memLLM-net:

An Empirical Comparison of Wi-Fi, Shared-Memory, and RDMA Transports for On-Device LLM Context Delivery

Hari Prasad Sampatirao
Walsh College, Troy, Michigan
github.com/hari-sampatirao/memLLM-net

Companion study to MemLLM [1]
August 2026

Abstract

Beyond a single machine is where MemLLM's [1] central argument stops holding: it argued that zero-copy shared memory eliminates the serialization overhead dominating on-device LLM inference, a claim scoped, by construction, to one machine. memLLM-net tests what happens at that scope's edges. We build and validate, on real hardware: (i) an eRPC-style reliable transport for the case where the application and inference server are not co-located, exercised over synthetic loss and then a real home Wi-Fi link between two physical devices; (ii) the Linux `memfd_create`/`eventfd` shared-memory path MemLLM always described but never measured, completing its own stated evaluation gap; and (iii) a real RDMA transport via Soft-RoCE, requiring no specialized hardware. Five genuine defects surfaced and were fixed in building the Wi-Fi transport, including a retry-storm failure mode that reproduces, and complicates, the standard claim that RoCE livelocks on lossy links. Real shared memory beats the Wi-Fi-capable transport by roughly 4× and real RDMA verbs beat our own Python shared-memory implementation by roughly 36× on the same machine — a measured ceiling this line of work falls well short of. We further argue, against an earlier framing of our own, that the KV cache should almost never cross the network at all: a thin client tracking only *what changed*, not the cache itself, is sufficient for the common architecture, which reframes what "KV-cache-over-Wi-Fi" should actually mean.

Keywords — KV cache, shared memory, memif, eRPC, RoCE, Soft-RoCE, Wi-Fi, RDMA, on-device inference, vLLM

1 Introduction

MemLLM's central argument is that a directly-mapped shared memory region eliminates the serialization and copy overhead that dominates multi-turn, on-device LLM inference [1]. That argument is scoped to one machine by construction: two processes can only map the same physical pages when they share a kernel. The paper's own roadmap named two open directions — a real Linux implementation (only a Windows prototype was ever benchmarked) and the multi-device case, where MemLLM's own inspiration, the Linux *memif* shared-memory interface [2], comes from a lineage — high-performance packet I/O — that itself grew out of the same question networking research answered for RDMA over Converged Ethernet (RoCE): what replaces shared memory once two endpoints are no longer the same machine.

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

This paper answers both directions with measurements rather than projections, on a single consistent platform: a Linux workstation, a Windows laptop, an ordinary home Wi-Fi network, and a live vLLM server as a real inference backend alongside a controlled synthetic one. Three systems were built and validated:

1. an eRPC-style [3] reliable UDP transport for the cross-device case, where shared memory is physically impossible;
2. the real Linux `memfd_create` + `SCM_RIGHTS` + `eventfd` path MemLLM described but never built;
3. Soft-RoCE — real RDMA verbs semantics with no specialized NIC — to locate RDMA precisely relative to both.

A fourth contribution is architectural rather than empirical: for the common case — a thin client talking to a fixed remote inference server — the KV cache should **not** be transferred at all. Section 3 makes this case before any transport numbers are presented, because it changes what "helping with the KV cache over Wi-Fi" should even mean.

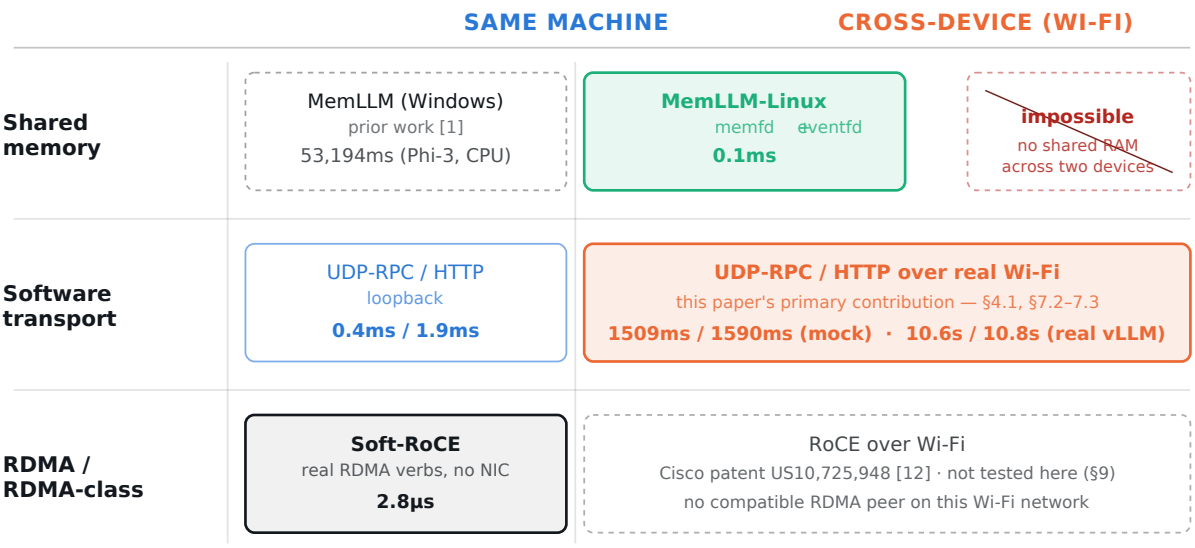

Figure 1: Every mechanism in this paper, placed on one map. Prior work [1] is dashed; this paper's own measurements are solid; the orange cell is the paper's central, validated contribution; the untested cell names exactly why (Section 9).

2 Background

2.1 MemLLM and memif

MemLLM [1] maps a shared region between an application process and an inference server, exchanging token descriptors and payloads through an SPSC ring with no per-turn serialization. Its design borrows directly from *memif* [2], a shared-memory interface originated at Cisco for VPP/DPDK userspace packet I/O — memfd-backed regions, fixed descriptor rings, eventfd-style notification. Both are same-host mechanisms: the entire performance argument rests on two processes mapping

identical physical pages, which requires a shared kernel and cannot be extended across a network by definition, not by engineering effort.

2.2 RoCE, eRPC, and the wireless gap

RDMA over Converged Ethernet extends memory-semantic access across a network fabric, but assumes the fabric is close to lossless, maintained by link-layer flow control (PFC/ECN). Applying RDMA directly to a wireless link removes that assumption without replacing it: prior work documents a livelock failure mode where a single dropped packet leaves the link fully utilized while the receiver never completes a message [4]. Industry has recognized the same gap: Cisco holds a granted patent [12] on making RoCE viable over Wi-Fi specifically, via MAC/PHY-layer mechanisms — pre-queuing placeholder frames to start 802.11 contention countdown early, and spreading redundant, XOR-coded packet fragments across non-contiguous OFDM subcarriers so narrowband interference cannot take out a whole packet. Both require driver- or firmware-level control over the radio. eRPC [3] takes the opposite position and layer — reliability implemented entirely in userspace over commodity UDP sockets, assuming loss as normal rather than exceptional, needing no specialized hardware or driver access — and is the design lineage this paper's Wi-Fi transport follows.

2.3 Why Wi-Fi is lossy, physically

This is not incidental. 802.11's CSMA/CA medium access is contention-based and only *avoids* collisions probabilistically, unlike wired Ethernet's now-obsolete collision-detection scheme — a busy channel means ongoing real loss, not an edge case. Consumer Wi-Fi also shares an already-crowded unlicensed band: overlapping channels from neighboring access points, Bluetooth and microwave interference on 2.4GHz, and distance-driven multipath fading all degrade the link independently of either endpoint. Rate adaptation then converts interference into latency variance as well as loss, as a station falls back to slower, more robust modulation. Loss and jitter are Wi-Fi's normal operating regime, which is precisely why a transport for it has to be validated against both, not merely assumed reliable once "made reliable" in the abstract.

3 Why the Cache Shouldn't Travel

Before any transport is described, it is worth being precise about what actually needs to cross a network link in the first place — because the answer is smaller than "KV-cache-over-Wi-Fi" suggests, and it shapes every design choice that follows.

Every transport result in this paper moved messages of this shape — a few hundred bytes, not cache tensors — and that was the correct design, not a shortcut around a harder test. For a fixed remote inference server, the client's role is genuinely light: hold the conversation as plain text for its own interface (kilobytes, trivial on any device with no GPU at all), and forward only what changed. In the common, linear, append-only case, "what changed" is trivial — it is simply whatever the user just typed, which the client already has for free.

This does mean the client needs to track *identity*, never *contents*, once a conversation stops being perfectly linear: an edited or branched turn needs a sequence number or hash so the server knows which cached suffix to discard; multiple devices sharing one context need to know how much of the server's cached prefix is still valid, which is exactly the problem RadixAttention's prefix hashing [6] solves server-side, with the client's part reduced to an identifier.

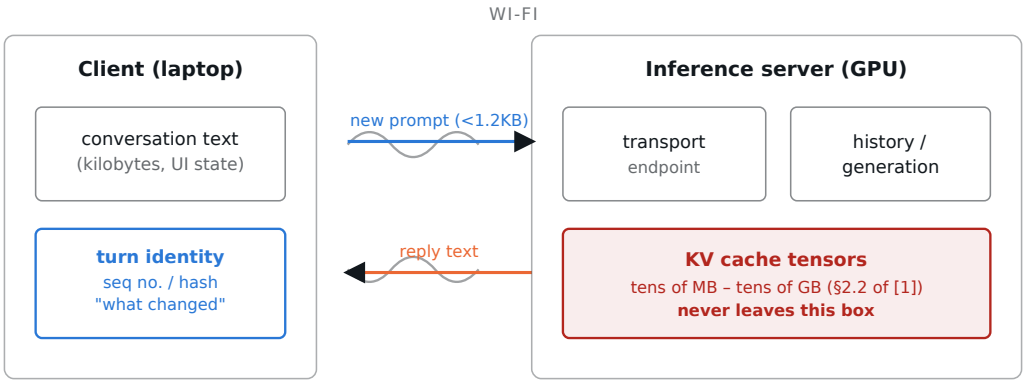

Figure 2: The architecture this paper actually validates. The client sends only the newest turn's text and tracks turn identity, not cache contents; the KV cache is produced and consumed entirely on the inference server and is never a network payload.

Real KV cache *migration* — the tensors physically crossing a network — is a genuine technique, but belongs to narrower and different problems than "a thin client talks to a fixed remote GPU":

1. **Prefill/decode disaggregation** (Mooncake [8], DistServe) moves cache between differently-optimized serving nodes *within a datacenter*, over RDMA on wired infrastructure built for it — not a home Wi-Fi problem at all.
2. **Compute handoff mid-conversation** (e.g. phone to laptop) is the one case that plausibly touches Wi-Fi, and even there, transferring the cache is one option, not the only one: the raw conversation text is kilobytes, the cache is megabytes-to-gigabytes, so simply resending the text and letting the new device recompute its own cache is usually cheaper than shipping the cache itself — unless recomputing prefill is prohibitively slow (very long context, a weak destination device), which is a real but narrower condition.
3. **Layer-split pipeline parallelism**, where a single model's layers run on two devices, requires activations to cross the link on *every token*, continuously — a categorically harder and more frequent transfer problem than a one-time migration, and, if anything, a stronger argument against attempting it over Wi-Fi.

None of these three is what this paper tests, and none needed to be. Sections 4–7 validate the transport for the architecture in Figure 2 — the one that actually applies to a thin client using a remote GPU over Wi-Fi.

4 System Design

4.1 eRPC-style transport for the cross-device case

The Wi-Fi transport reuses MemLLM's descriptor philosophy — a fixed-size header plus a payload pool — serialized onto plain UDP datagrams, since no directly-mapped address space exists across a wireless hop. Three choices follow eRPC [3] rather than RoCE:

1. **Reliability in userspace.** Per-fragment selective-repeat ACKs (a bitmap, not a cumulative

sequence number) mean one lost fragment costs one retransmission, not a full-message resend.

2. **Run-to-completion polling.** Both sides busy-poll a non-blocking socket rather than block on `recv()` — the same latency-vs-CPU tradeoff MemLLM's Windows prototype makes with `threading.Event` polling in place of `eventfd`.
3. **Exponential backoff with jitter** on retransmission, added after Finding 3 in Section 6 — the one point where the design diverges from a naive first pass and converges on what TCP and real eRPC already do.

4.2 The real Linux shared-memory path

MemLLM always described a Linux path — `memfd_create(2)` for the region, `SCM_RIGHTS` over `AF_UNIX` for fd transfer, `eventfd` for notification — but only the Windows named-shared-memory prototype was ever benchmarked. This work implements it, reusing the identical control-block and descriptor layout from the Windows path, so the "same physical layout and ring protocol" claim [1] is now literally true rather than asserted for a path that didn't exist.

4.3 Soft-RoCE

To locate RDMA precisely rather than cite its reputation, Soft-RoCE (`rdma_rxe`) — a kernel module presenting a real RDMA verbs device (queue pairs, memory registration, completion queues) over an ordinary NIC, no specialized hardware required — was brought up on the Linux workstation and measured directly with standard RDMA microbenchmarks.

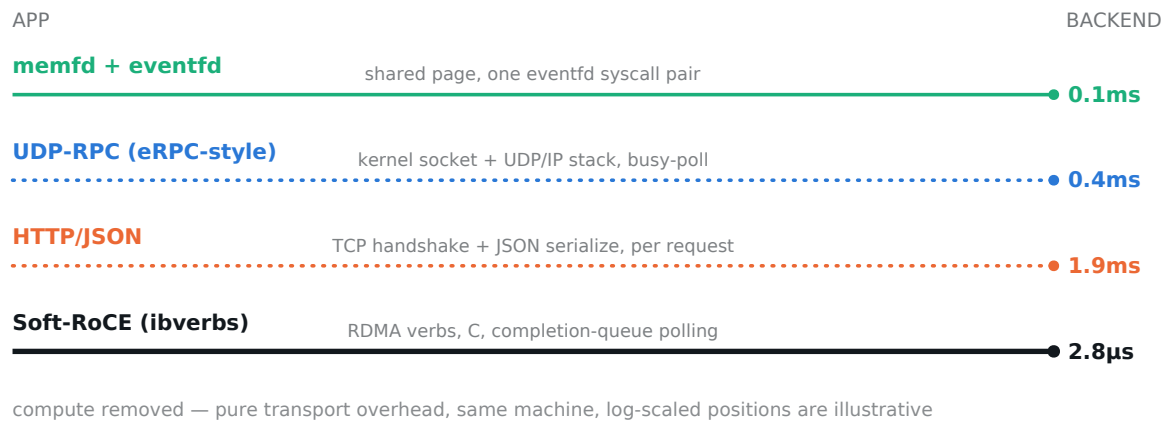

Figure 3: Four paths between the same two endpoints on one machine, and what each one costs once model compute is held constant. The gap from HTTP/JSON down to Soft-RoCE spans roughly three orders of magnitude.

5 Experimental Setup

All measurements ran on a single consistent platform: a Linux workstation (NVIDIA GB10) as the fixed inference server, a Windows laptop as the Wi-Fi client, connected over an ordinary home Wi-Fi network (measured baseline RTT 18.5ms, `ping`, n=2). A live, unmodified vLLM server (Qwen2.5-7B-Instruct, OpenAI-compatible API, Apache 2.0 licensed) served as a real inference

backend alongside a synthetic mock backend used to isolate transport behavior from compute variance, matching generation time statistically across both transports under test. Reaching the real-Wi-Fi configuration required reconnecting a dropped Wi-Fi adapter and opening LAN-scoped firewall rules on the server host — friction that HTTP-over-Ethernet mostly lets users forget about, and worth naming as a real deployment cost.

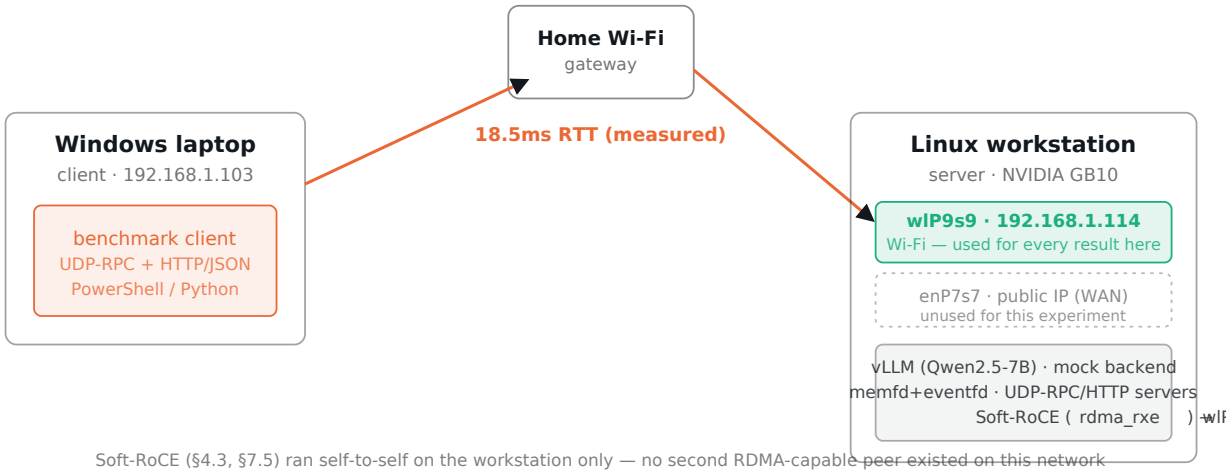

Figure 4: The physical setup behind every measurement. The workstation carries two interfaces; only the Wi-Fi-facing one (`wlP9s9`) is exercised by the results in this paper — reaching it required reconnecting a dropped Wi-Fi adapter and opening LAN-scoped firewall rules, a real cost this diagram makes visible.

6 What Building It Found

Five defects surfaced across three validation stages — synthetic backend, synthetic loss on loopback, then two physical devices on real Wi-Fi — that loopback testing at zero loss would not have exposed.

1. **Duplicate delivery under ACK loss.** A lost ACK causes the sender to retransmit an already-delivered message. Without dedup, this produced both a stale reply silently misdelivered to a later, unrelated caller, and a `KeyError` that silently killed the server thread once a duplicate was queued twice. Fixed by tracking delivered message IDs and re-acknowledging duplicates idempotently, without re-queueing them.

2. **Session-scoped identity.** A long-lived server plus a freshly restarted client reusing small integer message IDs collided with the previous session's dedup cache — the server silently ACKed the request without ever generating a real response. Sequence numbers need session scope, not just uniqueness.

3. **Fixed retry intervals cause retry storms.** Under synthetic loopback loss, a fixed 20ms retransmit timer produced far worse than predicted failure at high loss — **≈80% message failure at a nominal 30% loss**, versus a near-zero first-order estimate — because both endpoints' retransmissions outpaced the busy-poll drain rate. Exponential backoff with jitter restored **100% delivery at the same loss rate**. This is the single most RoCE-relevant finding: concrete evidence that a viable software transport needs real congestion control, not

merely "add retries."

4. **Fault isolation is not free.** One response exhausting its retry budget raised an uncaught exception that took the entire server down, not just that message. HTTP/TCP gives per-connection fault isolation for free at the kernel layer; a hand-rolled UDP transport has to be built to survive one failed message.

5. **A fast synthetic backend hides a real one's failure mode.** Once pointed at the live vLLM server, real 7B decode latency (6–14s/turn) exceeded the client's 30s receive-timeout margin often enough that the client gave up and issued the next turn's request while the server was still blocked inside the previous turn's non-polling `generate()` call — a pipeline desync that looked identical to a reliability failure but had nothing to do with loss. Widening the timeout past the backend's observed worst case resolved it.

   Building the Linux path, separately: Two further bugs surfaced implementing the `memfd`/`eventfd` path: `socket.recv_fds()`'s return tuple was unpacked in the wrong order (silently measuring a 1-byte placeholder instead of the file-descriptor list), and a `ctypes`-created buffer view had to be explicitly released before `mmap.close()` would succeed. Both are the kind of mistake that only surfaces by actually building the thing.

7 Results

7.1 The full latency spectrum

Figure 5 places every measurement in this study and its predecessor [1] on one log-scaled comparison, spanning roughly seven orders of magnitude from a CPU-bound cloud-style HTTP baseline down to raw RDMA verbs.

7.2 Real Wi-Fi, controlled backend

With generation time held statistically identical across transports, any latency difference is attributable to the transport alone.

Table 1: UDP-RPC vs. HTTP/JSON, controlled mock backend.

Condition	N	Mean	Median	stdev	p95	Retrans.
UDP-RPC, loopback	10	1485ms	1507ms	327ms	2006ms	0
UDP-RPC, real Wi-Fi	10	**1509ms**	1538ms	333ms	**2020ms**	3
HTTP/JSON, real Wi-Fi	10	1590ms	1604ms	308ms	2096ms	—

Real-Wi-Fi overhead over loopback was ≈25ms mean — consistent with the measured 18.5ms link RTT plus one retransmission round, not a multiplicative blowup. UDP-RPC ran ≈80ms lower than HTTP/JSON on both mean and p95, the same direction as MemLLM's local shared-memory-vs-HTTP result [1], now reproduced across a wireless hop.

7.3 Real Wi-Fi, real inference

UDP-RPC still ran marginally lower and absorbed two genuine Wi-Fi packet losses with no failed turns, but the gap shrinks to a rounding error against 6–14s of real decode. This revised an initial

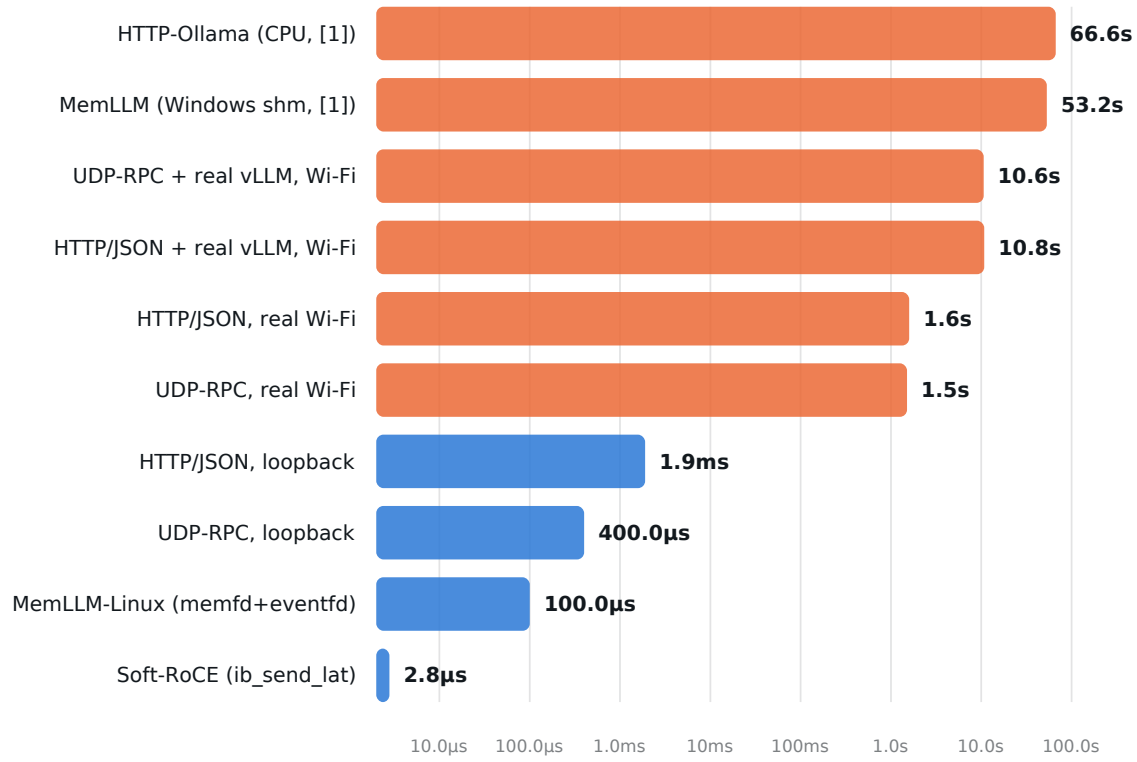

Figure 5: Mean latency across every mechanism measured, log-scaled. Orange marks require a second physical device (cross-device); blue stays on one machine. The RoCE 40%-loss condition (Table 4) is a connection-setup failure, not a completed measurement, and is omitted from this bar chart accordingly.

Table 2: UDP-RPC vs. HTTP/JSON, real Qwen2.5-7B-Instruct via vLLM.

Condition	N	Mean	Median	stdev	p95	Retrans.
UDP-RPC, real Wi-Fi + vLLM	10	**10,613ms**	12,071ms	3570ms	**13,720ms**	2
HTTP/JSON, real Wi-Fi + vLLM	10	10,769ms	12,557ms	3809ms	13,712ms	—

hypothesis that faster GPU decode would make transport choice *more* visible; the honest result is that this accelerator's decode speed was not fast enough to flip that balance — the same regime as the CPU-bound baseline in [1], for a different underlying reason.

7.4 Same machine, compute removed

Table 3: Pure transport overhead, one machine, synthetic near-zero-latency backend.

Mechanism	Mean	p95	Implementation
MemLLM-Linux (`memfd+eventfd`)	**0.1ms**	0.2ms	Python
UDP-RPC (loopback)	0.4ms	0.7ms	Python
HTTP/JSON (loopback)	1.9ms	14.8ms	Python

Real shared memory is ≈4× faster than the eRPC-style transport and ≈19× faster than HTTP/JSON on mean, with a far tighter tail — HTTP's p95 balloons to 14.8ms, plausibly from per-request TCP connection overhead neither UDP nor shared memory pays.

7.5 Where RDMA actually sits

Table 4: Soft-RoCE (`ib_send_lat`) under synthetic loss on the interface the traffic actually traversed.

Loss	Mean latency	Result
0%	**2.8µs**	1000 / 1000 iterations
20%	4.7µs (+67%)	1000 / 1000 iterations, no failures
40%	—	connection *setup* failed (reproduced twice)

Real RDMA verbs — even software-emulated, no specialized NIC — beat this paper's own Python `memfd`/`eventfd` implementation by roughly 36×, a gap almost certainly dominated by C-versus-Python and completion-queue polling versus per-operation syscalls, not by anything fundamental to shared memory as a mechanism. At 20% loss, the RDMA Reliable Connection transport's own retransmission logic absorbed the loss with a graceful increase and zero failures — more resilient than a naive reading of "RoCE assumes a lossless fabric" predicts. The failure at 40% is more precise than "RoCE breaks under loss": the benchmark tool's plain, non-redundant TCP side-channel for exchanging queue-pair parameters failed during connection bootstrap, not the RDMA data path mid-transfer (Figure 6). This nuances, rather than confirms, this paper's own framing in Section 2.2 — the vulnerability demonstrated is in naive control-plane tooling, not necessarily in the reliability mechanism RDMA's data path itself provides where it was actually exercised.

8 Synthesis

9 Limitations and Future Work

Loss injection was synthetic and loopback/local-interface only, not a contended wireless channel with the correlated bursts and rate-adaptation jitter Section 2.3 describes; the real-Wi-Fi run used a single link, a single device pair, and n=10 turns; the RoCE result is same-machine only, since

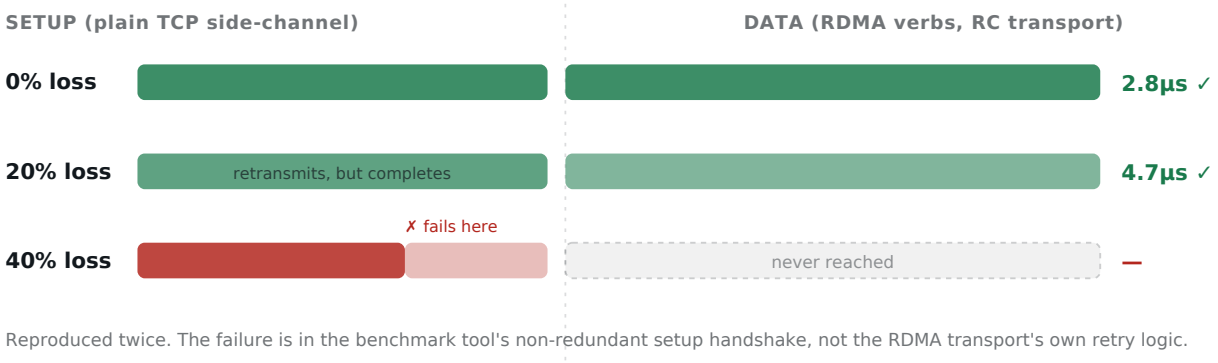

Figure 6: Where Soft-RoCE actually breaks under loss. The RDMA data path (right) was never reached at 40% loss — the plain TCP handshake used to bootstrap the connection (left) failed first, both times.

Table 5: Direct answers.

Question	Answer
Does MemLLM's own mechanism help over Wi-Fi?	No — shared memory needs shared RAM, which a second device removes by definition.
Does its design philosophy, as a wire protocol, help?	Modestly — ≈80ms under HTTP/JSON with a controlled backend; the gap vanishes once real compute dominates.
Does the real Linux `memfd`/`eventfd` path help?	Yes, substantially — ≈4× over UDP-RPC, ≈19× over HTTP — strictly on one machine.
Where does RoCE fit?	Another order of magnitude above all of the above (2.8µs); its control-plane setup is fragile under naive tooling, its data path was not, where tested.
Does any of this need the KV cache to cross the network?	No — see Section 3. That is the point.

no second RDMA-capable peer was available on the Wi-Fi network; and the Soft-RoCE loss test used two data points rather than a swept curve. A rigorous follow-up would add a second Linux Soft-RoCE peer to test RoCE across an actual Wi-Fi hop, real `tc netem`-shaped loss and jitter on a physical link, multiple independent runs per condition, and a finer-grained loss sweep to localize the RC transport's own breaking point, if one exists at all, separately from the benchmark tool's control-channel fragility. Real KV cache tensor transfer, for the disaggregated-serving and compute-handoff cases named in Section 3, is deliberately left as a different, future problem rather than an unfinished part of this one.

10 Conclusion

MemLLM's zero-copy argument does not, and cannot, extend past one machine — but its design philosophy does, in a wire protocol that measurably beats plain HTTP/JSON on a real Wi-Fi link and survives real packet loss without failing. On the machine where shared memory is actually available, this work supplies the real Linux measurement MemLLM always described, and locates it precisely against real RDMA: roughly four times behind real shared memory, and roughly thirty-six times behind Soft-RoCE, a ceiling this line of Python prototypes falls well short of. Most importantly, the architecture that makes any of this necessary at all turns out to be narrower than "KV cache over Wi-Fi" implies — the cache should stay exactly where it was computed, and only the smallest possible delta, plus a lightweight record of what changed, needs to travel at all.

References

[1] Sampatirao, H. P. (2026). *MemLLM: Zero-copy Shared Memory Interface for Unbounded Context Management in on-device LLM Inference.* Available at SSRN: https://ssrn.com/abstract=6845541 or http://dx.doi.org/10.2139/ssrn.6845541. Code: https://github.com/hari-sampatirao/memLLM

[2] Maroš, V. (2018). *memif: A Shared Memory Interface for Userspace Networking.* DPDK Summit.

[3] Kalia, A., Kaminsky, M., & Andersen, D. (2019). *Datacenter RPCs can be General and Fast.* NSDI.

[4] *WIP: When RDMA Meets Wireless.* (2022). WoWMoM.

[5] Kwon, W. et al. (2023). *Efficient Memory Management for LLM Serving with PagedAttention.* SOSP.

[6] Zheng, L. et al. (2024). *SGLang: Efficient Execution of Structured Language Model Programs.* arXiv:2312.07104.

[7] Xiao, G. et al. (2024). *Efficient Streaming LLMs with Attention Sinks.* ICLR.

[8] Mooncake Team (2025). *Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving.* USENIX FAST (Best Paper).

[9] Qwen Team (2024). *Qwen2.5 Technical Report.* Alibaba Group.

[10] Linux Kernel Documentation. *rdma_rxe — Soft-RoCE Transport Driver.* drivers/infiniband/sw/rxe.

[11] The Apache Software Foundation. *Apache License, Version 2.0*, §3 (Grant of Patent License).

[12] Cisco Technology, Inc. *RoCE over Wireless.* US Patent 10,725,948 B2, filed Oct. 2018, granted Jul. 2020.