

Compute, Not the Wire: An Experience Report on Distributed LLM Inference over Heterogeneous Consumer AMD Radeon, RDNA4, and Mixed ROCm+Metal Clusters*

Bishwanath Bastola, Independent Researcher

June 2026

Abstract

Every published distributed-LLM-inference result is measured on NVIDIA datacenter or consumer GPUs, Apple Silicon, or AMD CDNA accelerators (MI300X). None characterizes the hardware most enthusiasts actually own: old AMD Radeon Pro GPUs driven through Vulkan-over-Metal (MoltenVK), a consumer RDNA4 card (RX 9070 XT) on freshly added ROCm support, and clusters that mix these ROCm and Metal runtimes over a Tailscale mesh. We fill this empirical gap. Building on the Petals-derived weights-stay-local / activations-travel pipeline, we add Sthambha, a compute-aware layer planner, and run dense and Mixture-of-Experts models from 3B to 70B across a deliberately heterogeneous seven-node testbed of commodity desktop workstations plus a low-power single-board coordinator. Our central, measurement-backed reframing is that per-node compute, not the ~ 16 KB/token activation wire, governs throughput: a worker-to-worker push that removes a network hop measured slightly slower than client relay. We report a 4-machine 70B chain at ~ 0.21 tok/s and a catalogue of characterized failure modes: MoE cross-machine coherence collapse, Metal nondeterminism, a page-alignment buffer assert, a 30x speculative-decoding regression on the MoltenVK translation layer, and UDP burst loss. We close with reproducible lessons for getting such a chain toward usable throughput.

0.1 1. Introduction

Distributed inference of large language models (LLMs) has matured from a research curiosity into a small ecosystem of working systems. The lineage that concerns us begins with Petals, which serves 50B–176B-parameter models across geographically distributed volunteer machines by keeping each model’s weights resident on the workers that host disjoint transformer-block ranges and passing only the per-token activations between them [1–3]. Subsequent systems push the same weights-stay-local, activations-travel pipeline toward heterogeneous and low-resource clusters: Prima.cpp runs 30–70B models on mixed CPU/GPU home clusters with a heterogeneity-aware scheduler [4], Parallax allocates contiguous layer blocks proportional to per-device compute capacity [5], and Hetis splits work at module granularity across mixed-architecture GPUs [6]. A parallel set of open community projects — exo [7], Distributed Llama [8], Cake [9], Kalavai [10], Wavefy [11], and the llama.cpp RPC backend [12] — pursue the same goal of pooling everyday devices into an inference cluster.

What unites these results is the hardware on which they are measured. Published throughput and scaling numbers come from NVIDIA datacenter or consumer GPUs, from Apple Silicon (the primary target of exo and the Metal-on-unified-memory path), or — on the AMD side — from CDNA

*Published as a preprint. Cite as: <https://doi.org/10.5281/zenodo.20514966>

datacenter accelerators such as the MI300X, the platform for AMD’s own speculative-decoding results [19, 20]. None of these systems publishes a throughput or failure-mode characterization on the hardware that a large fraction of enthusiasts actually own and that sits at the awkward edge of every vendor’s support matrix: older AMD Radeon Pro GPUs driven through Vulkan, which on macOS necessarily routes through the MoltenVK Vulkan-over-Metal translation layer [31–33]; a consumer RDNA4 card (the RX 9070 XT), which became a formal ROCm target only with release 6.4.1 in May 2025 [24, 25]; and clusters that mix these ROCm and Metal runtimes over a software-defined mesh. This is the unmeasured class, and it is unmeasured precisely because it is fragile: ROCm’s consumer coverage is narrow and recent [24], the AMD quantization stack is still labeled Preview and fails on the RDNA4 architecture gfx1201 [26, 27], and macOS has no native Vulkan at all [33].

0.1.1 1.1 Why this gap matters

The class we characterize is not an exotic corner — it is the default for a builder assembling a cluster from secondhand workstations and a single new consumer card. Treating it as out of scope leaves the practitioner with no measured expectation of what a distributed chain on such hardware will actually deliver, and no catalogue of the failure modes it will hit. Following the case made for negative and experience results in pervasive-computing systems [42], we argue that an honest characterization of this regime — including the failures — is itself the contribution. We adopt the measurement-first, provenance-per-claim discipline of recent characterization work [43]: every number we report is tied to the configuration and source that produced it, and every gap we could not fill is marked [experiment needed] rather than estimated.

0.1.2 1.2 Compute, not the wire

Our central, measurement-backed reframing concerns where the bottleneck actually lies. The folklore of distributed inference treats the network as the scarce resource — Petals optimizes for it over the wide-area internet [3], and decentralized-training systems devote substantial machinery to bandwidth reduction [13–15]. For our deployment that intuition is simply wrong. In a layer pipeline, the only data crossing the wire per token is one hidden-state vector: for a 70B model with hidden size 8192 in fp16 that is 16 KB per token per hop, so a five-hop pipeline at a few tokens per second moves on the order of 400 KB/s in our deployment — negligible on a LAN or a Tailscale mesh. The compute, by contrast, is not negligible. We implemented a worker-to-worker push that removes one client-relay network hop per token, expecting a latency win; in our deployment it measured *slightly slower* than client relay (0.19 vs 0.21 tok/s on a 4-machine 70B chain; 1.37 vs 1.51 tok/s on a toy 2-worker chain), because the per-token compute cost dominates the per-token network cost so thoroughly that removing a hop is in the noise. A fabric micro-benchmark localizes the per-hop cost further: on the RX 9070 XT under ROCm, reading activations back from the GPU (`llama_get_embeddings`, a device-to-host DMA) was 86% of the GPU round trip (3.21 ms of 3.7 ms), while the matmul itself was under 10%. The binding constraint on this hardware is per-node compute, not bandwidth — a reframing that redirects engineering effort from faster links toward compute-aware scheduling and module-asymmetric placement, and that is independently consistent with the heterogeneity findings of Hetis [6].

0.1.3 1.3 This paper

We build on the Petals-derived weights-stay-local / activations-travel pipeline [1, 2] and add Nakshatra, an engine that loads only each worker’s assigned layers from a pre-staged sub-GGUF [34, 35],

together with Sthambha, a compute-aware layer planner that assigns contiguous layer ranges by measured per-node compute — the same water-filling/Halda family as Parallax and Prima.cpp [4, 5] — over a signed, Tailscale-pinned authenticated mesh. We run dense and Mixture-of-Experts models from 1.7B to 70B across a deliberately heterogeneous seven-node testbed of commodity desktop workstations plus a low-power single-board coordinator. The result is an experience and measurement report, not a new state-of-the-art system: we report a 4-machine 70B chain at ~ 0.21 tok/s in our deployment and a catalogue of characterized failure modes — MoE cross-machine coherence collapse, Metal nondeterminism at temperature 0, a page-alignment buffer assert that forces the last worker onto CPU, a $\sim 30\times$ speculative-decoding regression on the MoltenVK translation layer, CPU outpacing Metal on small-active-set MoE, and a multi-chunk UDP burst-loss bug on the activation fabric. Each finding is presented with its provenance, and missing data is marked [experiment needed].

0.1.4 1.4 Contributions

1. **The unmeasured hardware class:** the first reproducible distributed-inference characterization on old AMD Radeon Pro GPUs via Vulkan/MoltenVK, a consumer RDNA4 card (RX 9070 XT) via newly added ROCm, and mixed ROCm+Metal clusters over a Tailscale mesh — the consumer-AMD/Apple-translation regime every prior system benchmarks around but none measures.
2. **The compute-not-wire reframing, empirically grounded:** per-token activation traffic is trivial (16 KB/token/hop for a 70B, ~ 400 KB/s across a 5-hop pipeline), and eliminating a network hop via worker-to-worker push measured slightly slower than client relay — per-node compute, not bandwidth, is the binding constraint.
3. **A catalogue of characterized, reproducible failure modes on this class, each with provenance:** MoE cross-machine coherence collapse (FP drift flipping expert gating), Metal nondeterminism at temperature 0, the page-alignment last-worker buffer assert, the $30\times$ MoltenVK speculative-decoding regression despite high draft acceptance, CPU beating Metal 2–4x on small-active-set MoE, and 75%-to-0% UDP burst-loss mitigation.
4. **Nakshatra + Sthambha:** a weights-stay-local / activations-travel engine with partial sub-GGUF loading and a compute-aware layer planner that assigns layer ranges by measured per-node compute rather than memory or latency, plus a signed, Tailscale-pinned authenticated mesh.
5. **Reviewer-defensible lessons for usable throughput:** where compute-aware scheduling helps versus where the per-node ceiling caps it, the native-ROCm speculative-decoding opportunity (vs. its Metal/MoltenVK failure), and module-asymmetric placement as the path forward — framed as an honest experience/measurement paper with explicit [experiment needed] markers.

The remainder of the paper positions the work against decentralized inference, training, and heterogeneity-scheduling literature (§2); describes the Nakshatra engine and Sthambha planner (§3); documents the testbed, runtimes, and provenance-first methodology (§4); presents the throughput measurements, the compute-not-wire reframing, and the failure-mode catalogue (§5); distills these into lessons and a path toward usable throughput (§6); states limitations and threats to validity (§7); lays out the experiments that would extend the work (§8); and concludes (§9).

0.2 2. Background and Related Work

This work sits at the intersection of four research lines: collaborative layer-pipeline inference, heterogeneity-aware scheduling for consumer clusters, speculative decoding as a single-request throughput lever, and decentralized inference/training networks. We survey each in turn, then state the empirical gap that motivates this paper: none of these lines reports a throughput or failure-mode characterization on the consumer hardware class we measure — old AMD Radeon Pro GPUs driven through Vulkan-over-Metal (MoltenVK), a consumer RDNA4 card (RX 9070 XT) on freshly added ROCm support, and clusters that mix these ROCm and Metal runtimes over a Tailscale mesh.

0.2.1 2.1 Collaborative layer-pipeline inference (the ancestor)

Our architecture descends directly from Petals [1, 2], a BitTorrent-style system in which consumer GPUs each host a disjoint range of transformer blocks and pass activations down a pipeline, so that no single node holds the full model. Petals established the design we inherit — weights stay local, only activations travel — and demonstrated interactive serving of 70B–176B models (Llama-2-70B, BLOOM-176B) across geographically distributed volunteer machines, reaching roughly one generation step per second for the largest model [1]. The follow-up work added fault tolerance and load balancing for unreliable, geo-distributed consumer devices, reporting up to 10x faster interactive generation than CPU/RAM offloading [3].

Two properties of this lineage are central to positioning our contribution. First, Petals frames the *wire* as the design-relevant constraint: its load-balancing and routing machinery optimize for network conditions over the internet. Second, and decisively, all published Petals results are measured on NVIDIA consumer and datacenter GPUs. Petals never characterizes the AMD-Radeon-Pro-via-Vulkan/MoltenVK, RDNA4-via-ROCM, or mixed-ROCM+Metal classes. We adopt its pipeline model but arrive at the opposite emphasis: in our deployment, per-node compute, not the activation wire, is the binding constraint (§5).

The closest open systems analog is the llama.cpp RPC backend [12], which exposes remote GGML devices (CUDA, Metal, CPU) over TCP and auto-splits weights and KV cache proportional to device memory. It is the practical substrate many consumer multi-machine setups actually use, but it is self-described as a proof-of-concept, splits by memory rather than compute, and ships no robustness or throughput characterization on cross-vendor GPU meshes. Cake [9] is architecturally nearer to our engine — a Rust server that shards transformer blocks across iOS/Android/macOS/Linux/Windows devices over CUDA, Metal, Vulkan, and CPU backends with a weights-stay-local streaming design — but it too is explicitly experimental, with no published throughput or failure-mode study. Our paper supplies exactly the empirical characterization these projects omit for the consumer-AMD / mixed-ROCM+Metal class.

0.2.2 2.2 Heterogeneity-aware scheduling for consumer clusters

The planner component of our system, Sthambha, belongs to the family of heterogeneity-aware schedulers that assign work by measured per-device capability. Prima.cpp [4] runs 30–70B models on mixed-CPU/GPU consumer home clusters via pipelined-ring parallelism and the Halda scheduler, which co-optimizes per-device CPU/GPU workload split and device selection under RAM/VRAM limits, reporting a Q4 70B run at roughly 1.48 tok/s on four devices. Parallax [5] serves heterogeneous volunteer GPUs with a two-phase scheduler whose “water-filling” phase assigns each device a contiguous layer block proportional to its compute capacity under a VRAM cap

— the published method most directly analogous to Sthambha’s compute-proportional layer assignment — reporting average 1.58x and up to 3.6x throughput over HexGen. Hetis [6] goes further structurally, using *module-asymmetric* placement: keeping compute-intensive dense/MLP work on high-end GPUs while distributing parameter-free attention to weaker GPUs at fine granularity, for up to 2.25x throughput and 1.49x lower latency.

Hetis is the load-bearing prior result for our central thesis. Its finding that, under heterogeneity, per-node compute dominates and a weak GPU can lag a strong one by a large factor on dense/MLP layers is exactly the regime we observe — but we extend it to a harder setting, where the weak nodes are consumer AMD Radeon Pro and RDNA4 cards whose effective compute is further bottlenecked by immature ROCm/Vulkan kernels. The shared limitation across this line is the measurement substrate: Prima.cpp, Parallax, and Hetis are all evaluated on their own device mixes (predominantly NVIDIA-class, e.g. RTX 5090 volunteer nodes in Parallax), and report working positive throughput. They validate compute-aware placement as the right axis, but none characterizes how the planner’s per-node compute estimate F_i , or the failure modes, behave on the consumer-AMD / mixed-ROCm+Metal hardware where F_i is far harder to estimate and where a per-node compute ceiling — rather than allocation quality — caps throughput.

The table below summarizes how the closest scheduling/pipeline systems relate to ours.

System	Design	Scheduling axis	Reported result	Measured hardware class
Petals [1, 2]	Layer pipeline, weights-stay-local	Load balancing for the wire	~1 step/s (176B)	NVIDIA consumer/datacenter
llama.cpp RPC [12]	Remote GGML over TCP	By device memory	none (proof-of-concept)	CUDA/Metal/CPU
Cake [9]	Block sharding, multi-backend	n/a (experimental)	none published	CUDA/Metal/Vulkan/CPU
Prima.cpp [4]	Pipelined-ring, CPU+GPU	Halda (CPU/GPU split)	~1.48 tok/s (70B Q4, 4 dev)	consumer CPU/GPU mix
Parallax [5]	Layer blocks + pipeline select	Water-filling by compute	1.58x–3.6x vs HexGen	NVIDIA volunteer GPUs
Hetis [6]	Module-asymmetric placement	Dense/MLP vs attention split	up to 2.25x throughput	mixed-architecture NVIDIA
Nakshatra + Sthambha (this work)	Layer pipeline, partial sub-GGUF	Compute-aware layer ranges	~0.21 tok/s (70B Q4, 4-Mac)	AMD Radeon Pro / RDNA4 / mixed ROCm+Metal

The throughput figures in the final row are from our deployment; all other figures are as reported by the cited works on their own testbeds and are not directly comparable across hardware.

0.2.3 2.3 Speculative decoding as a throughput lever

Because a single interactive request leaves most layer-stages of a pipeline idle, speculative decoding is the most promising lever for lifting a compute-bound chain toward usable latency — but it ports unevenly across the runtimes in our mesh. On the favorable side, AMD reports that speculative

decoding works natively on the ROCm/vLLM stack, delivering up to 2.31x serving speedup on MI300X for Llama-3.1-70B/405B [19] and batch-1 token-generation speedups of 1.26x–2.99x across PyTorch and vLLM in eager and compiled modes [20]. The batch-1 numbers are directly relevant because our interactive case is single-request. Crucially, however, both results are on CDNA datacenter silicon (MI300X) — exactly the AMD class the literature covers and we do not. Whether these ROCm spec-decode gains carry to consumer RDNA4 (RX 9070 XT), or into a distributed pipeline, is unmeasured and we treat it as [experiment needed].

On the unfavorable side, an upstream llama.cpp report documents that MTP / draft-MTP speculative decoding on Apple Metal is a net throughput *loss* (11%–28%) at every tested configuration — even at 100% draft acceptance — because Metal draft-evaluation overhead exceeds the speculative gain [21]. This is the honest counterweight to the ROCm wins and bears directly on the Metal half of a mixed cluster (we cite it as a point-in-time platform limitation). Our own measurement of speculative decoding on a 16 GB AMD dGPU under MoltenVK shows the same translation-layer fragility (§5).

Pipelined variants target our exact deployment shape. SpecPipe [16] combines a dynamic speculative token tree with pipelined inference to keep every stage busy, reporting 4.19x–5.53x speedup for single-request pipeline-parallel inference; FlowSpec [17] applies tree-based speculative decoding at the network edge under sparse single-request load, reporting 1.37x–1.73x over baselines. Both are benchmarked on homogeneous or NVIDIA-class pipelines, so whether their draft-tree gains survive on consumer AMD/Metal stages is open. SpecExec [18] sets the baseline our distributed chain must beat: it runs 50B+ models at 4–6 tok/s (4-bit) on a single offloaded consumer GPU. If a 70B runs that fast on one consumer GPU, a heterogeneous chain that is compute-bound below it is a negative result worth reporting — though SpecExec is NVIDIA-offload-focused, not the AMD/Metal class we characterize.

0.2.4 2.4 Decentralized inference and training networks

A parallel line builds open or incentivized networks for distributed model serving. exo [7] auto-discovers everyday consumer devices and partitions a model across them with topology-aware parallelization and an OpenAI-compatible API; it is a widely cited consumer-cluster baseline (Prima.cpp benchmarks against it), but it is Apple-Silicon / Thunderbolt-centric and ships no peer-reviewed throughput characterization. Distributed Llama [8] connects home devices via tensor parallelism over LAN with a low-overhead synchronization protocol, and Wavefy [11] shares our weights-stay-local / activations-travel P2P layer-split design but only runs small 1B–3B models, routes by latency rather than per-node compute, and publishes no throughput numbers. Kalavai [10] operates one layer up, as an orchestration control plane that pools spare AMD and NVIDIA capacity into a cluster, rather than characterizing single-model layer-split performance. None of these documents GPU-backend-heterogeneity throughput or failure modes on our class.

The decentralized-*training* school provides useful contrast rather than direct comparison. INTELLECT-1 [15] collaboratively trained a 10B model on 1T tokens across three continents using communication-reducing techniques (int8 all-reduce, DiLoCo-FSDP2) for a 400x bandwidth reduction at 83–96% compute utilization — but on H100 datacenter nodes, the opposite of our consumer-AMD focus. Protocol Learning [14] and Gensyn [13] contribute weight-custody, incentive, and verification machinery for collaborative training. These efforts demonstrate that bandwidth-reduced *training* is feasible at scale; they make no consumer-GPU-heterogeneity *inference* throughput claims, and their bandwidth-centric framing further motivates our compute-not-wire reframing for the inference pipeline.

0.2.5 2.5 Measurement genre and the gap

Methodologically, we follow the discipline of measurement-and-characterization papers that tie every claim to a concrete configuration. Maliakel et al. [43] characterize LLM inference energy/latency tradeoffs across five models and four benchmarks with each number traceable to its setup; we emulate that “every claim has provenance” template, applied to a heterogeneous consumer pipeline rather than single-GPU energy. We also take our genre cue from the position that negative and failed-to-succeed systems results — arising from hardware quirks, communication problems, and software heterogeneity — are publishable and valuable [42]; its taxonomy of failure causes maps almost exactly onto what we encountered.

Gap statement. Across all four lines, the measured hardware is NVIDIA datacenter/consumer, Apple Silicon, or AMD CDNA (MI300X). No prior work reports a reproducible throughput or failure-mode characterization on old AMD Radeon Pro GPUs via Vulkan/MoltenVK, on a consumer RDNA4 card via newly added ROCm, or on a mixed ROCm+Metal cluster over a mesh. This is the empirical gap we fill, building on the Petals-derived pipeline [1, 2] and the compute-aware scheduling intuition of the Prima.cpp/Parallax/Hetis family [4–6], and reporting honest negative results in the spirit of [42].

0.3 3. System Architecture: Nakshatra and the Sthambha Planner

Nakshatra is a layer-pipeline collaborative-inference engine derived from the Petals design [1, 2], in which a single large model is partitioned across machines by transformer-block range: each worker holds only the weights for a contiguous slice of layers and computes that slice’s forward pass, passing hidden states (activations) to the next worker. The system’s distinctive choices are (i) *weights stay local, only activations travel*; (ii) *partial sub-GGUF loading* so a worker materializes only its assigned tensors; (iii) a *gRPC-plus-custom-fabric transport* over a Tailscale mesh; and (iv) a separate compute-aware planner, **Sthambha**, that assigns layer ranges by measured per-node compute rather than by memory or latency. We describe each in turn, then the security model. Throughout, every quantitative claim is tied to a measurement from our deployment; design elements we have not yet measured are marked [experiment needed].

0.3.1 3.1 Weights stay local; only activations travel

In Nakshatra a token’s forward pass is a pipeline: worker w_0 embeds the input and runs its layer block, emits the hidden state at its boundary, and a subsequent worker resumes from that boundary. Only the hidden state crosses the wire. For a dense model with hidden size h in fp16, the per-token per-hop payload is $h \times 2$ bytes. For a 70B-class model with $h = 8192$ this is **16 KB/token/hop** in our deployment; a five-hop pipeline carries roughly 80 KB/token, so even at 5 tok/s the activation fabric moves only about **400 KB/s** — trivial on a LAN or Tailscale mesh. This is the architectural basis for the paper’s central reframing: the wire is cheap; the binding constraint is per-node compute (g5).

This contrasts with memory-split RPC approaches such as llama.cpp’s RPC backend [12], which shard weights and KV cache by device memory over TCP. Because weights are streamed/assigned by memory rather than held permanently per node, such setups re-transfer large weight volumes; in our deployment the weights-stay-local design avoids the roughly 26 GB-per-worker weight movement a memory-split 70B Q4 configuration would otherwise incur per restart.

0.3.2 3.2 Partial sub-GGUF loading

Each worker loads only the tensors for its assigned layer range from a **pre-staged sub-GGUF** — a GGUF file [35] containing a subset of the original model’s blocks plus Nakshatra layer-range metadata (`nakshatra.layer_range_start` / `layer_range_end`). Stock `llama.cpp` [34] cannot load such a file: in our deployment a partial GGUF triggers a hard error, `missing tensor output_norm.weight`, because the loader expects the full tensor set (a 70B model is roughly 40 GB across 80 blocks / 724 tensors; a `--keep 20` slice is about 10 GB / 182 tensors). This empirically motivates a loader patch.

The patch surface is small. Across the partial-load and partial-decode paths we measured **5 files** / **~70 LOC net**, applied cleanly across two upstream `llama.cpp` commits; the worker daemon, worker, and client glue add roughly 155 / 190 / 140 LOC respectively. Mixture-of-Experts partial loading required additional model-specific patches: in our deployment Qwen3-MoE partial loading needed two further patches (on the order of 30–60 LOC each) before a sub-GGUF would load, a gap we have since closed.

0.3.3 3.3 Activation fabric and transport

Workers communicate over two paths in our deployment:

- a **gRPC** control/forward path used for the client-relayed pipeline, carrying `ForwardRequest`-style hidden-state payloads; and
- a **custom UDP “fabric”** transport intended to remove a client-relay hop by pushing activations worker-to-worker.

Both run over a Tailscale/WireGuard mesh on commodity LAN. The fabric exposed two transport lessons that we report as measured facts:

1. **Burst loss.** A first cut of the UDP activation transport suffered roughly **75% multi-chunk packet loss** over Tailscale on large activations (a 16 KB activation fragments into ~12 datagrams at a 1500-byte MTU). Adding a receive-buffer-size knob (`SO_RCVBUF`) and a send-pacing knob drove loss from ~75% to **0** on cluster smoke tests (before: 0/5 round trips succeeded; after: 5/5), with post-fix RTT p50 66 ms / p99 69 ms versus a single-datagram baseline of p50 50 ms / p99 140 ms. The loss appeared only with multi-chunk activations; single 1024-byte datagrams were lossless (10/10).
2. **The round trip is GPU-DMA-bound, not matmul-bound.** A fabric micro-benchmark on the RDNA4 card under ROCm showed `llama_get_embeddings` — the GPUhost DMA read of the boundary hidden state — accounting for **86%** of the GPU round trip (3.21 ms of 3.7 ms), while the decode step was only 0.33–0.37 ms (~9%). In CPU mode the picture inverts: decode is ~97% of the round trip (57–62 ms) and transport is noise. This locates the per-hop overhead in moving activations off the GPU, not in the layer matmul or the network, and reinforces the compute-not-wire framing (§5).

0.3.4 3.4 Sthambha: the compute-aware planner

Sthambha is the dedicated planner component, run as a separate daemon (on a low-power single-board coordinator node in our testbed). It assigns each worker a **contiguous layer range proportional to that node’s measured per-node compute**, subject to the node’s VRAM cap — the same water-filling / heterogeneity-aware partitioning family as Parallax’s compute-proportional

layer-block planner [5] and Prima.cpp’s Halda scheduler [4]. The key design commitment, and the one our measurements justify, is that the placement axis is *measured compute* F_i , **not** memory footprint or link latency: latency-routed P2P designs [11] and memory-split RPC [12] optimize the wrong variable for this hardware class, since §5 shows compute dominates.

Two consequences for the planner on consumer AMD / Apple hardware are worth stating up front. First, estimating F_i is harder here than on the NVIDIA/CDNA nodes prior planners assume: effective per-node compute depends on backend maturity (immature ROCm/Vulkan kernels) and on workload shape — for small-active-set MoE we measured CPU workers running 2–4× faster than Metal (§5), so a naive VRAM- or FLOP-nameplate estimate of F_i would mis-rank nodes. Sthambha must therefore derive F_i from *measured* per-node step latency rather than from device specifications. Second, compute-aware allocation sets placement but, per our central finding, cannot raise the per-node compute ceiling; module-asymmetric placement in the spirit of Hetis [6] is the structural lever we identify but do not yet implement here [experiment needed] (§6, §8).

0.3.5 3.5 Security model

Pillar calls between nodes are authenticated with **Ed25519 signatures** (scheme identifier **Sthambha-Ed25519**; per-worker signing key stored at restricted permissions, established trust-on-first-use). Channels are pinned: the mesh uses **SPKI-style public-key pinning with TLS**, with worker-to-worker push carried over a pinned TLS channel and pin mismatches recorded to an append-only audit log. Combined with Tailscale-only reachability, this gives a signed, identity-pinned authenticated mesh; a documented failure during the security sprint (gRPC hostname verification on IP connections) was fixed via an explicit SSL target-name override.

We distinguish what is shipped from what is aspirational. The shipped reality is **signed authenticated requests over pinned, Tailscale-only channels**. The stronger notion of a per-session *signed work receipt* as a standalone proof-of-work primitive — usable to attest that a worker actually performed its assigned layer computation — is [experiment needed]: we have the signing and pinning substrate but no measured receipt-as-proof-of-work artifact. Verifying such receipts across a cross-vendor chain is further complicated by floating-point non-associativity and architecture-dependent atomics [40, 41], which make bit-exact activation equivalence unattainable on this hardware (§5, §7).

0.4 4. Experimental Setup

This section documents the heterogeneous testbed, the runtime stacks under test, the models exercised, and the measurement methodology. Following the discipline of measurement-driven characterization work [43], every reported number in the Results section is tied to a specific configuration and source artifact; here we fix those configurations. We make no claim of generality beyond this testbed: it is a single, deliberately heterogeneous cluster of commodity hardware, and several findings are point-in-time against specific software builds (§7).

0.4.1 4.1 Testbed

The cluster is a seven-node mesh of commodity desktop workstations plus a single-board coordinator, connected over a Tailscale/WireGuard overlay on a commodity LAN. The nodes were chosen to span exactly the consumer hardware class the distributed-inference literature does not measure: older AMD Radeon Pro GPUs reached through Apple’s Metal stack and through Vulkan-over-Metal translation [31], a consumer RDNA4 card on recently added ROCm support [25], and the resulting cross-vendor ROCm+Metal combination.

Class	Count	GPU	VRAM	Runtime path
Commodity desktop workstation	4	AMD Radeon Pro 5700 XT	16 GB	Metal (native); Vulkan via MoltenVK [31]
Commodity desktop workstation	1	AMD Radeon Vega 56	8 GB	Metal (native); Vulkan via MoltenVK [31]
Consumer Linux desktop	1	AMD Radeon RX 9070 XT (RDNA4, gfx1201)	16 GB	Native ROCm [24]
Single-board coordinator	1	— (CPU only)	—	Planner daemon

The single-board coordinator hosts the Sthambha planner daemon and does not participate in layer execution. All inter-node traffic — control-plane calls and activation transport — traverses the Tailscale overlay; no node is exposed on a public interface.

Two properties of this testbed are worth stating explicitly because they shape every result that follows. First, the GPU vendors and runtimes are genuinely mixed within a single inference chain: a Radeon Pro worker reached through Metal can sit in the same pipeline as the RDNA4 worker reached through ROCm. Second, the AMD GPUs are reached through software paths that are not their native vendor stack — Metal and MoltenVK on macOS, and only-recently-supported ROCm on RDNA4 — which is precisely the regime prior work routes around.

0.4.2 4.2 Runtime maturity context

The maturity of the software stack on this hardware class is itself a measured property of the environment, not an incidental detail, so we record it here.

On the ROCm side, RDNA4 (the RX 9070 series, gfx1201) became a formally supported ROCm target only with the 6.4.1 release in May 2025 [25], and the broader compatibility matrix does not list many older consumer Radeon Pro parts at all [24]. Quantization tooling on consumer AMD lags further: bitsandbytes flags AMD support as Preview grade [26], and we independently hit a documented runtime failure of its LoRA/8-bit path on gfx1201 attributed to hipBLASLt on an unsupported architecture [27]. We therefore treat the ROCm consumer path as functional but bleeding-edge.

On the Metal side, macOS exposes no native Vulkan: any Vulkan workload — including llama.cpp’s Vulkan backend [34] — necessarily routes through MoltenVK, which layers a Vulkan subset over Apple’s Metal [31, 33]. MoltenVK’s own documentation states that this layer carries unavoidable memory and performance overhead and transpiles SPIR-V shaders to Metal Shading Language [32], in contrast to Apple’s hand-tuned native compute primitives [28, 30]. The native Metal baseline itself was designed around Apple-Silicon unified memory [29]; the discrete, non-unified AMD Radeon Pro GPUs in our testbed sit outside that design point, a mismatch that surfaces directly in the failure modes of §5.

0.4.3 4.3 Inference engine and models

All workers run on a patched build of llama.cpp [34], whose multi-backend design (CPU, Metal, Vulkan, ROCm/HIP) is what permits a single engine to span the mixed ROCm+Metal mesh.

Weights are loaded locally from GGUF files [35]; each worker loads only its assigned contiguous layer range from a pre-staged sub-GGUF rather than the full model. Because stock llama.cpp hard-fails when handed such a partial model (it aborts with a missing-tensor error for the final-block weights, in our deployment), partial-GGUF loading required patching the loader; the patch surface was small (5 files, ~70 LOC net in our deployment) and applied cleanly across two upstream commits. Architecture details are deferred to §3.

We exercise both a dense and a Mixture-of-Experts model to separate vendor/runtime effects from model-structure effects:

Model	Type	Quantization	Role
Llama-3.3-70B	Dense (hidden=8192)	Q4_K_M	Primary distributed-chain throughput target
Qwen3-Coder-30B-A3B	MoE (8/128 experts active)	Q4_K_XL	Cross-machine MoE coherence and CPU-vs-Metal study
Llama-3.2-3B	Dense (hidden=3072)	Q8_0 / q8	Toy chain and fabric micro-benchmark target
Qwen3-1.7B	Dense	Q4_K_M	Metal-nondeterminism probe; speculative-decoding draft model

The 70B is the headline distributed target; its hidden size of 8192 fixes the per-token activation payload at 16 KB/token/hop (fp16) used throughout the compute-not-wire analysis. The Qwen3-Coder-30B-A3B MoE, with 8 of 128 experts active per token (~3B active parameters), is the structural counterpoint used to surface the MoE cross-machine coherence collapse and the CPU-beats-Metal small-active-set result. The smaller dense models serve the toy two-worker chain, the fabric micro-benchmarks, and the speculative-decoding probe.

Running the Qwen3-Coder MoE under partial loading required two additional small loader/decode patches beyond the dense path (in our deployment, ~30–60 LOC each); these were in place for the MoE runs reported in §5.

0.4.4 4.4 Measurement methodology

Our methodology is measurement-first and provenance-anchored, in the spirit of [43]: each reported number is bound to the model, quantization, node placement, and software build that produced it, and is drawn from a logged run rather than reconstructed. We adopt the negative-results framing of [42]: characterized failure modes — crashes, coherence collapse, regressions — are reported as findings with their reproduction conditions, not omitted as noise.

Throughput is reported as steady-state tokens per second over the streaming generation path, measured on multi-token runs so that one-time prefill and model-load costs are excluded from the per-token rate. For the worker-to-worker push versus client-relay comparison, both transports are run on the identical node placement and prompt so that the only varied factor is whether each token’s activation makes an extra client hop; we report the two rates side by side rather than a single delta.

Per-step (per-layer-block) latencies for the CPU-versus-Metal MoE comparison are measured at the individual worker, attributing time to the specific node and backend executing that layer range. Fabric transport behavior (packet loss, round-trip latency) is measured with a dedicated cluster smoke harness over the live Tailscale overlay, and the GPU round-trip micro-benchmark instruments the engine’s internal stages (forward decode versus the `llama_get_embeddings` host read) separately so that compute time and DMA-readback time can be attributed independently.

For correctness and determinism, we compare token streams produced under identical arguments. We treat nondeterminism as a first-class measured outcome: where two runs with identical arguments at temperature 0 diverge, we report the divergence rather than averaging it away. We note up front that this makes bit-exact equivalence between heterogeneous backends unverifiable in principle on the Metal path — a consequence of floating-point non-associativity and architecture-dependent atomic ordering [40, 41] — and we therefore do not assert bit-exact cross-vendor equivalence anywhere in this work.

0.4.5 4.5 Scope and reproducibility caveats

The testbed is a single fixed cluster with low replication; reported rates are single-cluster measurements, and we do not extrapolate beyond them. Several runtime versions are pinned to specific llama.cpp builds (e.g. the Metal-nondeterminism and speculative-decoding probes were taken on a specific build/commit, recorded with each result in §5), and at least one platform-specific regression we characterize has since changed state upstream; we flag these as point-in-time rather than permanent in §7. Where a comparison we would like to make was not run on this testbed — most notably native-ROCm speculative decoding on the RDNA4 card, and whether MI300X ROCm spec-decode gains [19, 20] carry to consumer RDNA4 — we mark it [experiment needed] rather than estimating a number.

0.5 5. Results and Findings

This section reports what we measured running the Nakshatra engine across the heterogeneous testbed of §4. We organize the findings around the paper’s central reframing — that per-node compute, not the activation wire, governs throughput — and then catalogue the characterized failure modes, each tied to the deployment configuration and source artifact that produced it. Following the measurement-paper discipline of [43], every number below is reported with the configuration that generated it, and we mark with [experiment needed] any quantity we did not measure rather than estimating it.

0.5.1 5.1 End-to-end throughput

Our headline operating point is a four-machine dense chain. Running Llama-3.3-70B (Q4_K_M) split across four commodity workstations, the steady-state throughput was approximately **0.21 tok/s** for both the client-relay streaming path and the worker-to-worker forward path (in our deployment, 2026-05-13). A reduced two-worker “toy” chain — a single-layer-range handoff between two nodes serving an 8-bit 3B model — ran at **1.37–1.51 tok/s**.

For calibration against the well-characterized single-machine baseline, the earlier CPU-only v0.1 cross-machine acceptance run (two machines over the Tailscale mesh, an 8-bit 3B model with hidden size 3072) completed a single token in **510 ms (1.96 tok/s single-token)**, decomposed as roughly **220 ms of compute per worker** plus roughly **290 ms of Tailscale round-trip** to move the 72 KB of activations for a six-token prefill. The generated token (id 12366, “Paris”) byte-matched the single-machine reference, confirming the pipeline is numerically faithful on CPU. A localhost

two-worker configuration (no real network) completed a single token in **220 ms (4.52 tok/s)**, moving 73 KB of hidden state across the in-process wire.

Configuration	Model	Path	Throughput
4 machines	Llama-3.3-70B	streaming (client	~0.21 tok/s
	Q4_K_M	relay)	
4 machines	Llama-3.3-70B	forward (worker push)	~0.19–0.21 tok/s
	Q4_K_M		
2 workers (toy)	3B Q8	streaming	1.51 tok/s
2 workers (toy)	3B Q8	push	1.37 tok/s
2 machines, CPU-only (v0.1)	3B Q8, hidden=3072	single-token	1.96 tok/s (510 ms)
2 workers, localhost	—	single-token	4.52 tok/s (220 ms)

All numbers are from our deployment. These are single-cluster, low-replication measurements (see §7); they are not offered as competitive throughput, but as the empirical baseline on a hardware class no prior work measures. For context, the closest positive prior result on a heterogeneous home cluster, Prima.cpp, reports a 4-device Q4 70B at ~674 ms/token (~1.48 tok/s) on its own NVIDIA-and-CPU device mix [4]; our 0.21 tok/s is on the consumer-AMD-Radeon / Metal class it does not cover.

0.5.2 5.2 Compute, not the wire: the central reframing

The activation traffic on this pipeline is negligible. For the 70B (hidden size 8192), each hop carries `hidden_size` \otimes 2 bytes per token in fp16 — **16 KB/token/hop**. Across a five-hop pipeline that is ~80 KB/token; even at an aspirational 5 tok/s this is only **~400 KB/s**, trivial on a LAN or Tailscale mesh (activation math from our deployment; consistent with the architecture in [2]). The v0.1 CPU run moved 72 KB for a six-token prefill and still spent ~43% of its single-token wall-clock on compute even on a slow CPU path.

The decisive test was an optimization that should have helped if the wire were the constraint: a **worker-to-worker push** that eliminates one client-relay network hop per token. It measured **slightly slower**, not faster, on both scales:

Optimization	Setting	Relay (baseline)	Push	Result
Hop elimination	4-machine 70B, multi-hop	0.21 tok/s	0.19 tok/s	push slower
Hop elimination	2-worker toy	1.51 tok/s	1.37 tok/s	push slower

Removing a network hop did not improve throughput because per-token compute dominates per-token network; the pre-registered 25% latency-improvement criterion was unmet, and we re-cut the success criterion to “speedup conditional on the network’s share of total cost” (in our deployment). This is the paper’s core empirical message: on this hardware the binding constraint is the per-node compute ceiling, not bandwidth. It aligns with — and extends to consumer AMD/Metal silicon — the heterogeneity finding of Hetis, that under heterogeneity per-node compute on dense/MLP work dominates and a weak GPU can lag a strong one by a large factor [6].

A fabric micro-benchmark localizes *where* the per-node time goes. On the RX 9070 XT under ROCm, serving a Llama-3.2-3B Q8 layer-0 slice with one-token decode, `llama_get_embeddings` —

the GPUhost DMA read of the activation — accounted for **3.21 ms of a 3.7 ms (86%)** GPU round trip, while the decode matmul was only **0.33–0.37 ms (9%)** and the host-side memcpy/shm handoff was ~6 ts. In CPU mode the picture inverts: decode is 57–62 ms (**97%**) and transport is noise. The implication is that the residual per-node cost on the GPU path is the embeddings read-back, not the matmul — confirming again that the optimization surface is on-node, not on the wire.

0.5.3 5.3 Failure-mode catalogue

5.3.1 MoE cross-machine coherence collapse Splitting a Mixture-of-Experts model across machines failed in a categorical, not merely statistical, way. A Qwen3-Coder-30B-A3B (Q4_K_XL) chain across four Macs ran at **4.17 tok/s with zero transport errors**, yet the output **collapsed into repetitive garbage** (e.g. "redirection redirection", "BryCppClass Vietnamese Vietnamese"), with 8 of 128 experts active per token and 48 layer-boundary handoffs (in our deployment). Our hypothesis is that floating-point drift across heterogeneous nodes flips the top-k expert-gating decision: because gating selects a discrete expert set, a sub-threshold numerical perturbation produces a *different expert*, making the error categorical rather than a continuous drift the residual stream can absorb. Dense models survive the same cross-machine drift; MoE does not. This is consistent with floating-point non-associativity and atomic-ordering variation as documented sources of GPU run-to-run nondeterminism [40], whose hardware basis on the AMD side is the architecture- and memory-type-dependent FP atomics support in ROCm [41]. The precise gating-flip mechanism — instrumenting which token positions diverge at which layer boundary — is [experiment needed].

5.3.2 MoE on Metal: CPU beats Metal 2–4× On the same A3B MoE, CPU workers were **2–4× faster than Metal** per decode step, the opposite of the expected GPU advantage:

Stage	Backend	Layers	Per-step latency
Stage 1	Metal (Radeon Pro 5700 XT)	[0, 12)	117 ms
Stage 2	CPU	[12, 24)	53 ms
Stage 3	Metal (Radeon Pro 5700 XT)	[24, 36)	116 ms
Stage 4	CPU	[36, 48)	31 ms

(in our deployment). With only ~3B of 30B parameters active per token, the Metal kernel-dispatch overhead does not amortize over the small active footprint, so the CPU’s lower fixed dispatch cost wins. This is a concrete instance of the general lesson that on heterogeneous nodes effective per-node compute (**F_i**) must be *measured*, not inferred from the backend label or VRAM — the planning axis that compute-aware schedulers such as Parallax’s water-filling [5] and Prima.cpp’s Halda [4] operate on.

5.3.3 Metal nondeterminism at temperature 0 Two llama.cpp runs with **identical arguments and temperature 0** produced **different tokens** on the Radeon Pro 5700 XT under Metal — diverging from the *first* token, with the model fully on Metal and no CPU spillover (Qwen3-1.7B-Q4_K_M, **-ngl 99 --temp 0**, build 8142 / commit 8c2c0108d, in our deployment). The divergence is in the logits, upstream of sampling, so it is not a sampling artifact. This makes bit-exact equivalence proofs for a cross-vendor activation chain unsatisfiable on the Metal leg, and is grounded in floating-point non-associativity [40] and architecture-dependent atomics [41]. It echoes

upstream reports of the Metal backend mishandling discrete (non-unified-memory) AMD Radeon Pro GPUs [38].

5.3.4 Metal last-worker buffer assert Constructing an all-GPU 70B chain was blocked by a hard crash on the *final* worker. A `GGML_ASSERT(buf_src)` fires in `libggml-metal` (`ggml-metal-device.m:1624`) because `inp_out_ids` — a small `int32` tensor (24 bytes for a six-token prefill), below page size and unaligned — violates the page-alignment requirement of Apple’s `newBufferWithBytesNoCopy`. The assert fires every time, and only in last-mode (`has_lm_head=true`, which builds `inp_out_ids`); forcing `cp.embeddings != mode_last` did not fix it. The workaround is to place the final worker on CPU, which precludes a fully-GPU 70B chain (discovered 2026-05-13 during a five-machine 70B run, in our deployment). This sits at the same old-AMD-Radeon-Pro-under-Apple’s-GPU-stack intersection as the upstream Metal/AMD reports [38].

5.3.5 Speculative decoding: 30× slower on the MoltenVK translation layer Speculative decoding — our most promising throughput lever (ğ6) — regressed catastrophically on the Vulkan-over-Metal path. With a Qwen3-Coder-30B-A3B-Q4_K_XL target and a Qwen3-1.7B-Q4_K_M draft (`--draft-max 5`) on a 16 GB AMD discrete GPU (the Radeon Pro 5700 XT) on macOS via MoltenVK (`llama.cpp b8142`), draft decoding ran at **1.2–1.3 tok/s versus a 36–37 tok/s target-only baseline** — a **~30× regression** — **despite 59–71% draft acceptance** (in our deployment).

Critically, this is a **translation-layer failure, not an AMD-silicon failure**: the cause is VRAM contention and Vulkan context-switching between the two models routed through MoltenVK’s SPIR-VMSL pipeline, whereas CUDA handles dual-model speculative decoding cleanly. (Note: an earlier finding label loosely attributed this to a “Vega 56”; the source body records the hardware as a Radeon Pro 5700 XT — Navi/RDNA1, not Vega — so we attribute the result to the 5700 XT / 16 GB AMD dGPU on MoltenVK.) MoltenVK’s own documentation acknowledges the intrinsic overhead and shader transpilation of the VulkanMetal layer [32], and there is no native Vulkan on macOS — any Vulkan workload necessarily routes through it [33]. This is the honest negative counterweight to AMD’s native-ROCm speculative-decoding wins on MI300X [19, 20] and to the parallel Metal-side regression upstream [21]: the lever works natively on ROCm/CDNA but not through the translation layers our testbed is forced onto.

5.3.6 Fabric transport: 75% burst loss, fixed to zero The custom UDP activation transport initially suffered **~75% multi-chunk packet loss** over Tailscale on large activations: a 16 KB activation fragments into 12 UDP datagrams at MTU 1500, and before the fix only **0 of 5** round trips completed. Tuning two knobs — a larger receive buffer (`SO_RCVBUF, NAKSHATRA_FABRIC_RECV_BUF_BYTES=4194304`) and a send-pacing delay (`SEND_PACE_S=0.001`) — brought loss to **0 (5 of 5** round trips), with post-fix RTT `p50 = 66 ms / p99 = 69 ms` (resolved 2026-05-29; in our deployment). A single-datagram (1024-byte) baseline never exhibited the loss (10/10 zero-loss, `p50 = 50 ms / p99 = 140 ms`), confirming the loss is specific to multi-chunk bursts, not the link. This is an engineering fix at the fabric layer, consistent with the self-described proof-of-concept fragility of plain-transport remote-GGML approaches such as `llama.cpp`’s RPC backend [12].

0.5.4 5.4 Translation-layer correctness, in context

The MoltenVK and Metal failures above are not isolated to our cluster. Upstream `llama.cpp` reports independently document garbled Vulkan output on a vintage Radeon Pro 580X via MoltenVK after a Flash-Attention refactor [36], corrupted output on macOS x86 + AMD Radeon through

the Vulkan/MoltenVK backend at a specific build [37], Metal-backend garbage on a discrete AMD Radeon Pro 5300M while CPU and Vulkan are fine [38], and Vulkan garbling on integrated Ryzen-APU Radeon parts [39]. Together with our own measurements, these establish that translation-layer and discrete-AMD-on-Apple correctness quirks span the whole consumer-AMD class — they are a property of the regime, not a one-off on a single card.

0.5.5 5.5 Architecture confirmations

Two architectural choices were validated empirically rather than asserted. First, **partial sub-GGUF loading is necessary**: stock llama.cpp hard-fails on a partial model with `missing tensor output_norm.weight` (the 70B is 40 GB / 80 blocks / 724 tensors; a `--keep 20` slice is 10 GB / 182 tensors), empirically justifying the loader patch (in our deployment). The patch surface is small — **5 files, ~70 LOC net**, applied cleanly across two upstream commits [34, 35]. Because each worker loads only its assigned layers and weights never move at run time, the design avoids the per-worker weight-streaming cost of memory-split RPC approaches [12]. Second, byte-equal wire exchange (73728 bytes, matching fnv1a hash) confirmed the orchestration is correct independent of any numerical drift, and a separate streaming KV-cache reuse change gave a **~5 \times** multi-token speedup over a brute-force resend-context baseline (in our deployment).

On security, the shipped reality is Ed25519-signed pillar requests over an SPKI/TLS-pinned, Tailscale-only authenticated mesh; the stronger notion of a signed per-session **work receipt** as a proof-of-work primitive is **[experiment needed]** — the current implementation authenticates requests and pins channels but does not yet produce a verifiable proof of computation performed.

0.6 6. Discussion and Lessons

Our measurements converge on a single, reorienting claim: across this consumer AMD Radeon Pro / RDNA4 / mixed ROCm+Metal class, throughput is governed by per-node compute, not by the activation wire. This section interprets that finding into reviewer-defensible lessons about what realistically moves throughput on such hardware, and — equally important — what does not. We are explicit throughout about which lessons are measurement-backed in our deployment, which are inherited from prior work on different hardware, and which remain [experiment needed].

0.6.1 6.1 Why the wire is the wrong place to optimize

The single most counterintuitive result in our deployment is that a worker-to-worker push, which eliminates a client-relay hop per token, measured *slightly slower* than client relay: 0.19 vs 0.21 tok/s on the multi-hop 4-Mac 70B chain, and 1.37 vs 1.51 tok/s on the toy 2-worker chain. The pre-registered 25% latency-improvement criterion was unmet. The mechanism is straightforward once the magnitudes are laid out: per-token activation traffic for a 70B (hidden=8192, fp16) is 16 KB/token/hop, so a 5-hop pipeline at ~5 tok/s moves only ~400 KB/s — trivial on LAN or a Tailscale mesh. Even the earliest CPU-only cross-machine acceptance run, where a 72 KB transfer cost ~290 ms of Tailscale RTT against ~220 ms of compute per worker, already showed network and compute in the same order of magnitude; once the chain runs on GPUs at 70B scale, compute dominates outright.

This is the empirical content of the title. It implies that two classes of optimization the distributed-inference literature emphasizes are low-value on this hardware:

- **Wire/transport optimization beyond correctness.** Our fabric micro-benchmark localizes the real cost inside the GPU round trip, not on the link: `llama_get_embeddings` (the

GPUhost DMA read) was 3.21 ms of a 3.7 ms total round trip (86%) on the RX 9070 XT under ROCm, while the decode matmul itself was only 0.33–0.37 ms (9%). Even a perfect wire cannot touch the dominant term. Transport work is worth exactly enough effort to be *correct* — which, as §5 showed, was not free: a custom UDP activation transport hit ~75% multi-chunk packet loss over Tailscale before receive-buffer-size and send-pacing knobs drove it to 0. Beyond that correctness floor, further link tuning is not where throughput lives.

- **Micro-batching for link amortization.** Batching to amortize per-message network overhead presupposes the network is the bottleneck; here it is not. With single-request interactive load and a compute-bound pipeline, micro-batching does not address the binding constraint. (Batching to raise *aggregate* multi-request throughput is a separate goal we do not study; our regime is the single interactive stream.)

This reframing is consistent with — and extends — the heterogeneity-scheduling literature, which already treats per-device compute capacity, not bandwidth, as the placement axis [4–6]. We note the contrast with the decentralized-*training* school, where communication bandwidth genuinely is the lever optimized (e.g. int8 all-reduce and DiLoCo-style reductions for a 400x bandwidth cut) [14, 15]: that work is bandwidth-bound by construction; our single-request inference pipeline is not.

0.6.2 6.2 What compute-aware partitioning can and cannot do

If compute dominates, the first-order question is how to place layers so that the slowest stage is as fast as possible. This is precisely the water-filling / Halda family that Sthambha belongs to: assign contiguous layer ranges proportional to measured per-node compute capacity under each node’s VRAM cap [4, 5]. Our deployment validates the *axis* — compute, not memory or latency, is what placement should track — but it also exposes the ceiling that compute-aware partitioning cannot lift.

A pipeline’s steady-state throughput is bounded by its slowest stage. Compute-aware partitioning can balance stages so no node is gratuitously overloaded, but it cannot make a node compute faster than its hardware-and-runtime ceiling. On this class that ceiling is low and, worse, misestimated by the obvious proxies:

- VRAM is a poor proxy for compute here. Two of our nodes illustrate why estimating the capacity term F_i from memory is wrong on this hardware: on small-active-set MoE, CPU workers were *2–4x faster than Metal* (per-step latency Metal 117/116 ms vs CPU 53/31 ms), because Metal kernel-dispatch overhead does not amortize when only ~3B of 30B parameters activate. A scheduler that ranked nodes by VRAM, or that assumed “GPU > CPU,” would place layers exactly wrong. The lesson is operational: **estimate F_i from measured per-node, per-workload compute, not from VRAM or device class.**
- The ceiling itself is the binding term. Even with balanced placement, the 4-machine 70B chain sat at ~0.21 tok/s. Partitioning sets *which* node is the bottleneck stage; it does not change the fact that some stage’s per-token compute is the floor.

So compute-aware partitioning is necessary table-stakes — the right objective and the right input signal — but on heterogeneous consumer nodes it is a placement tool, not a throughput multiplier.

0.6.3 6.3 Module-asymmetric placement: the more promising structural lever

Because contiguous-range partitioning is capped by the slowest *node*, the more promising structural change is to stop treating a node as an atomic stage. Heti’s module-asymmetric placement — keeping compute-intensive dense/MLP work on stronger GPUs while distributing parameter-light attention to weaker GPUs at fine granularity — directly attacks the per-node ceiling rather than merely balancing around it [6]. On a “one-fast-many-slow” topology, which is exactly the shape of our testbed (a single RDNA4 ROCm node among several older Radeon Pro Metal/Vulkan nodes), this is the natural fit: route the heavy dense matmuls to the fast node and the cheaper, more numerous attention work to the slow ones.

We did not implement module-asymmetric placement; on our class it is a designed-but-unmeasured lever, and quantifying its effect under Sthambha is [experiment needed]. We flag two class-specific risks a port must confront, both grounded in our own findings: (i) finer-grained cross-node dispatch increases the number of activation handoffs, and while §6.1 shows handoffs are cheap on the wire, each handoff is a place where cross-vendor floating-point drift accumulates — and §5 showed that drift is benign for dense models but *categorically* destructive for MoE expert gating; and (ii) the same dispatch-overhead pathology that made Metal lose to CPU on small active sets could blunt fine-grained gains on the Metal nodes specifically. Module-asymmetric placement is therefore our most promising structural direction and simultaneously the one most entangled with this hardware’s correctness hazards.

0.6.4 6.4 Speculative decoding: the strongest single-stream lever, and an overturned assumption

For a single interactive request, most pipeline stages sit idle most of the time — the worst case for pipeline parallelism. Speculative decoding is the technique aimed squarely at this low-utilization regime, and pipelined variants (SpecPipe, FlowSpec) target exactly the sparsely-utilized distributed pipeline we run [16, 17]. On-device single-GPU speculative offload (SpecExec) reports 50B+ at 4–6 tok/s (4-bit) on one consumer GPU [18] — which doubles as the baseline a distributed chain must beat to justify its existence, and against which our ~0.21 tok/s 70B chain is, honestly, a negative result on raw throughput.

The critical nuance our data forces is *where* speculative decoding fails. In our deployment, speculative decoding was ~30x slower on a 16 GB AMD dGPU via MoltenVK (1.2–1.3 vs 36–37 tok/s) **despite 59–71% draft acceptance** — i.e., the drafts were good; the *execution path* was the problem. The cause was VRAM contention / context-switching between target and draft models through the VulkanMetal translation layer, not the AMD silicon. (We correct an earlier internal attribution: this ran on the Radeon Pro 5700 XT — Navi/RDNA1 — via MoltenVK, not on the Vega 56.) The honest counterweight on the Apple-native side is that Metal MTP speculative decoding was reported as a net throughput loss at every configuration even at 100% acceptance — we cite this as a point-in-time platform limitation [21].

This is the assumption our findings overturn: the failure is a *translation-layer* failure, not an *AMD* failure. On native ROCm, AMD’s own measurements show speculative decoding delivering real batch-1 gains — 1.26x–2.99x on MI300X, and up to 2.31x serving speedup on 70B-class models [19, 20]. Batch-1 is exactly Nakshatra’s interactive case. The opportunity is therefore concrete and class-specific: **native-ROCm speculative decoding on the consumer RDNA4 node (RX 9070 XT) is the most promising single-stream throughput lever on this testbed**, precisely because it sidesteps both the MoltenVK translation layer that killed it on the Radeon Pro and the Metal regression on the Apple side. The unavoidable caveat is the hardware transfer: every

supporting ROCm number is on CDNA datacenter silicon (MI300X), not consumer RDNA4, and ROCm consumer support for RDNA4 (gfx1201) is recent and uneven — formal RDNA4 support landed only in ROCm 6.4.1 (May 2025) [24, 25], and adjacent quantization tooling is “Preview” grade with documented gfx1201 failures [26, 27]. **Whether the MI300X spec-decode gains carry to consumer RDNA4, and survive inside a distributed pipeline, is [experiment needed]** and is the single highest-value experiment this work points to.

0.6.5 6.5 KV-cache quantization: real, but orthogonal to the throughput ceiling

KV-cache quantization (KVQuant, KIVI) shrinks per-node KV footprint substantially and can enlarge context or batch capacity on memory-starved consumer GPUs [22, 23]. In a layer-pipeline each node holds KV only for its own layers, so this directly relaxes the per-node memory pressure on the smaller cards (e.g. the 8 GB Vega 56). But our bottleneck analysis (§6.1) places the binding constraint on compute, not on KV memory or the ~16 KB/token activation wire. KV-cache quantization therefore buys *capacity* — longer context, larger batch — not single-stream throughput. We position it as a useful, orthogonal lever for long-context runs on memory-limited nodes, not as a path to lifting the compute ceiling.

0.6.6 6.6 Dense over MoE, and runtime fragility as a first-class concern

Two cross-cutting lessons fall out of the failure catalogue.

Prefer dense models for cross-machine robustness. Dense models tolerated cross-machine floating-point drift; MoE did not. The Qwen3-Coder-30B-A3B chain ran fast (4.17 tok/s) with zero transport errors yet emitted repetitive garbage — our hypothesis is that FP drift flips top-k expert-gating decisions, making the error *categorical* rather than a continuous quality sag. The lesson is structural, not tuning: on a cross-vendor activation chain, MoE gating is a discrete decision that small numerical differences can flip, so either keep MoE stages within a single node or default to dense models for multi-machine pipelines. This is grounded in the same floating-point non-associativity and architecture-dependent-atomics theory that explains our broader nondeterminism observations [40, 41].

Treat translation-layer runtimes as fragile. Several of our hardest failures are not about AMD compute capability at all but about immature or translated runtimes: the MoltenVK spec-decode regression (§6.4), the Metal `GGML_ASSERT(buf_src)` page-alignment crash that forced the last worker onto CPU, and Metal nondeterminism at temperature 0 on the Radeon Pro 5700 XT. Upstream bug reports on the same intersections — garbled Vulkan output on vintage Radeon Pro via MoltenVK, Metal garbage on discrete AMD Radeon Pro, and AMD-on-Mac Vulkan regressions — confirm these are class-wide, not one-off [36–39]. The mechanism is documented first-party: macOS has no native Vulkan, so the Vulkan path necessarily transpiles SPIR-VMSL through MoltenVK with admitted overhead [31–33], against a Metal runtime tuned for Apple-Silicon unified memory rather than discrete AMD cards [28, 30]. The lesson for anyone building on this class: budget for runtime fragility as a first-class engineering cost, prefer the native backend (ROCm on RDNA4, Metal/MPS on Apple Silicon) over translated ones (Vulkan-via-MoltenVK), and validate correctness — not just liveness — before trusting any cross-vendor stage.

0.6.7 6.7 Summary of lessons

Lever	Effect on single-stream throughput on this class	Status
Wire / transport optimization (beyond correctness)	Low value — wire is ~400 KB/s vs compute-dominated round trip (embeddings read 86%)	Measured negative (push 0.19 vs relay 0.21 tok/s)
Micro-batching for link amortization	Low value — link is not the bottleneck under single-request load	Argued from §6.1 measurements
Compute-aware partitioning (Halda / water-filling, in Sthambha) [4, 5]	Necessary, but a placement tool, not a multiplier; balances stages, cannot raise the per-node ceiling	Right axis; ceiling observed (~0.21 tok/s 70B)
Estimate F_i from measured compute, not VRAM/device-class	Avoids mis-placement (CPU beat Metal 2–4x on A3B MoE)	Measured
Module-asymmetric placement (Hetis) [6]	Most promising structural lever for one-fast-many-slow; attacks the per-node ceiling	[experiment needed] on this class
Native-ROCm speculative decoding on RDNA4 (single-stream) [19, 20]	Most promising single-stream lever; overturns “AMD fails” — MoltenVK failed, not AMD	[experiment needed]; HW transfer from MI300XRDNA4 unproven
Pipelined spec-decode (SpecPipe / FlowSpec) [16, 17]	Targets our exact low-utilization pipeline shape	Inherited; Metal-leg portability uncertain [21]
KV-cache quantization (KVQuant / KIVI) [22, 23]	Buys capacity/context, orthogonal to compute ceiling	Inherited; not throughput
Dense over MoE for cross-machine pipelines	Robustness, not speed — MoE gating flips under FP drift	Measured (MoE coherence collapse)

The throughline: on the consumer AMD Radeon Pro / RDNA4 / mixed ROCm+Metal class, the path to usable single-stream throughput runs through compute — native-ROCm speculative decoding and module-asymmetric placement — not through faster links. Compute-aware partitioning is the correct foundation but caps out at the per-node ceiling; the levers that move that ceiling are exactly the ones whose consumer-RDNA4 behavior the literature has never measured, which is where we direct future work.

0.7 7. Limitations and Threats to Validity

This is an honest experience and measurement paper in the negative-results tradition [42, 43]: its contribution is a characterization of a previously unmeasured hardware class, not a generalizable SOTA claim. The findings therefore carry the threats to validity that such a genre demands, which we state explicitly here so that no measurement is read as more than it is.

0.7.1 7.1 Single-cluster, low-replication scope

All measurements come from one fixed, deliberately heterogeneous testbed (four commodity workstations with AMD Radeon Pro 5700 XT via Vulkan/MoltenVK, one with a Vega 56, one consumer Linux desktop with an RX 9070 XT via ROCm, and a single-board coordinator). They

are single-cluster, low-replication numbers, and we present them as such. The headline figures — the 4-machine Llama-3.3-70B Q4_K_M chain at ~ 0.21 tok/s, the toy 2-worker chain at ~ 1.37 – 1.51 tok/s, and the worker-to-worker push measuring slightly *slower* than client relay (0.19 vs 0.21 tok/s multi-hop; 1.37 vs 1.51 toy) — are absolute datapoints from this specific machine mix and topology, not throughput claims that transfer to other consumer clusters. A different ratio of strong-to-weak nodes, a faster mesh, or a different model quantization would move every number. The value of these figures is the *direction* they establish (per-node compute, not the ~ 16 KB/token activation wire, is the binding constraint), which is corroborated by the independent module-asymmetry finding in [6], not their precise magnitudes.

We make no statistical claims: most configurations were run a small number of times, and we report point measurements rather than distributions with confidence intervals. Properly powered, replicated timing across reboots and thermal states is [experiment needed].

0.7.2 7.2 Point-in-time software builds; several failures are version-specific

Our results are tied to specific, fast-moving software builds (e.g., llama.cpp build 8142 / commit 8c2c0108d for the Metal nondeterminism and speculative-decoding runs), and the consumer-AMD/Apple-translation stack we measure is exactly the part of the ecosystem changing fastest. Several characterized failures are platform-version-specific rather than permanent:

- The MTP speculative-decoding regression on Metal that we cite as external corroboration is an upstream report [21] that we treat as a point-in-time platform limitation, not a standing defect.
- The Qwen3-MoE partial-load patch gap that an earlier draft of this work listed as open was in fact **resolved** (two additional patches, ~ 30 – 60 LOC each); we note this as a place where our own prior draft was stale relative to the implementation.
- RDNA4 became a formal ROCm target only with 6.4.1 (May 2025) [24, 25], and bitsandbytes AMD support remains Preview-grade [26] with a documented gfx1201 LoRA failure [27]. These are moving targets: a future release may erase failure modes we report, just as it may introduce new ones.

A reader should treat each failure-mode datapoint as “observed on this build,” not “intrinsic to this silicon” — a distinction we are careful to preserve in the speculative-decoding result.

0.7.3 7.3 Metal nondeterminism precludes bit-exact reproducibility

A foundational threat to any verification story for this class is that the Metal backend is not bit-reproducible. On the Radeon Pro 5700 XT, two llama.cpp runs with identical arguments and temperature 0 produced different tokens — diverging from the first token, with the divergence visible in the logits upstream of sampling, even within a single process. This makes bit-exact equivalence proofs between a distributed chain and a single-machine reference fundamentally unsatisfiable on Metal.

This is not merely an engineering nuisance; it is grounded in floating-point non-associativity and architecture-dependent reduction/atomic ordering [40, 41]. When activations travel between AMD and Apple GPUs with different atomic and reduction orders, run-to-run and node-to-node bit-divergence is expected, not anomalous. Consequently:

- Our cross-machine correctness checks rest on weaker guarantees than bit-equality. Where we *can* assert exactness, it is at the protocol layer (e.g., the byte-equal activation exchange confirming orchestration correctness), not the numerical layer.

- The MoE coherence collapse we report (Qwen3-Coder-30B-A3B running at 4.17 tok/s with 0 errors yet emitting repetitive garbage) is a *hypothesis* — that floating-point drift flips top-k expert gating, making errors categorical rather than continuous — not a proven mechanism. We did not instrument the gating decisions per layer to confirm which experts were selected on each node; doing so is [experiment needed].

Verification of a cross-vendor activation chain is therefore intrinsically harder than in the homogeneous-NVIDIA settings prior systems were validated on, and we do not claim to have solved it.

0.7.4 7.4 Attribution and provenance caveats in the measurements

Honest characterization requires flagging where our own records were imprecise:

Datapoint	Caveat
Speculative-decoding 30x regression	The ~30x slowdown (1.2–1.3 vs 36–37 tok/s baseline) and the draft-acceptance figure are sound, but acceptance was a range (59–71%) , not a flat value; and the hardware was the Radeon Pro 5700 XT (16 GB) via MoltenVK , not the Vega 56 as a loose earlier label suggested (the 5700 XT is RDNA1, not Vega). We attribute it to “a 16 GB AMD dGPU on macOS via MoltenVK.”
Activation-size arithmetic	The 70B figure (16 KB/token/hop) is for hidden=8192 (8192 ÷ 2 bytes). The separate fabric micro-benchmark runs on a different model (Llama-3.2-3B, hidden=3072), whose per-token activation is 6 KB in fp16 — a distinct quantity that should not be conflated with the 70B’s 16 KB despite both arising in this paper.
Signed work receipts	The shipped reality is Ed25519-signed pillar calls over an SPKI/TLS-pinned, Tailscale-only mesh. “Per-session work receipts” as a distinct proof-of-work primitive are [experiment needed] , not a deployed artifact.
Micro-benchmark split provenance	The 86% / 3.21 ms embeddings read, the spec-decode tok/s figures, the MoE 4.17 tok/s rate, and the per-stage CPU-vs-Metal latencies are drawn from the deployment records underlying this report; their qualitative findings are corroborated above, but the exact per-number splits rest on those logs rather than on independent replication, and tightening them is [experiment needed].

These do not change the qualitative findings, but they bound what each number licenses.

0.7.5 7.5 Forward-looking levers are unmeasured

Several of the most consequential claims in our discussion are explicitly hypotheses we have not measured, and we mark them as such rather than extrapolate:

- Whether the native-ROCm speculative-decoding gains reported on CDNA datacenter silicon (MI300X) [19, 20] carry to consumer RDNA4 (RX 9070 XT), and whether they survive inside a distributed pipeline, is **[experiment needed]**. We deliberately do not assume the MI300X result transfers; our own MoltenVK result is the negative counterweight [21].
- The effect of module-asymmetric placement [6] under the Sthambha planner on our heterogeneous nodes is **[experiment needed]** — we motivate it as the more promising structural lever but have not run it.
- The capacity benefit of KV-cache quantization [22, 23] on our memory-starved consumer GPUs for long-context runs is **[experiment needed]**; we position it as orthogonal to (not a fix for) the compute-bound throughput ceiling.

0.7.6 7.6 External corroboration leans partly on non-archival sources

Some external claims we use to contextualize our findings draw on vendor blogs and issue trackers rather than peer-reviewed venues: the AMD ROCm speculative-decoding results are from AMD’s own ROCm blog posts [19, 20]; the Metal MTP regression, the vintage Radeon Pro MoltenVK garble, the discrete-AMD Metal garble, and the APU Vulkan garble are llama.cpp issue reports [21, 36–39]; and runtime-maturity claims rest on AMD’s compatibility matrix and a news report [24–26]. We use these as corroborating context for our own measurements, not as independent quantitative ground truth, and we flag a vendor’s self-reported speedups as such. The MoltenVK overhead and SPIR-VMSL transpilation we invoke as mechanism are, by contrast, first-party documented behavior [31–33].

0.7.7 7.7 Genre and generalizability

Finally, the genre itself bounds the claims. This is an experience and negative-results paper [42]: the deliverable is a reproducible characterization of one hardware class and a catalogue of its failure modes, framed as lessons learned. We do not claim the absolute throughput numbers generalize, that the failure set is exhaustive, or that the compute-not-wire reframing holds at every scale or topology — only that, on the consumer AMD Radeon Pro / RDNA4 / mixed ROCm+Metal class the prior literature does not measure, per-node compute was the binding constraint in every configuration we ran, and the activation wire was not.

0.8 8. Future Work

This section lays out the concrete experiments and engineering that would move a heterogeneous consumer-hardware chain from the throughput we measured (a 4-machine Llama-3.3-70B Q4_K_M chain at ~0.21 tok/s in our deployment) toward usable latency. We order the items by how directly they attack our central finding: per-node compute, not the activation wire, is the binding constraint.

0.8.1 8.1 Implement and measure compute-aware placement in Sthambha

Our planner currently assigns contiguous layer ranges, but we did not isolate the placement policy as a measured variable. Two policies from the literature should be implemented and benchmarked head-to-head on the existing testbed:

- **Water-filling allocation** [5]: assign each node a contiguous layer block proportional to its measured per-node compute capacity F_i under its VRAM cap, rather than by memory or latency. The open question for our class is estimator quality: on consumer AMD Radeon Pro (via Vulkan/MoltenVK), RDNA4 via ROCm, and Metal nodes, F_i is far harder to estimate than on the NVIDIA volunteer pools where water-filling was validated, and several of our nodes have compute that varies by runtime quirk (e.g. CPU workers measured 2–4x faster than Metal on small active-set MoE in our deployment: Metal 117/116 ms vs CPU 53/31 ms per step). Whether a single scalar F_i suffices, or whether it must be measured per-model and per-quantization, is [experiment needed].
- **Module-asymmetric placement** [6]: keep compute-intensive dense/MLP work on the stronger nodes and distribute the cheaper, parameter-free attention to the weaker nodes at sub-layer granularity. Because our finding is that the per-node compute ceiling — not allocation across whole layers — caps throughput, module-asymmetric placement is the more promising structural lever than any whole-layer scheme. Implementing it under Sthambha and measuring the throughput delta on the heterogeneous testbed is the single highest-value placement experiment, and is [experiment needed].

The honest expectation, given our results, is that compute-aware scheduling sets placement but cannot lift the per-node ceiling; module-asymmetric placement is the one policy with a path to raising effective utilization on weak nodes rather than merely re-distributing the same total work.

0.8.2 8.2 Native-ROCm speculative decoding on consumer RDNA4

Speculative decoding is the strongest single throughput lever in the literature for low-utilization, single-request pipelines [16, 18], but it ported unevenly in our experience. On a 16 GB AMD dGPU on macOS via MoltenVK, draft-based decoding measured ~30x slower than the no-draft baseline in our deployment (1.2–1.3 tok/s with draft vs. 36–37 tok/s baseline) despite 59–71% draft acceptance — a translation-layer failure, not an AMD-silicon failure. This is consistent with the upstream Metal MTP result, where draft evaluation is a net throughput loss at every configuration even at 100% acceptance [21].

The contrasting signal is that AMD reports real batch-1 speculative-decoding gains on CDNA datacenter silicon — up to 2.31x for Llama-3.1-70B/405B and batch-1 speedups of 1.26x–2.99x on MI300X [19, 20]. Whether those native-ROCm gains carry to a *consumer* RDNA4 card (the RX 9070 XT, which became a ROCm target only with the recent 6.4.1 release [24, 25]) is the open question that most directly bears on usable throughput for this hardware class. Concretely:

Experiment	Configuration	Status
Single-GPU spec-decode, native ROCm	RX 9070 XT (gfx1201), target + small draft model	[experiment needed]
Spec-decode inside a distributed pipeline	RDNA4 node as one stage, pipelined draft tree [16, 17]	[experiment needed]
Cross-vendor spec-decode (ROCm draft, Metal verify)	mixed cluster	[experiment needed]

We flag the consumer-RDNA4 software path as not turnkey: bitsandbytes treats AMD as Preview [26] and has a documented LoRA failure on gfx1201 [27], so the spec-decode experiment must be staged on a runtime path verified to function on this card first.

0.8.3 8.3 Single-node SpecExec baseline as the bar to beat

Before investing further in a distributed chain, the chain must be justified against a strong single-consumer-GPU baseline. SpecExec reports 50B+ inference at 4–6 tok/s (4-bit) on a single consumer GPU with RAM offloading via massively parallel speculative decoding [18]. Our 4-machine 70B chain at ~0.21 tok/s in our deployment is far below that bar. We should therefore run SpecExec (or an equivalent single-GPU offload-plus-spec-decode configuration) on our RDNA4 node as a controlled baseline; the distributed chain only earns its existence where it beats the best single-node number a worker can achieve alone. Quantifying this gap on our exact hardware is [experiment needed], and SpecExec’s NVIDIA-offload focus means the consumer-AMD result is not predictable from the published numbers.

0.8.4 8.4 Hardening MoE cross-machine execution

Our most categorical failure was MoE coherence collapse: a Qwen3-Coder-30B-A3B chain across four Macs ran at 4.17 tok/s with zero transport errors yet emitted repetitive garbage, while dense models survived the same cross-machine numerical drift in our deployment. Our hypothesis is that floating-point non-associativity across heterogeneous reduction/atomic orders [40, 41] flips top-k expert-gating decisions, making the error categorical rather than continuous. Two mitigations are worth implementing and measuring:

1. Restrict MoE routing-and-expert stages to a single node (cross-machine handoffs only at layer boundaries where gating is already resolved), and measure whether coherence is restored.
2. Default to dense models for cross-machine chains, treating dense-over-MoE as the robust configuration and quantifying the throughput cost of that choice.

Both are [experiment needed].

0.8.5 8.5 Maturing the activation fabric and signed work-receipts

Our custom UDP activation transport initially hit ~75% multi-chunk packet loss over the mesh, reduced to 0 in cluster smoke tests via receive-buffer and send-pacing knobs in our deployment. The fabric micro-benchmark further showed that the GPUhost embeddings read, not the matmul, dominated the per-token round trip (3.21 ms of 3.7 ms, 86%, on the RX 9070 XT under ROCm in our deployment). Future fabric work should add explicit back-pressure and pacing as first-class flow control, and investigate overlapping the embeddings DMA read with the next stage’s compute, since that read — not the wire — is where the time goes. On security, the shipped reality is Ed25519-signed pillar calls over an SPKI/TLS-pinned, mesh-only authenticated channel; promoting these into a real signed-work-receipt proof-of-work primitive remains [experiment needed].

0.8.6 8.6 KV-cache quantization for per-node context capacity

KV-cache quantization [22, 23] is orthogonal to our compute-bound throughput ceiling but directly relevant to per-node capacity: in a layer-pipeline each node holds KV only for its own layers, so sub-4-bit KV quantization would let memory-starved consumer GPUs (the 8 GB Vega 56, the 16 GB Radeon Pro 5700 XT) hold longer-context KV locally. Quantifying its effect on per-node context length for long-context runs on this hardware is [experiment needed]; we expect it to extend capacity, not raise the single-request throughput ceiling.

0.9 9. Conclusion

We set out to measure the hardware class the distributed-inference literature skips. Every published result we are aware of is benchmarked on NVIDIA datacenter or consumer GPUs, Apple Silicon, or AMD CDNA accelerators (MI300X) [2, 4–6, 19]; none characterizes old AMD Radeon Pro GPUs driven through Vulkan-over-Metal (MoltenVK), a consumer RDNA4 card on freshly added ROCm, or clusters that mix these ROCm and Metal runtimes over a mesh. Building on the Petals-derived weights-stay-local / activations-travel pipeline [1, 2] and adding the Sthambha compute-aware planner, we delivered the first reproducible characterization of distributed LLM inference on that consumer Radeon-Pro / RDNA4 / mixed-ROCM+Metal class.

Our central, measurement-backed reframing is that **the binding constraint is per-node compute, not the activation wire**. Per-token activation traffic is trivial — 16 KB/token/hop for a 70B (hidden=8192, fp16), on the order of 400 KB/s across a 5-hop pipeline in our deployment — and a worker-to-worker push that removes a client-relay network hop measured *slightly slower*, not faster, than client relay (0.19 vs. 0.21 tok/s on the 4-machine 70B chain; 1.37 vs. 1.51 tok/s on the toy 2-worker chain). The fabric micro-benchmark told the same story from the other side: the GPUhost embeddings read, not the matmul, dominated the round trip. This reframing redirects effort away from faster links and toward compute-aware scheduling and, more promisingly, module-asymmetric placement [6].

The honest contribution is the negative results and the catalogued failure surface [42]: a 4-machine 70B chain at ~0.21 tok/s; MoE cross-machine coherence collapse where dense models survive but expert-gated models emit categorical garbage; CPU workers beating Metal 2–4x on small active-set MoE; Metal nondeterminism at temperature 0 and a page-alignment last-worker buffer assert [38]; a ~30x speculative-decoding regression on the MoltenVK translation layer despite 59–71% draft acceptance [21]; and ~75%-to-0% UDP burst-loss mitigation — each tied to its configuration and source. These are point-in-time, single-cluster measurements on a small testbed, and some failures are software-version-specific rather than permanent; we have marked the experiments we did not run as [experiment needed] rather than extrapolating.

The path to usable throughput on this hardware runs through native-ROCM speculative decoding on consumer RDNA4 and module-asymmetric placement — not faster links. More broadly, the value of this work is the value of honest measurement on the hardware enthusiasts actually own: the consumer-AMD and Apple-translation regime behaves differently enough from the NVIDIA/CDNA stacks the literature assumes that conclusions drawn there do not transfer for free. Characterizing where they break is a prerequisite for anyone trying to run large models on commodity, heterogeneous consumer GPUs.

References

- [1] A. Borzunov, D. Baranchuk, T. Dettmers, M. Ryabinin, Y. Belkada, A. Chumachenko, P. Samygin, and C. Raffel. "Petals: Collaborative Inference and Fine-tuning of Large Models." *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, 2023. arXiv:2209.01188.
- [2] A. Borzunov, M. Ryabinin, A. Chumachenko, D. Baranchuk, T. Dettmers, Y. Belkada, P. Samygin, and C. Raffel. "Distributed Inference and Fine-tuning of Large Language Models Over The Internet." *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. arXiv:2312.08361.
- [3] A. Borzunov, M. Ryabinin, A. Chumachenko, D. Baranchuk, T. Dettmers, Y. Belkada, P.

- Samygin, and C. Raffel. "Distributed Inference and Fine-tuning of Large Language Models Over The Internet." *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023. arXiv:2312.08361.
- [4] Z. Li, T. Li, W. Feng, R. Xiao, J. She, H. Huang, M. Guizani, H. Yu, Q. Ho, W. Xiang, and S. Liu. "Prima.cpp: Fast 30-70B LLM Inference on Heterogeneous and Low-Resource Home Clusters." arXiv preprint arXiv:2504.08791, 2025.
 - [5] C. Tong, Y. Jiang, G. Chen, T. Zhao, S. Lu, W. Qu, E. Yang, L. Ai, and B. Yuan. "Parallax: Efficient LLM Inference Service over Decentralized Environment." arXiv preprint arXiv:2509.26182, 2025.
 - [6] Z. Mo, J. Liao, H. Xu, Z. Zhou, and C. Xu. "Hetis: Serving LLMs in Heterogeneous GPU Clusters with Fine-grained and Dynamic Parallelism." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25)*, 2025. doi:10.1145/3712285.3759784, arXiv:2509.08309.
 - [7] exo Labs. "exo: Run your own AI cluster at home with everyday devices." 2024. <https://github.com/exo-explore/exo> (accessed 2026-06-02).
 - [8] B. Tadych. "Distributed Llama: Connect home devices into a powerful cluster to accelerate LLM inference." 2024. <https://github.com/b4rtaz/distributed-llama> (accessed 2026-06-02).
 - [9] S. Cirillo (evilsocket). "Cake: Distributed inference for mobile, desktop and server." 2024. <https://github.com/evilsocket/cake> (accessed 2026-06-02).
 - [10] Kalavai. "Kalavai: Aggregate compute from spare GPU capacity into a unified AI cluster." 2024. <https://github.com/kalavai-net/kalavai-client> (accessed 2026-06-02).
 - [11] Wavefy. "Wavefy: Decentralized LLM Inference." 2024. <https://github.com/wavefy/decentralized-llm-inference> (accessed 2026-06-02).
 - [12] llama.cpp contributors. "llama.cpp RPC Backend (rpc-server)." GitHub, ggml-org/llama.cpp, tools/rpc/README.md, 2024/2026. <https://github.com/ggml-org/llama.cpp/blob/master/tools/rpc/README.md> (accessed 2026-06-02).
 - [13] Gensyn. "Gensyn Litepaper." 2022. <https://docs.gensyn.ai/litepaper> (accessed 2026-06-02).
 - [14] A. Long. "Protocol Learning, Decentralized Frontier Risk and the No-Off Problem." arXiv preprint arXiv:2412.07890, 2024.
 - [15] S. Jaghouar, J. M. Ong, M. Basra, F. Obeid, J. Straube, M. Keiblinger, E. Bakouch, L. Atkins, M. Panahi, C. Goddard, M. Ryabinin, and J. Hagemann. "INTELLECT-1 Technical Report." arXiv preprint arXiv:2412.01152, 2024.
 - [16] H. Yin, M. Xiao, T. Li, X. Zhang, D. Yu, and G. Zhang. "SpecPipe: Accelerating Pipeline Parallelism-based LLM Inference with Speculative Decoding." arXiv preprint arXiv:2504.04104, 2025.
 - [17] X. Liu, L. Luo, M. Tang, C. Huang, and X. Chen. "FlowSpec: Continuous Pipelined Speculative Decoding for Efficient Distributed LLM Inference." arXiv preprint arXiv:2507.02620, 2025.

- [18] R. Svirschevski, A. May, Z. Chen, B. Chen, Z. Jia, and M. Ryabinin. "SpecExec: Massively Parallel Speculative Decoding for Interactive LLM Inference on Consumer Devices." *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2406.02532.
- [19] C. Liu. "Speculative Decoding - Deep Dive." ROCm Blogs, Advanced Micro Devices, March 2025. <https://rocm.blogs.amd.com/software-tools-optimization/speculative-decoding---deep-dive/README.html> (benchmarked on AMD Instinct MI300X with vLLM v0.6.7, ROCm 6.3.1; up to 2.31x speedup on Llama 3.1-70B/405B).
- [20] S. Singh, K. Sangaiah, S. Zhang, R. Swann, and G. Dasika. "Accelerating LLM Inference: Up to 3x Speedup on MI300X with Speculative Decoding." ROCm Blogs, Advanced Micro Devices, March 2025. https://rocm.blogs.amd.com/artificial-intelligence/spec_decode_mi300x/README.html (batch-size-1 token-generation speedups of 1.26x2.99x on MI300X across PyTorch gpt-fast and vLLM, eager vs compiled).
- [21] thewesjohnson. "MTP speculative decoding degrades throughput on Metal (Apple Silicon): net loss at every configuration." llama.cpp GitHub issue #23752, ggml-org/llama.cpp, May 2026. <https://github.com/ggml-org/llama.cpp/issues/23752> (opened 27 May 2026; reports 11%28% throughput loss vs baseline across all tested configs on Apple Silicon Metal, including at 100% draft acceptance).
- [22] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami. "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization." *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2401.18079.
- [23] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu. "KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache." *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. arXiv:2402.02750.
- [24] Advanced Micro Devices, Inc. "ROCm Compatibility Matrix." AMD ROCm Documentation, 2026. <https://rocm.docs.amd.com/en/latest/compatibility/compatibility-matrix.html> (lists gfx1201 / RX 9070 XT, RDNA4, as a supported target across the ROCm 7.x line; accessed 2026-06-02).
- [25] M. Larabel. "AMD ROCm 6.4.1 Released." Phoronix, May 2025. <https://www.phoronix.com/news/AMD-ROCM-6.4.1-Released> (first ROCm release with formal RDNA4 / RX 9070 series support; accessed 2026-06-02).
- [26] bitsandbytes contributors. "bitsandbytes Documentation: Installation." Hugging Face Documentation, 2026. <https://huggingface.co/docs/bitsandbytes/main/en/installation> (prebuilt Linux x86-64 wheels include gfx1201 from ROCm 6.4.4 onward; AMD support marked "Preview"; accessed 2026-06-02).
- [27] bitsandbytes contributors. "Issue #1464: gfx1201 LoRA failure 'invalid device function' / hipBLASLt on unsupported architecture." GitHub, bitsandbytes-foundation/bitsandbytes, 2026. <https://github.com/bitsandbytes-foundation/bitsandbytes/issues/1464> (accessed 2026-06-02).
- [28] Apple Inc. "Metal Overview." Apple Developer, 2026. <https://developer.apple.com/metal/> (accessed 2026-06-02).

- [29] Apple Inc. "Performing Calculations on a GPU." Apple Developer Documentation, 2026. <https://developer.apple.com/documentation/metal/performing-calculations-on-a-gpu> (accessed 2026-06-02).
- [30] Apple Inc. "Metal Performance Shaders." Apple Developer Documentation, 2026. <https://developer.apple.com/documentation/metalperformanceshaders> (accessed 2026-06-02).
- [31] The Khronos Group. "MoltenVK: A Vulkan Portability Implementation over Apple Metal." GitHub, KhronosGroup/MoltenVK, 2026. <https://github.com/KhronosGroup/MoltenVK> (accessed 2026-06-02).
- [32] The Khronos Group. "MoltenVK Runtime User Guide." GitHub, KhronosGroup/MoltenVK, 2026. https://github.com/KhronosGroup/MoltenVK/blob/main/Docs/MoltenVK_Runtime_UserGuide.md (documents added memory/performance overhead, SPIR-V to MSL shader conversion, and pipeline caching; accessed 2026-06-02).
- [33] LunarG, Inc. "Getting Started with the macOS Vulkan SDK." LunarG Vulkan SDK Documentation, 2026. https://vulkan.lunarg.com/doc/view/latest/mac/getting_started.html (on macOS, Vulkan is provided via MoltenVK over Metal rather than a native driver; accessed 2026-06-02).
- [34] G. Gerganov and llama.cpp contributors. "llama.cpp: LLM Inference in C/C++." GitHub, ggml-org/llama.cpp, 2026. <https://github.com/ggml-org/llama.cpp> (accessed 2026-06-02).
- [35] ggml contributors. "GGUF: A File Format for Storing Models for GGML Inference." GitHub, ggml-org/ggml, docs/gguf.md, 2026. <https://github.com/ggml-org/ggml/blob/master/docs/gguf.md> (accessed 2026-06-02).
- [36] llama.cpp contributors. "Misc. bug: Vulkan backend produces garbled output on vintage AMD Radeon Pro 580X since aa6f918c." GitHub, ggml-org/llama.cpp, Issue #20465, 2026. <https://github.com/ggml-org/llama.cpp/issues/20465> (garbled output via MoltenVK after a Flash-Attention refactor; closed as duplicate of #20029; accessed 2026-06-02).
- [37] llama.cpp contributors. "Misc. bug: b8143 produces garbage Mac x86 Vulkan with AMD GPU." GitHub, ggml-org/llama.cpp, Issue #20029, 2026. <https://github.com/ggml-org/llama.cpp/issues/20029> (regression after build b8143: corrupted output on macOS x86 + AMD Radeon RX 6900 XT via Vulkan/MoltenVK; b8142 fine; accessed 2026-06-02).
- [38] llama.cpp contributors. "bug: Metal backend produces garbage output on AMD discrete GPUs (Radeon Pro 5300M)." GitHub, ggml-org/llama.cpp, Issue #19563, 2026. <https://github.com/ggml-org/llama.cpp/issues/19563> (garbled output on a discrete AMD Radeon Pro 5300M via the Metal backend; CPU and Vulkan paths fine; likely CPUGPU transfer on non-unified-memory AMD; accessed 2026-06-02).
- [39] llama.cpp contributors. "Eval bug: Gemma 3n on Vulkan on Ryzen APUs produces garbled output." GitHub, ggml-org/llama.cpp, Issue #14525, 2026. <https://github.com/ggml-org/llama.cpp/issues/14525> (garbled output on integrated AMD Radeon, RADV RENOIR/PHOENIX, via Vulkan; accessed 2026-06-02).
- [40] S. Shanmugavelu, M. Taillefumier, C. Culver, O. Hernandez, M. Coletti, and A. Sedova. "Impacts of Floating-Point Non-Associativity on Reproducibility for HPC and Deep Learning Applications." arXiv preprint arXiv:2408.05148, 2024.

- [41] Advanced Micro Devices, Inc. "Hardware Atomics Operation Support." AMD ROCm Documentation, 2026. <https://rocm.docs.amd.com/en/latest/reference/gpu-atomics-operation.html> (per-architecture support and limits for FP atomic add/min/max; basis for atomic-order-dependent nondeterminism; accessed 2026-06-02).
- [42] E. Peltonen, N. Mohan, P. Zdankin, T. Shreedhar, T. Nguyen, S. Bayhan, J. Crowcroft, J. Kangasharju, and D. Nicklas. "Perspectives on Negative Research Results in Pervasive Computing." *Proceedings of the First International Workshop on Negative Results in Pervasive Computing (PerFail)*, co-located with IEEE PerCom, 2022. arXiv:2210.05708.
- [43] P. J. Maliakel, S. Ilager, and I. Brandic. "Characterizing LLM Inference Energy-Performance Tradeoffs across Workloads and GPU Scaling." 2025. arXiv:2501.08219.