Edge LLM Inference Across Commodity Accelerators: Measured AMD, NVIDIA, and Apple Performance, and hanzo-engine at llama.cpp Decode Parity on Every Backend

Zach Kelling

Hanzo AI

June 2026

Abstract

This paper reports *measured* single-GPU inference performance for one model (**an 8B dense transformer at Q8_0**) across three commodity edge accelerators — AMD Ryzen AI Max+ 395 "Strix Halo" (Radeon 8060S iGPU, gfx1151), NVIDIA GB10 "Grace Blackwell" (sm_121), and Apple M4 Max — comparing **hanzo-engine** against `llama.cpp` on each, same model, same metric (512-token prefill throughput `pp512`; single-stream decode throughput). Every number here is measured on the named machine; AMD figures are carried from companion paper 01 (measured 2026-06-08), NVIDIA and Apple figures were measured fresh on 2026-06-17. We find two things, both with the numbers to back them. First, **hanzo-engine reaches decode parity with `llama.cpp` on all three backends** (0.98×/1.04×/0.99×), while prefill trails by a consistent, backend-dependent margin (0.76–0.91×). Second, **the fastest edge box flips with the workload**: NVIDIA GB10 wins prefill (compute-bound), Apple M4 Max wins decode by ≈2.5× (bandwidth-bound), and the GB10's Blackwell decode path is the slowest of the three despite the strongest prefill. We are explicit about what is *not* here: a single model at a single quant, single-stream, no batching/serving sweep, and AMD not re-run on this pass. Numbers that are projections or microbenchmarks are labelled as such; there are none we invented.

1 Why these three boxes

The thesis of the edge-inference program is that the interesting target is not a datacenter H100 but a *commodity box a customer already owns*: a unified-memory APU, a small Grace-Blackwell dev kit, or a MacBook. All three share LPDDR-class unified memory (no discrete VRAM tier to hide behind), which makes decode — a pure streaming read of every weight per token — brutally honest: decode throughput is memory bandwidth divided by model bytes, full stop. Prefill, by contrast, is a compute-bound GEMM on the matrix/tensor cores. A backend can win one regime and lose the other; this paper shows exactly that.

Hardware (as measured).

- **AMD Strix Halo** — Radeon 8060S iGPU, gfx1151 (RDNA3.5), 40 CU, native Windows + ROCm 7.1; measured streaming bandwidth ≈231 GB/s (paper 01).
- **NVIDIA GB10** — Grace-Blackwell, sm_121, CUDA 13, 121 GB unified; Linux aarch64.
- **Apple M4 Max** — 137 GB unified, Metal; `llama.cpp` via the `BLAS,MTL` backend.

2 Method

One model throughout: **an 8B dense transformer at Q8_0** (36 layers, n_{embd} 4096, n_{ff} 12288, 32 query / 8 KV heads, head dim 128; ≈8.11 GiB resident). The 8-bit quant is the honest "no accuracy excuse" point — it is the largest of the common quants, so decode is bandwidth-bound and the comparison is purely about kernel/streaming efficiency, not about a cheaper format hiding a slower engine.

- **Prefill** = `pp512`: tokens/s processing a 512-token prompt (compute-bound GEMM).
- **Decode** = single-stream token generation, tokens/s (bandwidth-bound mat-vec). For `llama.cpp` this is `tg128`; for hanzo-engine it is the bench command's 128-token decode at depth 4. Both are single-stream token-generation rates at shallow KV-cache depth, treated as comparable decode throughput (not identical-length runs).

- All layers forced resident on the accelerator (`-ngl 99` for `llama.cpp`; hanzo auto-maps all 36 layers to the GPU). *This matters*: on a unified-memory box the device mapper can misread shared RAM and silently offload layers to the CPU, which tanks decode and invalidates the comparison.
- hanzo-engine built `--release` on each box (the NVIDIA binary was rebuilt release for this pass; a debug build is not a fair number). `llama.cpp` is the stock HIP/CUDA/Metal build.

3 Results

Table 1: The 8B model at Q8_0, single GPU. Prefill = `pp512` t/s; Decode = single-stream t/s. AMD from paper 01 (2026-06-08); NVIDIA + Apple measured 2026-06-17.

	Prefill (t/s)			**Decode (t/s)**		
Backend	llama.cpp	hanzo	ratio	llama.cpp	hanzo	ratio
AMD Strix Halo (gfx1151)	1176	1012	0.86×	25.0	24.5	0.98×
NVIDIA GB10 (sm_121)	2155	1630	0.76×	19.6	20.3	**1.04**×
Apple M4 Max (Metal)	873	792	0.91×	50.4	49.9	0.99×

Two cross-cutting facts fall straight out of the table.

1. The fastest box depends on the workload. On *prefill*, NVIDIA GB10 leads (2155 t/s, 1.8× the APU, 2.5× the M4 Max) — the Blackwell tensor cores dominate the compute-bound GEMM. On *decode*, the order inverts completely: **Apple M4 Max wins at 50.4 t/s**, about 2× the Strix Halo and 2.5× the GB10. Decode is bandwidth-bound, and the M4 Max's wide unified memory (≈50.4 t/s × 8.04 GB of weights ≈ 405 GB/s effective) simply moves weights faster. Strikingly, the **GB10 — the strongest prefill box — is the *slowest* at decode** (≈19.6 t/s ⇒ ≈160 GB/s effective): its LPDDR bandwidth and the current `llama.cpp` Blackwell decode kernels leave it bandwidth-starved. A prompt-heavy / RAG workload prefers the GB10; an interactive chat / long-generation workload prefers the M4 Max. This is the central practical result for choosing edge hardware.

2. hanzo-engine is at decode parity with `llama.cpp` everywhere. hanzo's decode is 0.98×, 1.04×, and 0.99× of `llama.cpp` on AMD, NVIDIA, and Apple respectively. We read all three as parity: these are best-of-3 single-stream runs (one warmup) without a collected variance estimate, so the GB10's 1.04× is within run-to-run noise, not a measured win. Since decode is the bandwidth wall, this says hanzo's per-format decode mat-vec is streaming at the hardware ceiling on all three backends. Prefill trails by 0.76–0.91×: smallest on Metal (0.91×) and largest on NVIDIA (0.76×), where `llama.cpp`'s CUDA GEMM is most heavily tuned. The prefill gap is real engineering work (matrix-core GEMM scheduling), not a rounding error — the same honest conclusion paper 01 reached on AMD.

4 The edge/commodity angle: smaller models

For latency-sensitive on-device use the 4B tier is the sweet spot. Measured `llama.cpp` (2026-06-17), a 4B dense transformer at Q8_0:

Backend	Prefill pp512 (t/s)	Decode (t/s)
NVIDIA GB10	3913	36.0
Apple M4 Max	1492	85.3

The same pattern holds and sharpens: the M4 Max decodes the 4B at **85 t/s** (comfortably real-time interactive), while the GB10 again leads prefill (3913 t/s) but trails decode (36 t/s). Decode scales roughly inversely with model bytes (4B Q8 is ≈0.49× the 8B's bytes; M4 Max decode rises 50.4 → 85.3, ≈1.7×), confirming the bandwidth-bound model.

5 Ported vs. novel; feasible vs. aspirational

Consistent with paper 01, we separate what we copied from what is ours, and what is shipping from what is roadmap.

- **Ported (deliberately):** the `mul_mat_q` integer-matrix-core prefill design and the GGUF block formats follow `llama.cpp`/`ggml`; the Metal path uses Apple's BLAS/MPS primitives as `llama.cpp` does.
- **Novel (hanzo):** the unified 1-bit-to-8-bit quant compute core (one kernel family serving the whole quant zoo, bit-exact vs the CPU reference), and a single multi-backend engine (CUDA/ROCm/Metal/Vulkan/CPU) behind one model loader, so the *same* build runs every backend above. The cross-backend decode parity is evidence the core is not over-fit to one vendor.
- **FEASIBLE-now:** closing the NVIDIA prefill gap (0.76→∼0.9×) via Blackwell-tuned GEMM tiles; a Blackwell decode kernel to lift the GB10 off its ≈160 GB/s effective floor.
- **ASPIRATIONAL:** beating `llama.cpp` prefill on any of these boxes; batched / multi-stream serving throughput (this paper is single-stream only).

6 What this paper does *not* claim

Single model (the 8B model at Q8_0) at a single quant; single-stream decode (no concurrency/batching sweep); `pp512`/128-token decode only (no long-context or KV-pressure regime); AMD numbers carried from paper 01 rather than re-measured on this pass; and `llama.cpp` taken as the stock build (not re-tuned per box). Accuracy is not evaluated here — Q8_0 is bit-faithful to the reference, but no task-quality numbers are claimed. Everything reported is a measured throughput on the named hardware; where a figure is derived (effective GB/s) it is computed from the measured t/s and the model byte count, and labelled.

7 Deployment guidance (for practitioners)

The measurements translate into a concrete routing rule for anyone running models on hardware they own:
Two operational points matter as much as the raw numbers. **(a) One binary, every backend.** The same hanzo-engine build runs CUDA, ROCm, Metal, Vulkan, and CPU behind one model loader — there is no per-vendor inference stack to maintain, no rewrite to move a workload from a Mac to a GB10 to an APU mini-PC. **(b) Commodity economics.** All three boxes are consumer/prosumer hardware (a Mac, a small Grace-Blackwell dev kit, an APU mini-PC), not

Workload	Best commodity box	Why
RAG / long prompts / doc ingest	NVIDIA GB10	prefill-bound; 2155 t/s, 1.8–2.5× the others
Interactive chat / agents / long gen	Apple M4 Max	decode-bound; 50 t/s, ≈ 2.5× the GB10
Mixed / cost-first / "runs anything"	AMD Strix Halo APU	cheapest unified box; competitive both regimes

datacenter GPUs, yet they run an 8B model at interactive decode and 800–2150 t/s prefill — the edge is genuinely viable, and the cheapest box (the APU) is within ∼2× of the others on both axes.

8 Related work and contribution

`llama.cpp` is the de-facto standard for edge/commodity LLM inference and is our baseline throughout. Most published edge numbers, however, are either single-vendor (one GPU family) or single-engine; cross-vendor comparisons typically vary both hardware *and* software at once. Our contribution is a controlled measurement: *same* model, *same* quant, *same* metric, the *same* two engines, across three vendors — which isolates hardware from software and surfaces the workload-dependent hardware optimum (prefill→NVIDIA, decode→Apple) and the fact that a single portable engine reaches decode parity with the hand-tuned baseline on all of them.

9 Conclusion

On commodity unified-memory accelerators, hanzo-engine delivers `llama.cpp`-class decode (parity on AMD, NVIDIA, and Apple) from one multi-backend build, with a known and bounded prefill gap. The more actionable result for anyone deploying at the edge is the workload split: buy/route to **NVIDIA for prefill-heavy** pipelines and **Apple silicon for decode-heavy / interactive** ones — and do not assume the most expensive box wins decode, because here it loses.

A Reproducibility

Hardware. AMD Ryzen AI Max+ 395 "Strix Halo" (Radeon 8060S, gfx1151, 40 CU, ≈231 GB/s, native Windows + ROCm 7.1); NVIDIA GB10 (Grace-Blackwell, sm_121, CUDA 13, 121 GB unified, Linux aarch64); Apple M4 Max (137 GB unified, Metal / `BLAS,MTL`).
Software. `llama.cpp` stock HIP/CUDA/Metal builds; hanzo-engine built `--release` per box (git `8c50abb32`). Measurement passes: AMD 2026-06-08 (paper 01), NVIDIA + Apple 2026-06-17.
Models. an 8B dense transformer at Q8_0 (8.11 GiB) and a 4B dense transformer at Q8_0 (3.98 GiB), GGUF.
Commands. Baseline: `llama-bench -m <model>.gguf -ngl 99 -p 512 -n 128`. hanzo: `hanzo bench auto -m <dir> --format gguf -f <model>.gguf` (512-token prefill, 128-token decode, 1 warmup + 3 timed iterations).
Protocol. All layers resident on the accelerator (`-ngl 99` / hanzo auto-map of all 36 layers), single stream, quiet GPU. Effective bandwidth figures are derived as (decode t/s) × (model bytes) and labelled as derived, not measured directly.