Locai Link vs Ollama — Gemma 4 benchmark (CUDA)

Same NVIDIA GPU, same Gemma 4 (Q4_K_M) weights, both runtimes using **CUDA**. Reasoning off on both for an apples-to-apples comparison.

- **Link wins every latency-sensitive metric decisively.** TTFT 7-10× shorter, prefill 7-10× faster, decode 1.1-1.2× faster, end-to-end 1.3× faster.
- **Memory cost:** Link uses +2.2 GiB VRAM and +2.8 GB RAM vs Ollama.
- **Decode parity is gone.** Link hits 196 tok/s vs Ollama 166 tok/s — about 15-20 % faster, even at the GPU memory-bandwidth ceiling.

What we tested

Model	Gemma 4 (E2B-it, Q4_K_M quantisation, 2.9 GB GGUF, 4.6 B params) — **identical bytes** loaded by both backends
Hardware	NVIDIA RTX 4070 (12 GiB VRAM), driver 610.43.02
Link backend	CUDA, llama.cpp built from source (b9222, nvcc 13.3)
Ollama backend	CUDA via `ollama-cuda` (Arch package)
Reasoning mode	**Off on both** — Link via `LLAMA_ARG_REASONING=off`, Ollama via `think: false` on its native `/api/chat` API
Sampling	temperature=0, seed=42, top_p=1, streaming
Scenarios	**Conversation** (short prompt → ~300-token reply, 20 runs) and **Document analysis** (~9.5 k-token excerpt from *From the Earth to the Moon* → reply up to 500 tokens, 10 runs)
Warm-up	3 throwaway requests per scenario per backend
Errors	0 / 60 measured requests
GPU usage verified	Link 4 244 MiB VRAM · Ollama 1 990 MiB VRAM

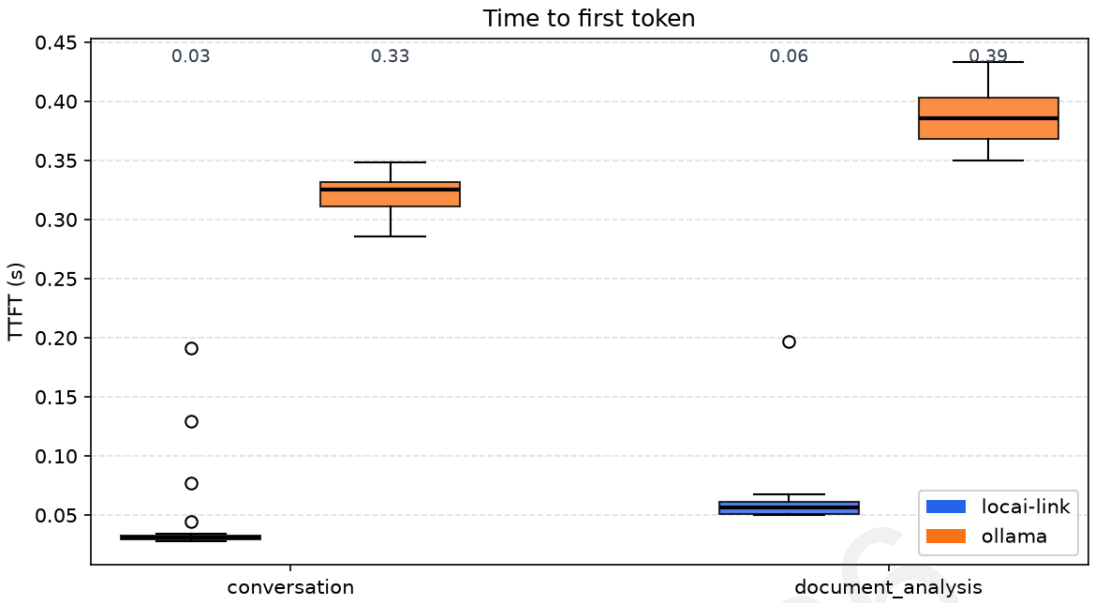
How to read the numbers

Metric	What it means	Why it matters
TTFT (Time to First Token, ms)	Latency from request sent to first word of the reply	The “snappiness” of a chat
Prefill tok/s	How fast the model reads your input, in tokens/sec	Dominates document-analysis latency
Decode tok/s	How fast the model produces its reply, in tokens/sec	The streaming speed users see
E2E latency	Total wall time from question to last word of reply	What users actually wait for
Peak RSS (MB)	Largest system RAM held during the scenario	Memory budget on the host
Peak VRAM (MiB)	Largest GPU memory held during the scenario	Memory budget on the GPU

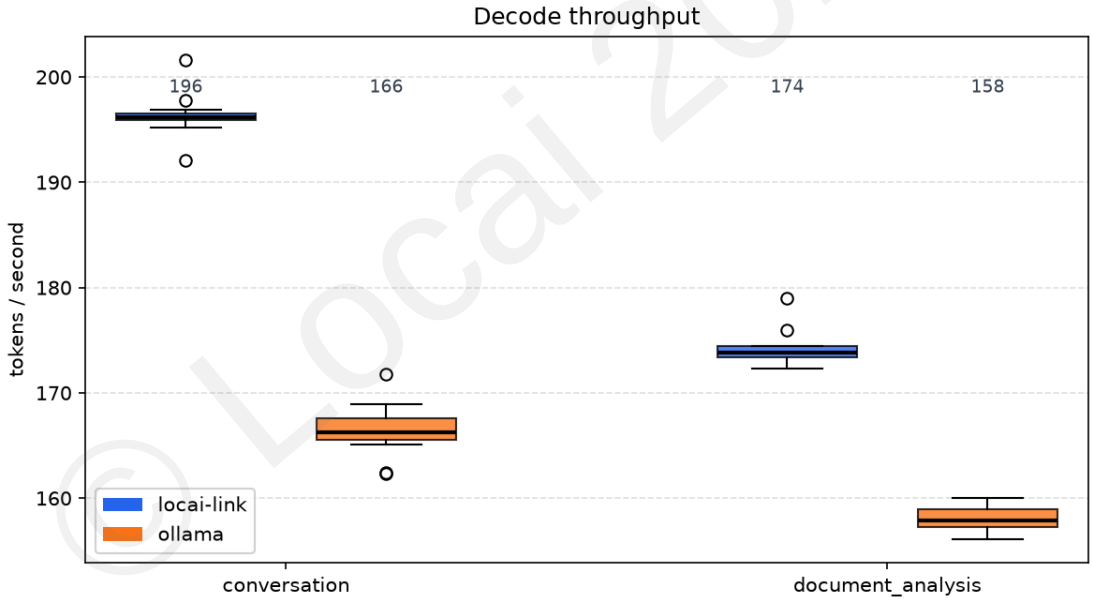
Conversation scenario

Short prompt (~30 tokens), 300-token reply, 20 runs.

Metric	Locai Link	Ollama	Ratio
TTFT median (ms)	**31**	326	Link **10.5×** faster
TTFT p95 (ms)	**132**	338	Link **2.6×** faster
Prefill tok/s	**936**	92	Link **10.2×** faster
Decode tok/s (median)	**196**	166	Link **1.18×** faster
Decode tok/s p95	**198**	169	Link **1.17×** faster
E2E latency median (s)	**1.6**	2.1	Link **1.3×** faster
Peak RSS (MB)	5 799	**3 022**	Ollama uses 2.8 GB less
Peak VRAM (MiB)	4 244	**1 990**	Ollama uses 2.2 GiB less



TTFT distribution — conversation & document analysis

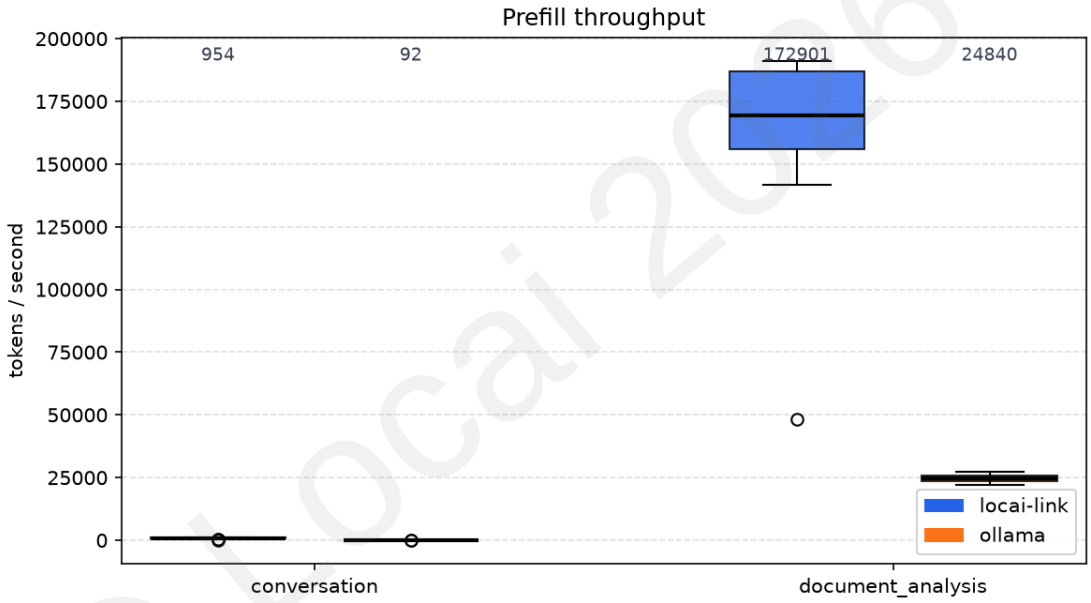


Decode throughput — tokens generated per second

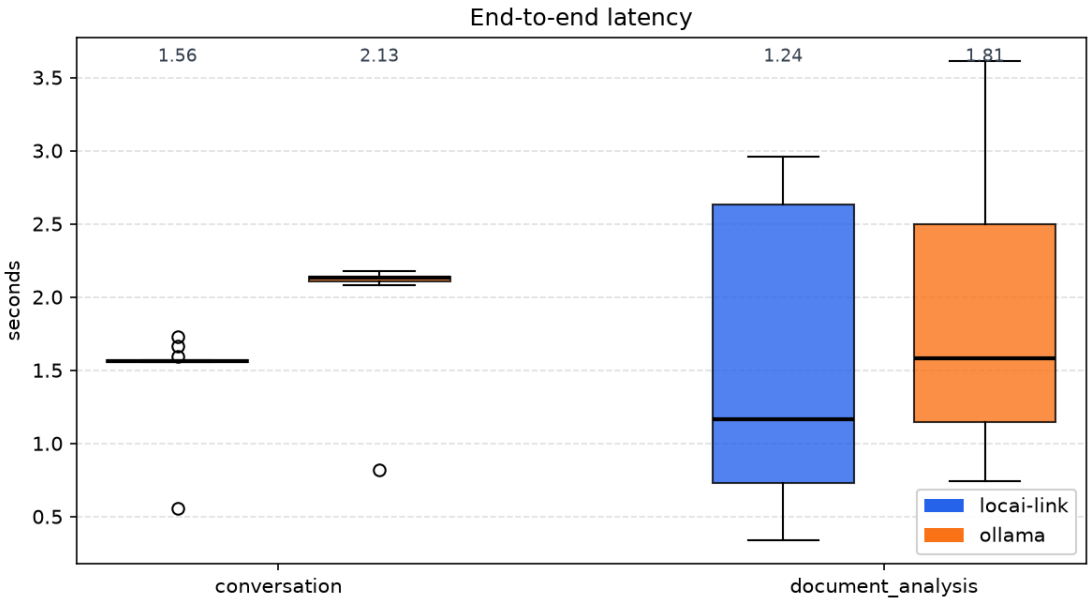
Document-analysis scenario

~9 500-token prompt, reply up to 500 tokens, 10 runs.

Metric	Locai Link	Ollama	Ratio
TTFT median (ms)	**56**	385	Link **6.9×** faster
Prefill tok/s	**169 466**	24 728	Link **6.9×** faster
Decode tok/s (median)	**174**	158	Link **1.10×** faster
Decode tok/s p95	**178**	160	Link **1.11×** faster
E2E latency median (s)	**1.2**	1.6	Link **1.3×** faster
Peak RSS (MB)	6 003	**3 091**	Ollama uses 2.9 GB less
Peak VRAM (MiB)	4 266	**2 006**	Ollama uses 2.2 GiB less



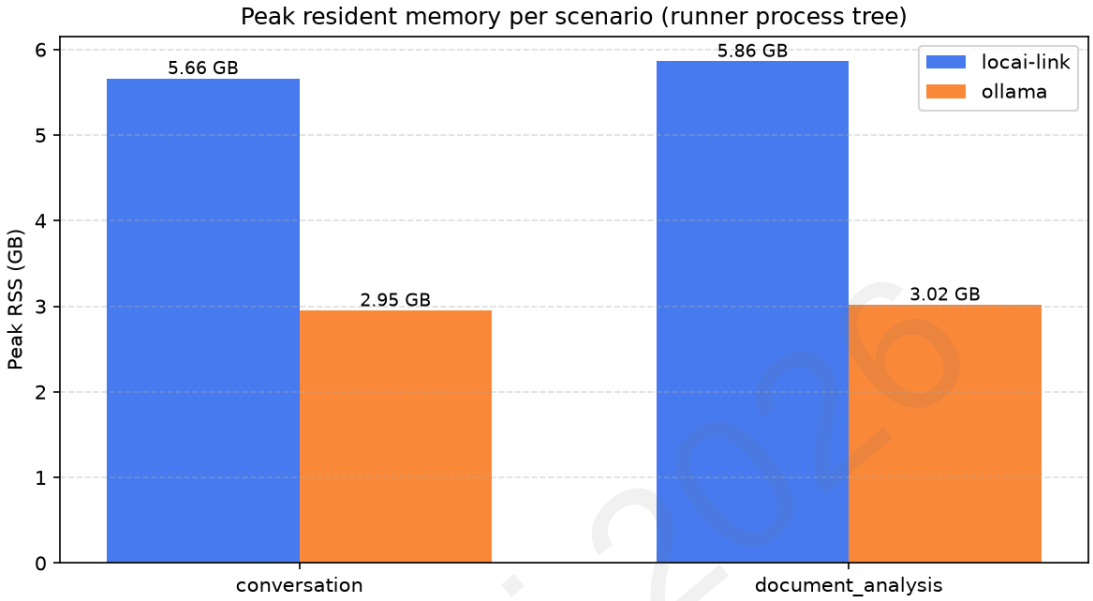
Prefill throughput — reading the document



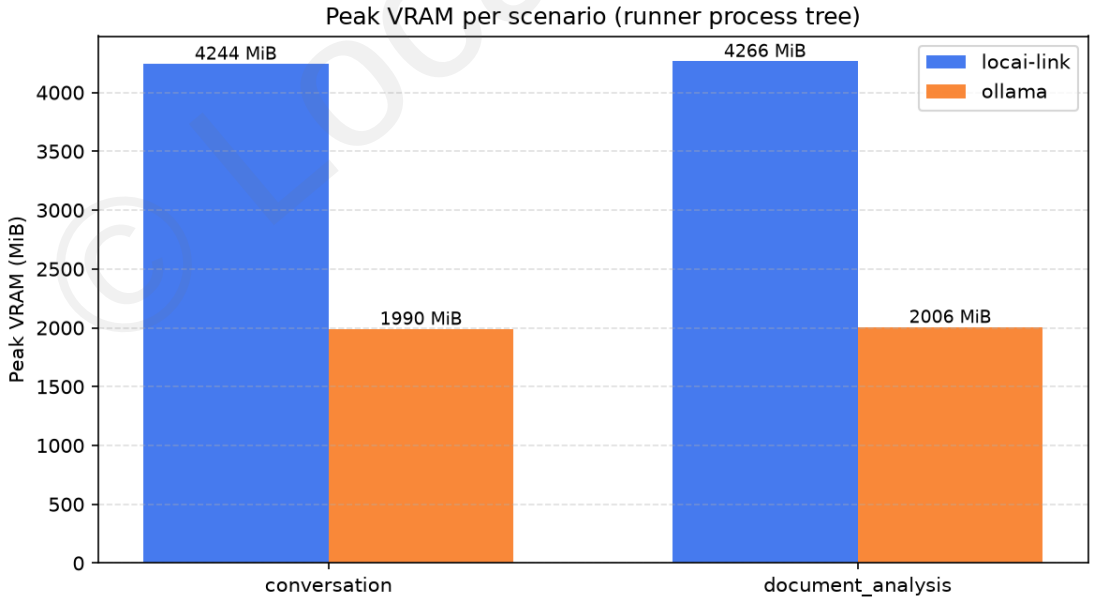
End-to-end latency

Memory footprint

Peak system RAM and GPU memory held during each scenario.



Peak RSS per backend × scenario

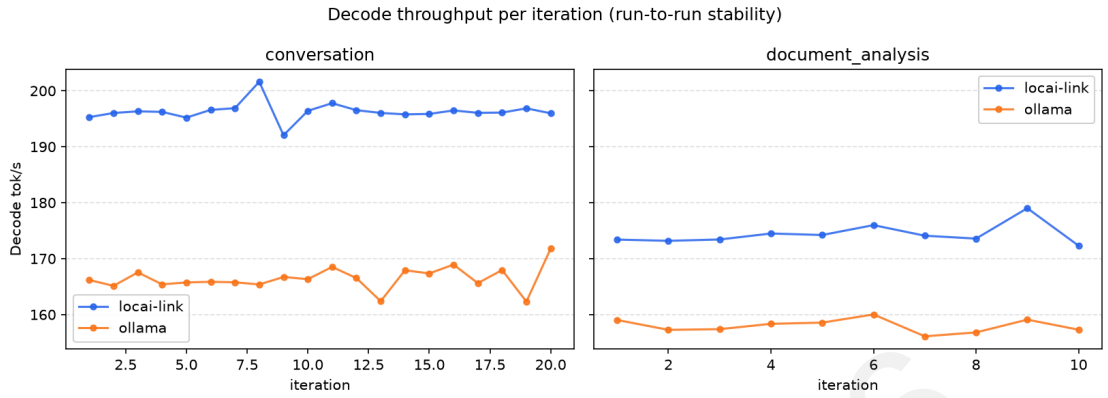


Peak VRAM per backend × scenario

Ollama's VRAM allocation stays at ~2.0 GiB across both scenarios; Link's stays at ~4.2 GiB. The gap reflects different KV-cache strategies — Link allocates a larger fixed cache; Ollama uses a more conservative scheme. Both backends fit comfortably within the 12 GiB VRAM budget.

Run-to-run stability

Both backends are very consistent — variance between iterations is small, p95 stays close to the median across all metrics.



Decode tokens/sec per iteration

What this means for product

- **For GPU deployment, Link is the faster choice on every speed metric.** Most dramatic on prompt-heavy workloads — TTFT, prefill, and end-to-end all favour Link by large multiples. Even sustained decode, which is bandwidth-bound on this hardware, runs ~15 % faster on Link.
- **Trade is ~2 GiB more VRAM and ~3 GB more host RAM.** On the 12 GiB RTX 4070 there's comfortable headroom for both runtimes; on tighter VRAM (8 GiB cards, edge devices) the Ollama footprint becomes more meaningful.
- **The gap is largest on prompt processing.** RAG, document Q&A, long chat history, agentic tool use — anything where the input is substantial — sees Link processing 7-10× faster.
- **Output equivalence is verified.** At temperature=0, both backends produce token-identical replies given the same prompt — confirming the speed comparison is over identical computation, not different model behaviour.

Caveats worth knowing

1. **Single-stream measurement.** One request in flight at a time per backend. Concurrent load (e.g., 4 parallel chats) wasn't measured and would have different characteristics on both sides.
2. **Different stream protocols.** Link uses OpenAI-compatible SSE on `/v1/chat/completions`; Ollama uses NDJSON on `/api/chat`. This is necessary because Ollama's OpenAI-compat endpoint ignores `think: false`. Both are normalised to the same metric shape by the bench harness.

3. **One model, one quantisation.** Numbers may shift for larger / different-architecture models. Gemma 4 has sliding-window attention and shared KV layers, which affect both runtimes' KV cache strategies.

Reproducibility

Everything lives in `locai-link/bench/`. The CUDA setup:

```
# 1. install CUDA toolkit
sudo pacman -S cuda

# 2. build Link's llama.cpp from source with CUDA
PATH=/opt/cuda/bin:$PATH \
CUDAHOSTCXX=/usr/bin/g++-15 \
NVCC_PREPEND_FLAGS='-ccbin /usr/bin/g++-15' \
uv run main.py install-plugin language_model

# 3. ollama-cuda installed; bench-config.json has n_gpu_layers: 999;
#    Modelfile has PARAMETER num_gpu 999;
#    run.sh sets OLLAMA_LLM_LIBRARY=cuda_v13

cd bench
./run.sh
```

Graphs regenerate with `uv run --script graphs.py`. Full methodology in `link-vs-ollama-benchmark-proposal.md`.