mil` is byte-identical to the M4 Pro's HeteroEngine program, and its compute plan is 1,586/1,586 ANE-preferred against 1,593/1,593 for the log-normalizer variant. The dual-versus-sequential state-hash gate passes. Across seeds 33–37 at $Q{=}2, K{=}2$ and 300 updates, the paired endpoint difference is $+0.0000222$ CE (90% CI $[-0.0009747, +0.0010192]$), a $+0.0022\%$ perplexity difference (90% CI $[-0.097\%, +0.102\%]$), inside the prespecified $\pm \ln(1.01)$ margin. Against the log-normalizer endpoints on the same seeds the paired change is -0.0467 CE (90% CI $[-0.0560, -0.0374]$), negative for all five seeds. Three controls bound the attribution: the GPU-only arm reproduces the record-312 GPU state hashes bit for bit on all five seeds; the current driver running the log-normalizer program reproduces the record-312 hybrid state hash bit for bit; and a repeated hybrid run at a different time and machine load is bit-identical. The comparison therefore isolates the head program. This record makes no throughput, power, or energy claims, since other user processes were resident during these runs; the record-332 panel below measures these quantities under a registered protocol.

M2 throughput and energy panel (record 332). A preregistered panel on the same M2 first swept this chip's GPU-only batch size ($M \in \{4, 8, 10, 12, 16, 20, 24\}$, three rotated passes; the highest median was $M{=}24$ at 2,786 tok/s, and $M{=}20$ lay within the prespecified 1% tie band, so the tie rule selected $M{=}20$). The panel then measured the hybrid's CPU+GPU+ANE energy per token against batch-matched GPU-only controls: one finite `powermetrics` sampler per cell, five cyclically ordered blocks, seed 7, updates 5–124, and a 20-second idle window before and after each cell. Idle-window gates voided 4 of 25 cell-runs; every surviving cell retains at least four blocks. As geometric means of per-block ratios with 90% t intervals, $Q{=}4, K{=}6$ against GPU $M{=}10$ has an energy ratio of 0.821 $[0.819, 0.824]$ and a throughput ratio of $1.493\times$; $Q{=}2, K{=}2$ against GPU $M{=}4$ has 0.782 $[0.778, 0.785]$ and $1.590\times$; and against the swept $M{=}20$ the hybrid has 0.796 $[0.759, 0.835]$ and $1.479\times$. Both matched comparisons pass the preregistered direction test: the hybrid uses less energy per token than the GPU-only control. These are repeat-measurement intervals at one seed; load power varies little across cells (9.0–12.0 W), so the energy ratios largely follow the throughput ratios. M2 quality evidence remains the $Q{=}2, K{=}2$ equivalence of record 330.

Why the M2's energy gain is smaller. A post hoc decomposition analyzes the batch-matched pair ($Q{=}4, K{=}6$ against GPU $M{=}10$), which both chips' panels contain, within each chip's own panel (M4 Pro blocks 1–5; M2 blocks 2, 4 and 5, where both cells passed the gates). The throughput gain is similar on the two chips ($1.478\times$ and $1.493\times$); only energy per token differs (0.701 and 0.821). The energy ratio decomposes exactly as

$$E = s\left[\tfrac{K}{M}\, g_G + \tfrac{Q}{M}\, r_A\right] + c,$$

where r_A is ANE-rail energy per ANE token divided by GPU-rail energy per token in GPU-only training, g_G is the same ratio for the hybrid's GPU tokens, s is the GPU rail's share of GPU-only CPU+GPU+ANE power, and c is the hybrid's CPU term. g_G is similar on both chips, but r_A is

0.172 on the M4 Pro and 0.442 on the M2 (Table 15). Substituting the M4 Pro's r_A into the M2's decomposition lowers its energy ratio from 0.821 to 0.715, closing about 88% of the gap. Most of the difference in r_A comes from ANE energy per microbatch, about $2.2\times$ higher on the M2; GPU energy per microbatch differs by about 15% in the other direction. Both SoCs list a 16-core Neural Engine but belong to different generations: the M2 is built on enhanced second-generation 5-nm technology (Apple Inc., 2022) and the M4 Pro on second-generation 3-nm technology (Apple Inc., 2024a). An M2 ANE microbatch takes longer on the host timer, and its estimated power while active is higher. At this allocation the M2's GPU leg sets the update time and the ANE is occupied for at most about half of each update; its rail power is near zero while it waits, and its energy per ANE token is the same at $Q{=}2, K{=}2$ (1.52 and 1.53 J per 1k tokens), so the imbalance adds no energy. The decomposition locates the difference but does not test a physical cause such as process technology or ANE design. The `powermetrics` manual states that its power values are estimates and should not be used to compare devices; we measured one machine per chip, each on a different macOS release (26.6.2 and 26.4.1).

The M2 runtime refuses the 67 MB input-streamed program outright (hard error, no CPU fallback), while the state-resident program executes. A separate single-layer streamed-weight probe executes on both chips; under default placement it is 40–$82\times$ slower than forced CPU, so explicit placement checks are essential. A fresh build's first load spends several seconds in ANE compilation and reads as "ANE idle" until the caches are warm; all reported cells ran with warm caches.