TECHNICAL WHITE PAPER · METALSCOPE

metalscope: Roofline Analysis for Metal Compute Kernels Against Measured, Not Asserted, Ceilings

An ML-native kernel profiler for Apple silicon: arithmetic intensity computed analytically from GEMM, attention, norm and elementwise shapes; compute and bandwidth roofs measured on the local chip rather than multiplied off a specification; a three-tier timing ladder that degrades instead of extrapolating; and a diff that names no winner when two measured spreads overlap.

Metal / MPS | Swift · No Dependencies | Measured Roofline | Analytic ML Shapes | Static Occupancy | Schema-Versioned Traces

ABSTRACT

Apple's GPU tooling can report what a Metal kernel did. It cannot report how far that kernel is from the best its shape allows on the chip in front of it, because answering that requires an arithmetic intensity a generic profiler cannot obtain, and an honest ceiling, which the published peak-FLOPS figures for Apple GPUs are not. metalscope is a dependency-free Swift tool supplying both: it derives FLOPs and compulsory DRAM traffic analytically from ML kernel-shape annotations, measures the chip's real compute and bandwidth roofs before scoring against them, and places each dispatch on the resulting roofline alongside a static occupancy analysis needing no counters. The evaluation on an M1 Pro includes two measurement bugs the instrument found in itself, a quantification of specification error across two chips, and a diff rule that cut phantom findings from 30 of 36 kernel comparisons to 4 — the residual four published rather than rounded away.

01 Introduction

A competitive hardware claim is only as checkable as the instrument behind it. NVIDIA ships Nsight Compute alongside its GPUs [6], so a claim about a CUDA kernel arrives with a per-kernel roofline, an occupancy figure and a baseline diff a skeptical reader can reproduce. Apple silicon has the hardware argument and no ML-native equivalent of that instrument.

Instruments' Metal System Trace [9] is the closest thing, but it has no notion of a kernel's *shape*: roofline analysis needs arithmetic intensity — FLOPs per byte of DRAM traffic, exactly `2mnk / e(mk + kn + mn)` for an $m \times n \times k$ matmul — and Instruments sees a dispatch with a thread count. Nor chip ceilings: Apple publishes no peak FLOPS, and the figures in circulation — 5.2 TF for an M1 Pro, 10.4 TF for an M1 Max — are community arithmetic of the form *cores × ALUs × 2 FLOP × boost clock*, which Section 3.2 shows overstates achievable fp32 GEMM by roughly 1.5–1.7×, and by *different amounts on different chips*. Nor a diff, though kernel optimisation is a loop whose entire payload is one bit.

metalscope targets that gap and nothing else; Section 6 lists what remains Instruments-only. What it adds is a trust layer, built on the position that a profiler is trusted by default and that this trust is spent every time it prints a number it cannot stand behind.

02 Scientific Background

2.1 The Roofline Model and the Ridge Point

Williams, Waterman and Patterson introduced the roofline model as a bound on achievable performance for a kernel on a given machine [1]. Attainable performance is

$$P_{attainable} = \min(P_{peak}, AI \times B_{peak})$$

where AI is the kernel's arithmetic intensity in FLOPs per byte of DRAM traffic. The two roofs intersect at the **ridge point**, AI = P_{peak} / B_{peak}, and that is the diagnostic payload: a kernel to its left cannot be helped by faster arithmetic, only by moving fewer bytes. The classification turns a stopwatch reading into a direction for the next hour of work, and it depends on both roofs being right.

2.2 Theoretical Versus Empirical Ceilings

The cache-aware roofline [2] models each level of the memory hierarchy rather than DRAM alone; hierarchical roofline work for GPUs [3] and the Empirical Roofline Toolkit [4] established the point that matters most here: where vendor figures are theoretical, *measured* ceilings are the only defensible ones. `calibrate` sits in that tradition — square GEMMs through Metal Performance Shaders [14] for the compute roofs, a STREAM-style triad [5] for the bandwidth roof. It is single-level DRAM deliberately: a half-calibrated cache roofline would be worse than none on a chip whose cache hierarchy is undocumented.

2.3 Arithmetic Intensity and Compulsory Traffic

Counting FLOPs analytically from a model's shape is standard in the ML systems literature [13], but it is rarely wired into a GPU profiler as the *source of truth* for intensity, and that connection is metalscope's contribution. The alternatives are to instrument the shader, which perturbs what is measured, or to read hardware counters, which on Apple silicon do not exist (Section 4.5).

The byte side follows the fused-attention literature: Dao et al. [12] frame a kernel's cost by the traffic it forces to and from high-bandwidth memory, and show that fusing the softmax between the two matmuls removes the $s \times s$ score matrix entirely. metalscope's `attention` shape models the fused kernel — Q, K and V in, O out — so an unfused implementation reports well under one hundred percent. That gap is not a modelling error; it is a measurement of the cost of not fusing.

Those byte counts are *compulsory* DRAM traffic: each input read once, each output written once, assuming perfect on-chip reuse. Real kernels move more, so intensity is an upper bound and roofline placement is conservative in the useful direction — a kernel that looks bandwidth-bound really is. The cost is that measured efficiency can exceed one hundred percent when a working set is partly cache-resident — the clearest argument for the cache-aware extension [2] of Section 6.

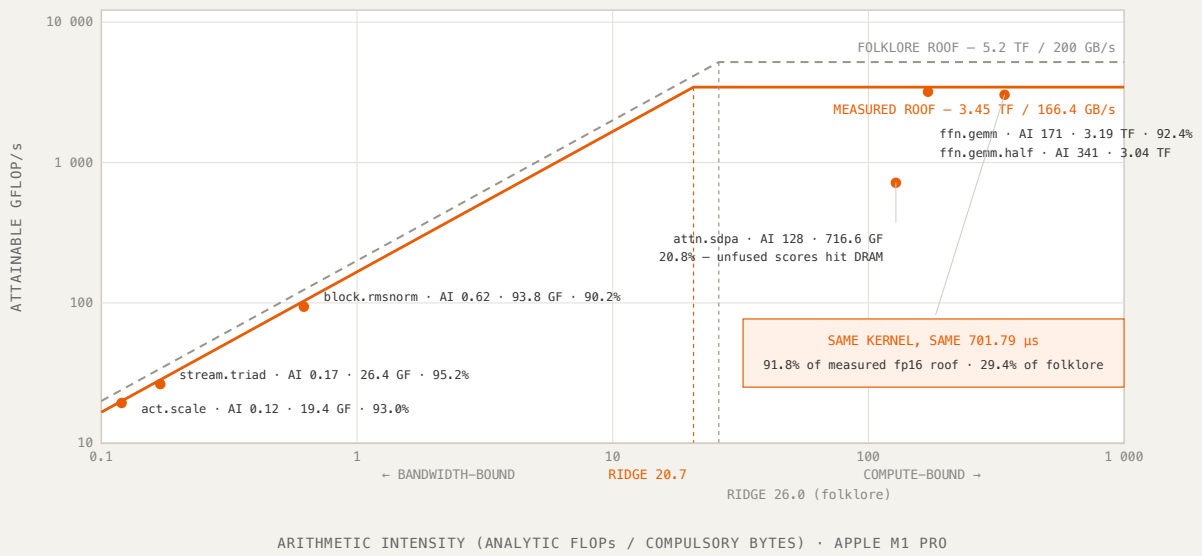


Figure 1. Roofline construction from measured peaks, with the six kernels of `metalscope bench --variant baseline` on it. The solid roof is `calibrate`'s measurement on this M1 Pro; the dashed roof is the spec-sheet arithmetic in circulation. Because the two are wrong by different ratios — compute by 1.5–1.7×, bandwidth by 17% — the ridge point moves, from 20.7 FLOP/byte measured to 26.0 asserted.

03 System Architecture

3.1 Module Structure and Attach Model

metalscope is 4,143 lines of Swift across three targets with no external dependencies. `MetalscopeCore` (1,469 lines) holds the trace schema, shape registry, roofline and occupancy mathematics, repeat statistics, diff and number formatting; it imports no Metal. `MetalscopeCapture` (839 lines) is the opt-in capture API plus the per-counter-set resolvers. The CLI (1,835 lines) provides `info`, `calibrate`, `bench`, `profile`, `report` and `diff`.

metalscope does not inject into arbitrary processes: a target links `MetalscopeCapture`, wraps annotated work in a `capture { }` region, and `metalscope profile -- <cmd>` runs it with `METALSCOPE_TRACE` set. Swizzling was rejected: without dispatch-boundary sampling there is nothing useful to do with an intercepted encoder (Section 3.4).

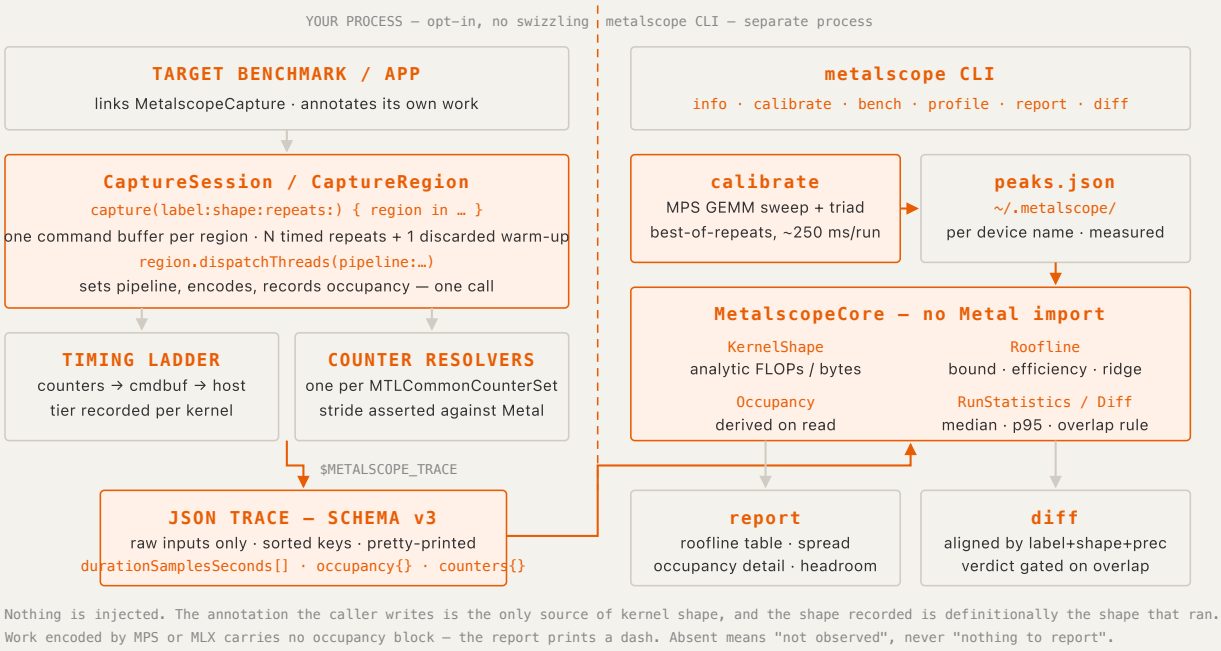


Figure 2. metalscope's two-process structure. Left of the boundary is opt-in library code the target links deliberately; right of it analyses a JSON trace it never produced. `MetalscopeCore` imports no Metal, so every number a report prints can be recomputed with no GPU present — which makes a published trace checkable by a reader who does not own the machine.

3.2 Measured, Not Asserted, Peaks

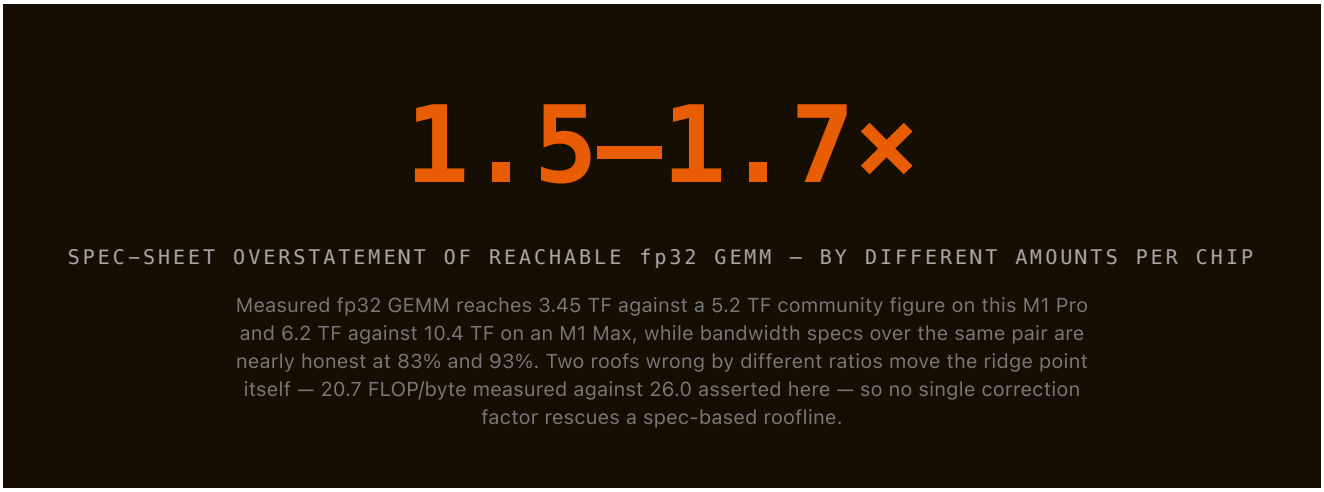
Every efficiency percentage metalscope prints is a fraction of a ceiling, which makes the ceiling the most load-bearing number in the tool. `calibrate` measures it and caches it per device. Two choices mattered more than the benchmark. **Run length:** Apple GPUs ramp their clocks over tens of milliseconds, and the same 1024³ MPS GEMM measures 0.88 TF over four iterations and 3.3 TF over 120, so `calibrate` probes once and auto-sizes to roughly 250 ms of GPU work per run. **Best-of-repeats, not mean:** since throttling and contention only move a sample downward, the maximum converges on what the hardware can sustain while the mean converges on the machine's current mood.

When no measurement exists, metalscope falls back to community figures and stamps every number derived from them **spec-sheet folklore**. On this M1 Pro, and on a Mac Studio M1 Max as an independent second data point:

Ceiling	Measured	Spec-sheet	Ratio
M1 Pro — fp32 compute	3.45 TF	5.2 TF	**66%**
M1 Pro — fp16 compute	3.33 TF	10.4 TF	32%
M1 Pro — memory bandwidth	166.4 GB/s	200 GB/s	**83%**
M1 Max — fp32 compute	6.2 TF	10.4 TF	**60%**

Ceiling	Measured	Spec-sheet	Ratio
M1 Max — memory bandwidth	371.5 GB/s	400 GB/s	**93%**

The two chips agree on the shape of the error, and it is not uniform: bandwidth is nearly honest — 83% and 93%, both ordinary STREAM efficiencies — while compute overstates achievable fp32 GEMM by roughly 1.5–1.7× on both. A fudge factor cannot repair that, since two roofs wrong by different amounts move the ridge point.



The fp16 row deserves comment. The 10.4 TF figure assumes double-rate half precision, which the measurement does not find: fp16 GEMM through MPS reaches 3.33 TF, essentially the same as fp32. Scoring one trace both ways, the same `ffn.gemm.half` kernel at 701.79 µs reads as **91.8%** of the measured ceiling and **29.4%** of the folklore one — optimal and in no headroom list against the first, the second-worst kernel in the trace against the second.

Calibration has a noise floor worth publishing. Two `calibrate` runs minutes apart on a *loaded* machine returned 3.24 TF / 147.8 GB/s and 3.17 TF / 155.1 GB/s against the idle machine's 3.45 TF / 166.4 GB/s — a 7% spread on compute, 11% on bandwidth. Best-of-repeats mitigates this within a run; an idle machine is still required between them.

3.3 Analytic FLOPs and Bytes from Annotation

A `KernelShape` annotation supplies both terms of arithmetic intensity directly, before the kernel has run:

Shape	FLOPs	Bytes (e = element size)
`gemm(m, n, k)`	`2mnk`	`e(mk + kn + mn)`
`attention(b, h, s, d)`	`4bhs²d`	`4e · bhsd`
`elementwise(n)`	`n`	`2en`
`norm(n)`	`5n`	`2en`

Shape	FLOPs	Bytes (*e* = element size)
`opaque(flops, bytes)`	as given	as given

The `attention` row counts the two matmuls and excludes the softmax as transcendental overhead. `norm` models a two-pass mean and variance over a cache-resident row, hence one read and one write. `opaque` is the escape hatch for kernels with no registered shape. Inferring shapes heuristically was rejected: a profiler that guesses a shape and then reports a confident intensity derived from the guess is the failure mode this tool exists to avoid.

3.4 The Three-Tier Timing Ladder

Metal on Apple silicon supports counter sampling at `.atStageBoundary` only, which shapes the entire capture design: **there is no way to bracket someone else's encoders with counter samples.** metalscope degrades instead, and every kernel record states which tier it reached. The tiers, traced in Figure 3, are `counter-sample-buffer` (per-encoder GPU timestamps from `MTLCounterSampleBuffer` [7]), `command-buffer` (`gpuStartTime`/`gpuEndTime`, when a framework such as MPS created the encoders), and `host` (wall clock around commit and wait — a fallback, not a measurement).

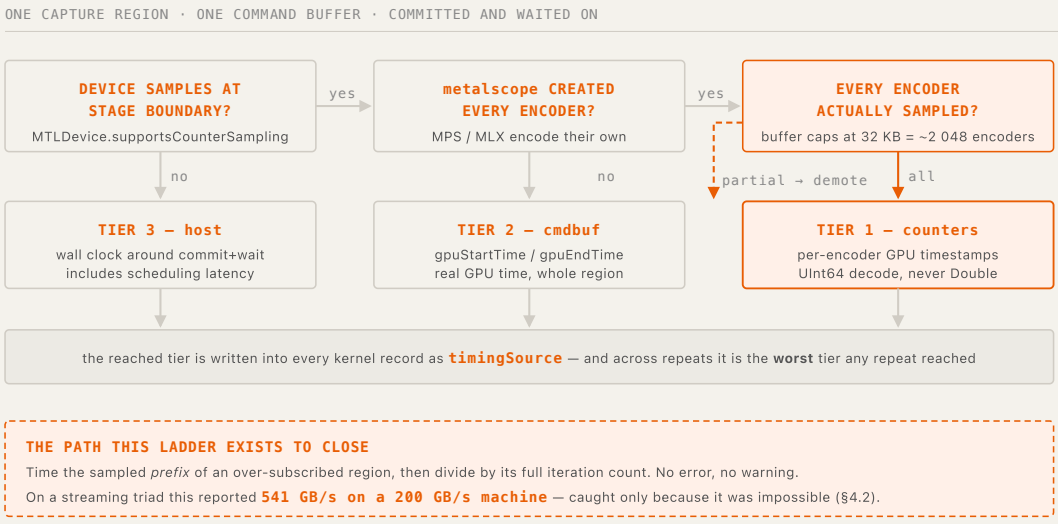


Figure 3. The capture ladder. Each gate demotes rather than extrapolates, and the all-or-nothing rule at the third gate carries the weight: a region whose sample buffer covered only part of its encoders falls to command-buffer timing rather than scaling a partial span.

Tier 2 is still a real GPU-time measurement, because each region is its own command buffer, committed and waited on so its span brackets exactly the annotated work. The critical rule is that **a region claims tier 1 only if every encoder in it was sampled**; partial sampling demotes the whole region, for reasons Section 4.2 documents. Metal caps a counter sample buffer at 32 KB on this chip — 4,096 timestamps, roughly 2,048 sampled encoders — and an over-large request abandons sampling rather than failing. Timestamps

are decoded as `UInt64`, never `Double`: they are nanoseconds since boot, so after roughly 104 days of uptime they exceed 2^{53}, at which point a `Double` rounds them and silently shortens every kernel duration.

3.5 Occupancy Derived on Read

`MTLComputePipelineState` [8] exposes `maxTotalThreadsPerThreadgroup`, `threadExecutionWidth` and `staticThreadgroupMemoryLength`. Paired with the threadgroup size a dispatch used, those four numbers support a static occupancy analysis needing **no counters at all** — working identically on every chip in the matrix, including the ones that expose nothing.

The trace stores *only those raw inputs*; every ratio and the limiter classification are derived on read, which makes a trace whose stored ratios contradict its stored inputs unrepresentable. SIMD groups per threadgroup are rounded **up**: a 100-thread threadgroup still costs four full 32-wide SIMD groups, and the 28 lanes in the fourth idle in every threadgroup, forever.

Metal offers no way to ask an encoder which pipeline it holds, so `CaptureRegion`'s dispatch wrappers set the pipeline, encode and record occupancy in one call — the recorded shape is definitionally the shape that ran. Work encoded by MPS carries no occupancy block, and the report prints a dash.

3.6 Deliberately Not Flagging a Low Occupancy Ratio

The most consequential judgement in the tool is a negative one: **a low threadgroup-occupancy ratio is not a finding, and metalscope does not report it as one.** 256 threads out of a 1024-thread pipeline ceiling is 25% "occupancy" and a perfectly good shape on Apple silicon. A profiler that flagged it would fire on nearly every well-written kernel, and the response to a section that is mostly false positives is to stop reading it.

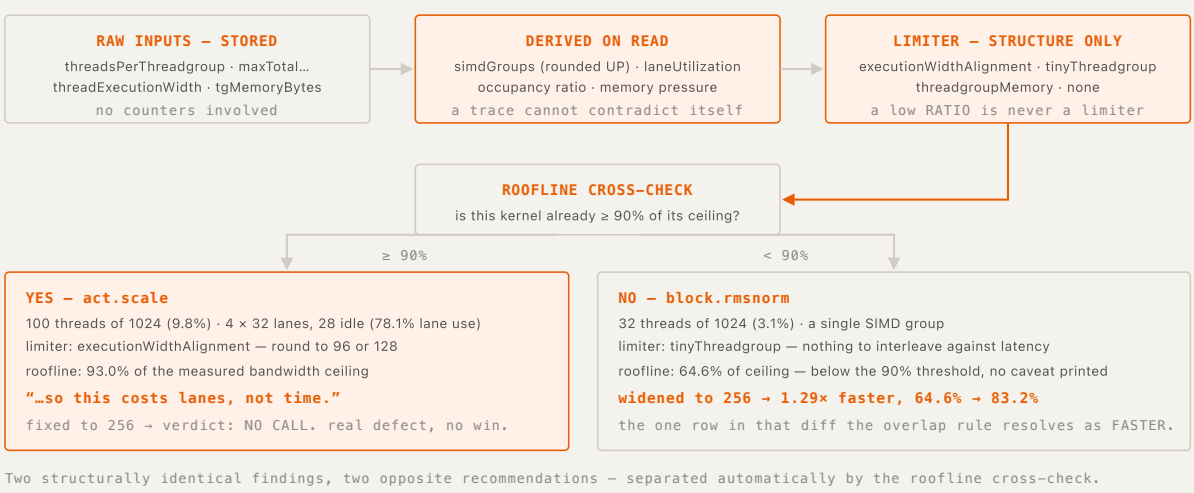


Figure 4. The occupancy analysis and its roofline cross-check. Both kernels carry a genuine structural limiter; what differs is whether fixing it can buy anything, and only the measured roofline can answer that.

Three things are flagged instead, all structural facts rather than ratios. `executionWidthAlignment`: the threadgroup is not a multiple of the SIMD width, so lanes idle. `tinyThreadgroup`: a single SIMD group, nothing to interleave against latency. `threadgroupMemory`: over half the device limit. A kernel at 3.1% occupancy *is* flagged, because 32 threads is one SIMD group, while one at 25.0% is not. The number is context; the structure is the verdict.

A second cross-check, traced in Figure 4: before printing an occupancy hint the report places the kernel on the roofline, and if it is already at or above 90% of its ceiling the hint gains the clause *"so this costs lanes, not time."*

3.7 Schema-Versioned Traces

One JSON schema, currently version 3, is shared by capture, `report` and `diff`, pretty-printed with sorted keys. Every addition since version 1 is optional, so older traces still read; readers reject a version newer than they understand. Version 3 stores the per-repeat duration samples and nothing derived from them. Two conventions are load-bearing: the median is the *lower* of the two middle samples on an even count, and the 95th percentile is nearest-rank, so both numbers a report prints are runs that actually happened.

Apple's three common counter sets resolve to three different C structs — one field, six and eight — and each resolver takes its stride from the real Metal struct, with tests asserting that stride equals the counter count times the stride of `UInt64`, so the day Apple adds a field the suite fails rather than the decode sliding by eight bytes. A counter the GPU declined to write is `MTLCounterErrorValue` and is **omitted**: a missing key means "not written", and must never be readable as zero.

04 Verification Strategy

All measurements here are on an Apple M1 Pro (16 GB, macOS 26.5.1 build 25F80) unless stated otherwise, with metalscope 0.1.0 built via `swift build -c release`.

4.1 Test Suite and Coverage

194 XCTest cases pass across ten suites, weighted towards where the correctness risk lives: the largest is `TextTableTests` (34), because every number a reader sees passes through it. Coverage of the two library targets is **91.16% region / 93.96% function / 96.57% line**, against 73.20% / 79.11% / 84.17% across 102 cases before this round of test work. The largest remaining gap is hardware-bound: the auxiliary-counter resolution path cannot execute on a chip exposing only `timestamp`, so its aggregation *rules* are covered synthetically.

4.2 Two Measurement Bugs Found by Self-Application

metalscope was used to profile its own benchmark workloads throughout development. Two bugs surfaced that way, each now the reason for a design rule.

The counter-prefix bug: 541 GB/s on a 200 GB/s machine. When a region encoded more encoders than its sample buffer could cover, the timing path used the span of the *sampled prefix* and divided it by the full

iteration count. The result was a streaming triad reporting 541 GB/s on a machine whose bus tops out at 200 GB/s — roughly 3× reality, with no error and no warning.

It was caught only because it was physically impossible. Had the workload been a GEMM, the same 3× error would have produced an efficiency figure that looked merely excellent, and it would have shipped. The fix is the all-or-nothing tier-1 rule of Section 3.4.

The clock-ramp bug: 0.88 TF or 3.3 TF, same GEMM. The first `calibrate` ran a fixed small number of iterations and reported 0.88 TF fp32 for a 1024^3 MPS GEMM; the same GEMM over 120 iterations reports 3.3 TF. This one is the more insidious, because 0.88 TF is not absurd — it is 17% of the folklore ceiling, rationalisable as MPS overhead on small matrices. The fix is the probe-then-auto-size design of Section 3.2: a harness which does not control run length is measuring the power manager.

The effect is not confined to calibration. The example transformer block initially ran 32 iterations per region, and its MPS QKV projection measured 44.3% of the compute ceiling. Raising the two microsecond-scale kernels *ahead* of it from 32 to 2,048 iterations — leaving the GEMM's own count untouched — moved that same GEMM to **95.2%**. A kernel's measured efficiency depends on what ran before it, and the `spread` column says as much: that GEMM reads 198.1–324.7 µs.

4.3 The Observer Effect, Re-Measured Against the Project's Own Claim

Stage-boundary sampling is not free. Ten back-to-back dispatches of an identical kernel, with per-encoder sampling and with none, best of twelve runs each:

Buffer	Unsampled	Sampled	Overhead	Range over six repetitions
1 MB	221.1 µs	333.4 µs	+50.8%	49–129%
4 MB	601.7 µs	793.0 µs	+31.8%	32–49%
8 MB	1117.5 µs	1358.6 µs	+21.6%	—
16 MB	2180.8 µs	2441.4 µs	+12.0%	12–18%
32 MB	3934.9 µs	4407.5 µs	+12.0%	7–12%
64 MB	8080.1 µs	8158.8 µs	+1.0%	1–6%

The overhead is an approximately fixed per-encoder cost, and it is real GPU work rather than an artefact: counter timing and command-buffer time for the *same sampled run* agree to within a microsecond. This re-measurement disagreed with the project's own documentation, and the disagreement was published rather than reconciled: the figure previously recorded — roughly 35% at 4 MB, nothing beyond noise at 32 MB — holds at 4 MB but is too optimistic at 32 MB, where six runs show a consistent 7–12%.

4.4 Repeats, Spread, and Refusing to Call a Winner

A single run per kernel is not a measurement. Timing one region and dividing by its iteration count controls the clock ramp but says nothing about run-to-run variance, which on a microsecond kernel is the

larger term. Four captures of an unchanged `bench --variant baseline`, one timed run each, against the same captures at five repeats:

Kernel	Single run, 4 captures	Ratio	5 repeats, 4 captures	Ratio
`ffn.gemm`	156.8 – 349.9 µs	2.23×	104.7 – 108.8 µs	1.04×
`ffn.gemm.half`	124.6 – 289.4 µs	2.32×	94.3 – 98.7 µs	1.05×
`attn.sdpa`	170.1 – 333.5 µs	1.96×	172.6 – 178.3 µs	1.03×
`block.rmsnorm`	58.4 – 92.8 µs	1.59×	58.9 – 61.9 µs	1.05×
`act.scale`	51.6 – 67.2 µs	1.30×	49.6 – 51.0 µs	1.03×
`stream.triad`	64.6 – 80.7 µs	1.25×	58.7 – 62.9 µs	1.07×

A diff of two of those single-run captures — of *identical* code — reports `ffn.gemm` 1.95× faster, `ffn.gemm.half` 0.43×, `attn.sdpa` 0.51×, and moves `block.rmsnorm` 32 percentage points of its bandwidth ceiling. Every one of those numbers is a stopwatch reading presented as a result.

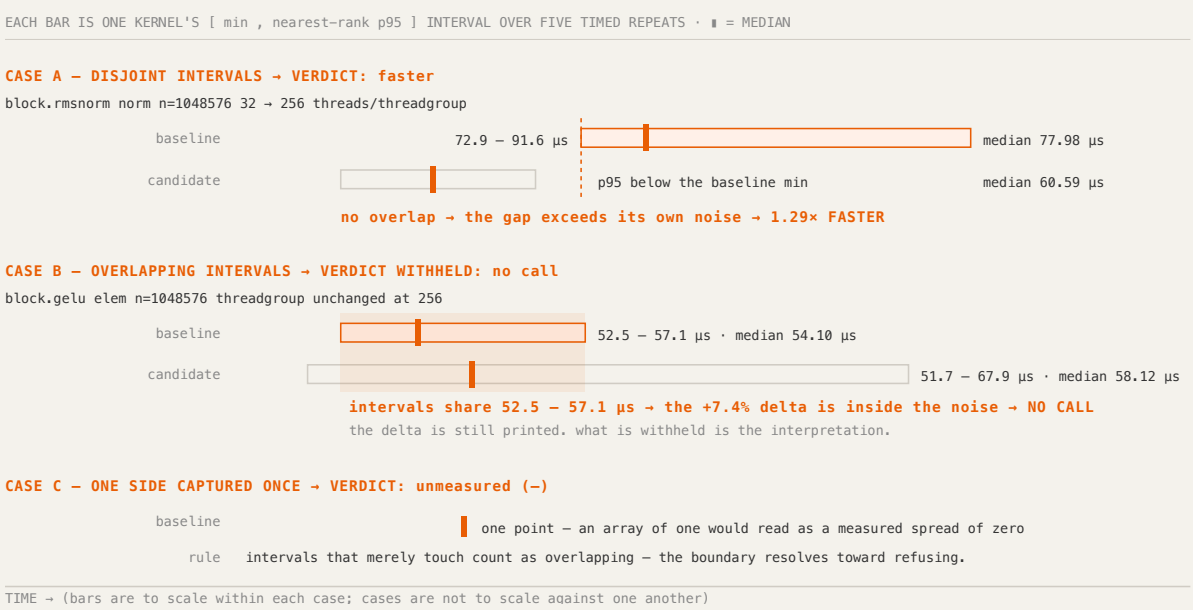


Figure 5. The refusal rule. A verdict is issued only when the two `[min, p95]` intervals are disjoint; overlap yields `no call`, and a single-run capture on either side yields no verdict at all. The interval runs to the nearest-rank p95 rather than the maximum, so one scheduler hiccup does not swallow every real result.

The fix has three parts, and the third is the one that matters. **Repeat, and warm up first**: `bench --repeats N` (default five) runs N timed regions per kernel preceded by one discarded region at the same iteration count, so no timed sample pays the clock ramp — on a 64 MB working set this collapses `ffn.gemm`'s across-capture spread from 48.8% to 1.6%. **Record the samples, derive the summary**: schema v3 stores every per-invocation sample, and `report` prints the median with a min-to-p95 spread column, so a wide

spread discounts the whole row. **Refuse the call**: `diff` compares medians and withholds a verdict when the two intervals overlap, printing the rule beside the table so a reader can check it.

30 → 4

OF 36 KERNEL COMPARISONS THAT WOULD HAVE READ AS A RESULT

Measured over all six pairings of four captures of an *unchanged* microsecond-scale baseline: single-run comparisons showed a greater-than-5% delta in 30 of 36 kernel pairs, worst 132%. At five repeats behind a discarded warm-up, 4 of 36 produced a verdict at all, worst median gap 7.2%. The rule narrows the false-positive rate by roughly an order of magnitude and does not abolish it — four marginal calls out of thirty-six is the number, not zero.

The `occupancy changes:` section of a diff needs no repeats, because it reports a change in shape rather than a stopwatch reading. Profiling the same transformer block at 512 KB instead of 4 MB moved every roofline efficiency in the table (21.8% to 64.6%, 44.3% to 95.2%, 9.8% to 41.0%, 16.5% to 93.2%) while the `tinyThreadgroup` limiter appeared identically in both. Structural facts survive bad benchmarking in a way timing does not.

4.5 The Counter Matrix as Community Infrastructure

Which counter sets `MTLDevice.counterSets` returns is a per-chip, per-OS fact nobody publishes. Apple documents `MTLCommonCounterSet` as three sets; what a device offers is a separate question, answered in ten seconds by `metalscope info --json`. The matrix collects those answers — measured, never inferred.

Chip	OS	Counter sets	Sampling	Tier	TG mem	Source
Apple M1 Pro	macOS 26.5.1 (25F80)	`timestamp`	stage	1	32 KB	measured
Apple M1 Max	macOS 26.5.1	`timestamp` only	?	?	?	**partial**
Apple Paravirtual device	GitHub `macos-latest`	*none*	dispatch, blit — *not* stage	2	?	measured, CI probe
M1 / M1 Ultra · M2 · M3 · M4 · M5	—	—	—	—	—	*unfilled*

The **M1 Max row is deliberately partial.** A second machine confirmed the same timestamp-only counter situation, and that single fact is the one recorded; the question marks are not copied from the M1 Pro row.

Sampling boundaries and the threadgroup-memory limit are per-chip facts, and a table whose premise is *measured, not documented* cannot infer cells from a sibling chip without becoming the folklore it replaces.

The **paravirtual row is the ladder's worst case, and it held.** The GitHub-hosted runner presents a device that is the mirror image of every real Apple part here: *no* counter sets at all, while supporting dispatch- and blit-boundary sampling and not stage boundaries. Capture falls to tier 2 and the stage-sampling tests skip cleanly. Resolvers for the two sets no shipping Apple part exposes are unit-tested synthetically, so a chip offering them populates the counter field with no code change.

05 Application Domains and Economic Analysis

5.1 Honest Measurement as an Economic Precondition

The economic argument for machine learning on Apple silicon is an owned-hardware argument: a machine already bought, whose marginal cost is electricity, against rented accelerator time. It holds only if the owned machine is fast enough, and "fast enough" is a claim a reviewer has to be able to check.

Measurement quality is therefore an economic input, and Section 3.2 quantifies how wrong the available denominators are. The consequence is not a miscalibrated number but a misallocated week: the `ffn.gemm.half` reading that says *move on* and the one that sends an engineer after a two-thirds gap that does not exist are the same kernel at the same duration. The cost side is negligible — `calibrate` takes ten seconds per machine, cached thereafter.

5.2 The Instrument Behind the Sibling Projects

The scope of the claim here is narrow and worth stating precisely. metalscope's calibrated ceilings are what the triton-metal compiler project's ratios are fractions of: its 2048 GEMM at 4.64 TFLOP/s is quoted as 75% of the *measured* 6.2 TF M1 Max peak rather than of the 10.4 TF spec figure — a strictly less impressive framing of the same result. The mccl collective-communication numbers are *not* produced by metalscope; they come from `mcclbench`, and what they share with this paper is the reporting standard, not the instrument.

5.3 Where This Sits Against Apple's Own Direction

WWDC 2026 gave Metal TensorOps [10], an MSL API for tensor mathematics reaching a neural accelerator that M5 places directly in each shader core, alongside the other GPU pipelines. That is a *second* arithmetic ceiling on one chip, selected by the path a kernel compiles to — so percentage-of-peak means nothing there without saying *which* peak, and the 1.5–1.7× specification error becomes a floor on the problem rather than the whole of it. M5-class silicon makes calibration more necessary, not less.

Apple's own WWDC 2026 profiling work went elsewhere: session 388 [11] is frame-shaped — frames per second, frame interval, MetalFX — with no arithmetic intensity, no measured ceiling and no kernel diff. There is, at the time of writing, no Apple measured-ceiling story for compute kernels. The lever this creates is a TensorOps-aware peak probe inside `calibrate`, so a chip is calibrated on both paths and each

kernel scored against the ceiling it uses. That is stated at specification level only: no M4 or M5 part was available for this work, and nothing here is measured on that hardware.

06 Limitations and Roadmap

Every non-timestamp counter set is hardware-blocked. No part metalscope has run on — M1 Pro and M1 Max, both on macOS 26.5.1 — exposes `stageutilization` or `statistic`, so the counter field is absent from every real trace.

There is no measured occupancy, and stalls stay Instruments-only. The classic occupancy figure — threadgroups actually resident per core — requires a per-core thread and register budget Apple does not publish; metalscope reports what fits inside the threadgroup-memory limit, labels it an upper bound, and stops. Instruction-level stalls and bank conflicts are not deferred features: Metal exposes no path to them.

The counter matrix has one fully measured row. The M1 Max row's blank cells stay blank, and the M4 and M5 rows — the highest-value missing ones — stay unknown rather than estimated.

A diff can still call a marginal winner. The refusal rule narrows the false-positive rate rather than removing it: 4 of 36 comparisons of an unchanged baseline still produced a verdict, the widest on a 7.2% median gap. No threshold rule distinguishes a genuine 5% win from an unlucky pair of five-sample intervals.

The roofline is single-level, and shape annotation is manual. A cache-aware or hierarchical roofline [2] [3] would explain the above-100% efficiencies directly, but that needs per-level bandwidth calibration — a research question on a chip whose cache hierarchy is undocumented. For annotation a middle path exists: a Metal-library-aware front end could propose annotations from function names and buffer sizes for a human to confirm.

07 Conclusion

The useful question about a GPU kernel is not what it did but how far it is from the best its shape allows on the chip in front of you. Answering it requires an arithmetic intensity derived from knowing what the kernel computes, and a ceiling derived from measuring the hardware rather than multiplying its specifications.

The recurring theme of its design is a refusal to fill gaps with plausible numbers. When the timing path cannot cover a whole region, it degrades a tier and says so. When peaks have not been measured, every percentage derived from them is labelled folklore. When two spreads overlap, the diff prints the delta and withholds the interpretation. Each rule costs information; each exists because the alternative is a number that reads as knowledge and is not. The two bugs of Section 4.2 are that in miniature: both were wrong by roughly 3×, and only one was caught by inspection — because 541 GB/s on a 200 GB/s machine is impossible, while 0.88 TF on a nominally 5.2 TF machine is merely disappointing.

The wider claim is that this is infrastructure an alternative to CUDA needs rather than a convenience for one codebase. On Apple silicon the measured-versus-specification gap is large enough, and unevenly enough distributed across chips, that a specification-based roofline cannot be corrected into an honest one, and M5's second arithmetic ceiling widens it. A profiler is trusted by default; that trust is the whole of its value, and it is spent every time it prints a number it cannot stand behind.

// References

[1] Williams, S., Waterman, A., and Patterson, D., "Roofline: An Insightful Visual Performance Model for Multicore Architectures," Communications of the ACM, vol. 52, no. 4, pp. 65–76, April 2009.

[2] Ilic, A., Pratas, F., and Sousa, L., "Cache-aware Roofline Model: Upgrading the Loft," IEEE Computer Architecture Letters, vol. 13, no. 1, pp. 21–24, 2014.

[3] Yang, C., Kurth, T., and Williams, S., "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system," Concurrency and Computation: Practice and Experience, vol. 32, no. 20, 2020.

[4] Lawrence Berkeley National Laboratory, "Empirical Roofline Toolkit (ERT)," Computational Research Division. https://bitbucket.org/berkeleylab/cs-roofline-toolkit

[5] McCalpin, J.D., "Memory Bandwidth and Machine Balance in Current High Performance Computers," IEEE Computer Society Technical Committee on Computer Architecture Newsletter, pp. 19–25, December 1995.

[6] NVIDIA Corporation, "Nsight Compute Documentation" — roofline and occupancy sections, 2024. https://docs.nvidia.com/nsight-compute/

[7] Apple Inc., "MTLCounterSampleBuffer," Metal — Apple Developer Documentation. https://developer.apple.com/documentation/metal/mtlcountersamplebuffer

[8] Apple Inc., "MTLComputePipelineState," Metal — Apple Developer Documentation. https://developer.apple.com/documentation/metal/mtlcomputepipelinestate

[9] Apple Inc., "Metal System Trace," Instruments — Apple Developer Documentation.

[10] Apple Inc., "Explore Metal 4 tensors and TensorOps," WWDC 2026, session 330. https://developer.apple.com/videos/play/wwdc2026/330/

[11] Apple Inc., WWDC 2026, session 388 (GPU profiling: frame interval and MetalFX). https://developer.apple.com/videos/play/wwdc2026/388/

[12] Dao, T., Fu, D.Y., Ermon, S., Rudra, A., and Ré, C., "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness," Advances in Neural Information Processing Systems 35, 2022.

[13] Kaplan, J., et al., "Scaling Laws for Neural Language Models," arXiv:2001.08361, 2020 — appendix on analytic per-token FLOP accounting from model shape.

[14] Apple Inc., "Metal Performance Shaders — MPSMatrixMultiplication," Apple Developer Documentation. https://developer.apple.com/documentation/metalperformanceshaders