# Running Gemma 4 26B on a 16 GB AMD GPU

**40 tok/s at 64K context — 262K context validated — a build report and a practitioner read**

*Ver07 — 09/May/2026 — Des Donnelly — https://www.dd.ie/*

---

*Researched, built, written, and validated entirely on the hardware described. The first draft was generated by Gemma 4 26B A4B running locally; subsequent revisions worked from the configuration data captured in the live server logs. The mistakes corrected along the way are noted in the document. Hardware bought at consumer prices, work done from Co Tyrone in North Ireland, with no sponsorship and no early access. Good luck building and kudos to AMD!*

---

## 1. Introduction

Google announced last week that Gemma 4 has been downloaded over 60 million times since release. A small fraction of those downloads have been turned into running systems. A smaller fraction again on AMD hardware. A smaller fraction still on a 16 GB consumer card. This document is for that small fraction — and for everyone else who is wondering whether it is possible.

It is. The numbers are real, the build is reproducible, and what follows is the working configuration, the traps that cost me time, and an honest assessment of the limits of the resulting system.

A note on perspective: I have been a paying user of frontier cloud LLMs since 2024 and use them daily. That calibration shapes how I read what local hardware delivers, and I will name where that lens matters. The aim is a piece useful to AMD builders, with claims that match what the hardware actually does.

---

## 2. The Headline Numbers

Three configurations were tested and validated on this hardware. All three load successfully. Each has its own trade-off and use case.

| Configuration | Decode | Use case | Sample |
|---|---|---|---|
| 64K, parallel=4, n-cpu-moe=4 | ~40 tok/s | Daily production, multi-slot | n=21 averaged |

| Configuration | Decode | Use case | Sample |
|---|---|---|---|
| 131K, parallel=4, n-cpu-moe=4 | ~37 tok/s | Long-context daily use | n=1 (1,302 tokens) |
| 262K, parallel=1, n-cpu-moe=8 | 26.74 tok/s | Full native context, single slot | n=1 (1,712 tokens) |

The 262K figure is the model's native training context (`n_ctx_train = 262144`), validated on 16 GB AMD consumer hardware.

**Verified environment:**

| Metric | Value |
|---|---|
| Model on disk | 12.35 GiB at 4.21 BPW (IQ3_S mix) |
| llama.cpp build | b9031-bf76ac77b |
| ROCm | 7.2.1 |
| Hardware | Sapphire Nitro+ RX 9060 XT 16 GB (gfx1200) |
| Prefill (warm cache) | 600–1,100+ tok/s |
| Prefill (cold cache) | 60–700 tok/s, scales with library warmup |

> *64K @ 40 tok/s for daily use. 262K @ 26.74 tok/s for full native context. The number that matters depends on the task; both are reproducible from the launch commands documented below.*

---

## 3. The Components That Affect Performance

Only the parts that materially affect Gemma's speed and behaviour:

| Component | Detail | Why it matters |
|---|---|---|
| GPU | Sapphire Nitro+ AMD Radeon RX 9060 XT 16 GB GDDR6 (gfx1200, RDNA 4) | Primary inference compute and KV cache |
| CPU | AMD Ryzen 7 7700 — 8c/16t, Zen 4 | MoE expert offload via `--n-cpu-moe`; prefill on long prompts |
| RAM | 32 GB DDR5-6000 (2×16 GB) | CPU-side expert layers; bandwidth matters for offload speed |
| Storage | NVMe PCIe Gen4 (Crucial T500 1TB) | Model load time only (one-time on server start) |
| OS | Ubuntu 24.04.4 LTS, kernel 6.17.0-23-generic | ROCm 7.x compatibility; in-tree RDNA 4 drivers |

### The iGPU Detail

On AM5 hardware, ROCm enumerates two devices: the discrete RX 9060 XT (Device 0, gfx1200) and the integrated GPU on the Ryzen die (Device 1, gfx1036). Without an explicit `HIP_VISIBLE_DEVICES=0` pin, llama.cpp can spread layers across both, dragging decode speed from ~40 tok/s to single digits. This is the most important single environment variable in the whole build.

---

## 4. The Model: Gemma 4 26B A4B

Architectural facts confirmed from the GGUF metadata:

| Attribute | Value |
|---|---|
| Total parameters | 25,233,142,046 ('26B' nominal) |
| Active parameters per token | ~3.8B |
| Architecture | Mixture-of-Experts (MoE) |
| Total experts | 128 |
| Active experts per token | 8 |
| Transformer layers | 30 |
| Attention pattern | Alternating standard / Sliding Window Attention |
| SWA window | 1,024 tokens |
| Native max context | 262,144 tokens (256K theoretical) |
| Licence | Apache 2.0 |

### Why This Architecture Fits 16 GB

Three architectural facts work together. Any one alone would not be enough; together they make the build viable:

**Mixture of Experts (MoE) reduces active compute.** Of 128 experts in the model, 8 fire per token. Of 25.2B total parameters, ~3.8B are active at any moment. The GPU moves roughly 15% of the model's weight per inference step. This places the practical reasoning capacity in the range of a well-trained 4–8B dense model, rather than a true 26B dense one.

**Sliding Window Attention (SWA) caps KV cache growth.** In a standard transformer, KV cache scales linearly with context length. With SWA, alternating layers attend to a fixed window of 1,024 tokens regardless of total context. The model metadata confirms an alternating pattern across the 30 layers. The practical result, measured on this build:

| Context | KV cache size | Δ per 2× context |
|---|---|---|
| 64K | 1,158 MiB | — |
| 131K | 1,838 MiB | +680 MiB (1.6×) |

| Context | KV cache size | Δ per 2× context |
|---|---|---|
| 262K | 2,879 MiB | +1,041 MiB (1.6×) |

Each doubling of context costs roughly 1.6× the KV cache, not 2×. A standard transformer would have shown 2× scaling at every step. This is the architecture's contribution made visible.

**IQ3_M (IQ3_S mix) hits a usable size point.** The bartowski quantisation lands at 12.35 GiB at 4.21 bits-per-weight. The naming is slightly misleading — the file is labelled IQ3_M, but llama.cpp reports the actual format as "IQ3_S mix", with a blend of q5_0, q5_1, q8_0, q5_K, q6_K, iq4_nl, and iq3_s tensors. The mix preserves higher precision where it matters most while compressing aggressively elsewhere.

---

## 5. The Working Configuration (64K Production)

The launch command that produced the 40 tok/s headline figure:

```
HIP_VISIBLE_DEVICES=0 ~/llama.cpp/build-hip/bin/llama-server \
  -m ~/models/gemma-4-26b-a4b-it/*IQ3_M*.gguf \
  --host 127.0.0.1 --port 8090 \
  --n-gpu-layers 99 \
  --ctx-size 65536 \
  --flash-attn on \
  --cache-type-k q8_0 \
  --cache-type-v q8_0 \
  --n-cpu-moe 4 \
  --jinja
```

### Flag Breakdown

| Flag | Purpose |
|---|---|
| `HIP_VISIBLE_DEVICES=0` | Pins HIP to the discrete GPU only. Single most important variable on AM5. |
| `--n-gpu-layers 99` | Push every layer to GPU. '99' means 'all of them'. |
| `--ctx-size 65536` | 64K context — the daily-use balance of speed and memory. |
| `--flash-attn on` | Flash Attention. Required for the Q8_0 V-cache to allocate correctly. Modest decode improvement, larger prefill improvement. |
| `--cache-type-k q8_0` | 8-bit KV cache for keys. Halves cache memory vs FP16 default with negligible quality loss. |
| `--cache-type-v q8_0` | 8-bit KV cache for values. Same effect on the V side. Requires `--flash-attn`. |
| `--n-cpu-moe 4` | Offloads 4 expert layers to the Ryzen 7700, freeing GPU VRAM. MoE routing tolerates this well — only the active experts run |

| Flag | Purpose |
|---|---|
| | anyway. |
| `--jinja` | Enables Jinja2 chat template rendering. Auto-detects Gemma 4's peg-gemma4 template. |

### What Is Not in the Command

**No GRUB kernel parameters.** Some guides recommend `amdgpu.gttsize=16384` or similar for VRAM 'spillover' into system memory. This build runs at default kernel parameters — `GRUB_CMDLINE_LINUX=""` — and never spills. The architectural choices above keep the working set inside VRAM.

**No batch-size override.** Default `n_batch=2048` and `n_ubatch=512` are appropriate for this hardware. Earlier guidance that recommended `--batch-size 256` came from a pre-flash-attn era and is no longer needed.

### Memory Breakdown at 64K

From the live server log at startup:

```
Model weights on GPU:    11,092 MiB
KV cache (Q8_0):          1,158 MiB  (680 non-SWA + 478 SWA)
Compute buffers:            532 MiB
------------------------  ------
Total GPU memory:        12,782 MiB
Free GPU memory:          2,801 MiB
CPU offload (4 experts):  2,284 MiB
```

---

## 6. Two Larger Configurations

The same launch command, with selected flags adjusted, gives two further valid configurations.

### 131K (Long-Context Daily Use)

Change `--ctx-size 65536` to `--ctx-size 131072`. Everything else stays the same.

```
HIP_VISIBLE_DEVICES=0 ~/llama.cpp/build-hip/bin/llama-server \
  -m ~/models/gemma-4-26b-a4b-it/*IQ3_M*.gguf \
  --host 127.0.0.1 --port 8090 \
  --n-gpu-layers 99 \
  --ctx-size 131072 \
  --flash-attn on \
  --cache-type-k q8_0 --cache-type-v q8_0 \
  --n-cpu-moe 4 \
  --jinja
```

**Memory at 131K:** 13,462 MiB GPU (model 11,092 + KV 1,838 + compute 532), 2,121 MiB free, 2,412 MiB host offload. Decode speed ~37 tok/s, sampled.

### 262K (Full Native Context)

Two changes from the 131K command: `--ctx-size 262144` and `--parallel 1`. Also `--n-cpu-moe 8` to free additional GPU memory for the larger compute buffers and KV cache that 262K requires.

```
HIP_VISIBLE_DEVICES=0 ~/llama.cpp/build-hip/bin/llama-server \
  -m ~/models/gemma-4-26b-a4b-it/*IQ3_M*.gguf \
  --host 127.0.0.1 --port 8090 \
  --n-gpu-layers 99 \
  --ctx-size 262144 \
  --parallel 1 \
  --flash-attn on \
  --cache-type-k q8_0 --cache-type-v q8_0 \
  --n-cpu-moe 8 \
  --jinja
```

**Memory at 262K:** 13,443 MiB GPU (model 9,579 + KV 2,879 + compute 984), 2,126 MiB free, 4,175 MiB host offload. Decode speed 26.74 tok/s, measured over a 1,712-token output.

The trade-off: `--parallel 1` means a single conversational slot. Without it, llama.cpp silently rewrites the requested 262K context down to 512 tokens (see Hard-Won Lessons). For personal use — one user, one conversation at a time — single-slot is the natural configuration anyway. For multi-user serving, the practical ceiling on this hardware is 131K.

A second observation worth recording: compute buffers are not flat across context sizes. At 64K and 131K they hold at 532 MiB; at 262K they grow to 984 MiB. Flash Attention's working space scales gently with context, just not at the rate the KV cache does.

---

## 7. ROCm Setup

A short version of what worked, with the points that caused friction:

```
# 1. Download the AMD installer (pin exact version)
wget https://repo.radeon.com/amdgpu-install/7.2.1/ubuntu/noble/amdgpu-
install_7.2.1.70201-1_all.deb

# 2. Install the installer package
sudo apt install -y ./amdgpu-install_7.2.1.70201-1_all.deb

# 3. Install ROCm — graphics + rocm, no DKMS, no 32-bit
sudo amdgpu-install -y --usecase=graphics,rocm --no-dkms --no-32
```

```
# 4. Add yourself to render and video groups
sudo usermod -a -G render,video $USER

# 5. Reboot, then verify
rocminfo | grep -E "Marketing Name|gfx1200"
```

> *Note: ROCm 7.2.2 attempted first — the graphics/7.2.2 sub-repository returned HTTP 404 during apt update. Downgrade to 7.2.1 (AMD's officially supported release for Ubuntu 24.04 Noble) resolved this. Worth verifying repo URLs against repo.radeon.com before following any guide.*
>
> *Note: On kernel 6.17, the RDNA 4 drivers are in-tree. Installing rocm-dkms creates parallel kernel modules that can conflict on a kernel update. Use `--no-dkms`.*
>
> *Note: dmesg requires sudo on Ubuntu 24.04 (`kernel.dmesg_restrict=1` is set by default). Older guides that run dmesg without sudo will silently return nothing.*

### What rocminfo Should Show

After install, both GPUs should be enumerated:

```
Marketing Name:          AMD Radeon RX 9060 XT
  Name:                    gfx1200
Marketing Name:          AMD Radeon Graphics       (the iGPU)
  Name:                    gfx1036
```

Both visible is correct. The `HIP_VISIBLE_DEVICES=0` pin in your launch command keeps the iGPU out of the inference path.

---

## 8. Building llama.cpp

Build the HIP backend from source, targeting gfx1200. The repo has moved — use the new URL:

```
# Clone (note: ggml-org, not ggerganov)
git clone https://github.com/ggml-org/llama.cpp ~/llama.cpp
cd ~/llama.cpp && git rev-parse HEAD

# Confirm the cmake flag name in your checkout
grep -E "GGML_HIP\b|GGML_HIPBLAS\b" ~/llama.cpp/CMakeLists.txt

# Build
export PATH=/opt/rocm/bin:$PATH
export CMAKE_PREFIX_PATH=/opt/rocm
cmake -S ~/llama.cpp -B ~/llama.cpp/build-hip \
  -DGGML_HIP=ON \
  -DAMDGPU_TARGETS=gfx1200 \
```

```
  -DGGML_HIP_NO_VMM=1 \
  -DCMAKE_BUILD_TYPE=Release
cmake --build ~/llama.cpp/build-hip --config Release -j$(nproc)
```

*Note: Current llama.cpp uses `-DGGML_HIP=ON`. Older guides use `-DGGML_HIPBLAS=ON`. The grep step above tells you which applies to your checkout. The wrong flag produces a CPU-only binary that reports no HIP devices.*

## Vulkan Fallback Build

Worth building after the HIP build succeeds. Provides a working alternative if HIP fails on first run.

```
# Install full SPIR-V toolchain — spirv-tools alone is insufficient
sudo apt -y install libvulkan-dev glslc spirv-tools spirv-headers vulkan-
tools

cmake -S ~/llama.cpp -B ~/llama.cpp/build-vulkan \
  -DGGML_VULKAN=ON -DCMAKE_BUILD_TYPE=Release
cmake --build ~/llama.cpp/build-vulkan --config Release -j$(nproc)
```

*Note: The Vulkan build will fail at ~47% with: 'spv' is not a class, namespace, or enumeration — if spirv-headers is not installed. The spirv-tools package does not pull it in as a dependency.*

## Downloading the Model

Accept the Gemma 4 licence at huggingface.co/google/gemma-4-26b-a4b-it (Google form, ~2 minutes), generate a HF token, then:

```
# Set up uv-managed Python venv
uv venv --python 3.12 ~/venvs/ai && source ~/venvs/ai/bin/activate
uv pip install -U "huggingface_hub" hf_xet

# Login (use --token flag; masked-input paste drops tokens silently)
hf auth login --token "hf_PASTE_YOUR_TOKEN_HERE"

# Download IQ3_M quant (~12.35 GiB)
hf download bartowski/google_gemma-4-26B-A4B-it-GGUF \
  --include "*IQ3_M*.gguf" \
  --local-dir ~/models/gemma-4-26b-a4b-it
```

*Note: huggingface-cli is deprecated in HF Hub 1.x. Use `hf` instead.*

---

## 9. Running the Server

Wrap the launch in a tmux session so it survives terminal close:

```
sudo apt -y install tmux  # not pre-installed on minimal Ubuntu 24.04

tmux new-session -d -s llama "HIP_VISIBLE_DEVICES=0 \
  ~/llama.cpp/build-hip/bin/llama-server \
  -m ~/models/gemma-4-26b-a4b-it/*IQ3_M*.gguf \
  --host 127.0.0.1 --port 8090 \
  --n-gpu-layers 99 \
  --ctx-size 65536 \
  --flash-attn on \
  --cache-type-k q8_0 --cache-type-v q8_0 \
  --n-cpu-moe 4 --jinja \
  2>&1 | tee ~/logs/llama-server-\$(date +%F-%H%M).log"
```

### Verify the Server is Ready

```
until curl -s http://127.0.0.1:8090/v1/models | grep -q '"id"'; do
  echo "waiting..."; sleep 2
done && echo "READY"
```

OpenAI-compatible API at `http://127.0.0.1:8090/v1`, plus a built-in web chat UI at `http://127.0.0.1:8090`. Both work out of the box.

---

## 10. Model Behaviour

**Reasoning mode is on by default.** Gemma 4 outputs a `<think>` block before its final answer. The `reasoning_content` field contains the internal scratchpad; `content` contains the user-facing response. Prefix prompts with `/no_think` for direct, fast answers.

**finish_reason: "length" instead of "stop".** Reasoning mode consumes tokens before the final answer. Use `max_tokens ≥ 500` for short answers, `≥ 2000` for code generation. At `max_tokens: 40`, the model exhausts its budget on internal reasoning and returns empty content.

**Conversation truncated mid-response.** The active context limit has been reached. Start a new conversation client-side; no server restart needed.

**Training cutoff:** approximately January 2025. Release: 02/04/2026.

---

## 11. Connecting an IDE: Status as of 09/05/2026

This section is unfinished, deliberately. The state of investigation:

### Cursor

Cursor 3.2.16 has a current limitation in which custom local model names do not survive the request pipeline. The Override Base URL setting appears to work in the UI but the model identifier does not route to a locally-served endpoint. Cursor's primary focus is cloud inference, so a fix may not be near-term.

### Antigravity

Google Antigravity allows custom model configuration via `.antigravity/config.json`, pointing at a local OpenAI-compatible endpoint. The configuration is straightforward (Masaki Hirokawa's antigravitylab.net articles cover it well). The constraint is performance.

Antigravity is built around an Architect/Builder agent pattern. Every request triggers multiple sequential LLM calls — planning, decomposition, implementation, verification. That works comfortably at cloud speeds (200+ tok/s). At 40 tok/s with prefill on each step, a single 'explain this function' turn takes 30–60 seconds. The slowness reflects the architectural fit between the agent paradigm and single-card local hardware, not a configuration error.

### VS Code + Continue.dev

Continue.dev is open-source, MIT-licensed, and architected for single-call chat against a custom endpoint. No agent orchestration overhead, designed local-first. It fits the performance profile of this hardware.

I have not yet tested it on this build. The next attempt to close the IDE question will start there. If it works, the integration is solved in 20 minutes. If it does not, that is a useful data point too.

> *Honest status: the model is running, the API is live, the IDE front-end remains an open problem. Builders who need a working IDE today should plan to test their preferred client end-to-end against their own hardware before relying on it.*

---

## 12. Hard-Won Lessons

Things worth knowing before you start, drawn from where time was lost:

| Issue | What works |
|---|---|
| ROCm 7.2.2 graphics repo 404 | `graphics/7.2.2` sub-repository returned HTTP 404 during apt update. Use ROCm 7.2.1, AMD's officially supported release for Ubuntu 24.04 Noble. |
| amdgpu-install –usecase omissions | Default usecase set is `dkms,graphics,opencl,hip` — `rocm` is not in the default. Specify `--usecase=graphics,rocm` explicitly. Without `rocm`, you get HIP and OpenCL but not rocBLAS/hipBLAS, which llama.cpp's HIP build requires. |
| ROCm + DKMS on kernel 6.17 | Kernel 6.17 includes RDNA 4 drivers in-tree. DKMS adds parallel kernel modules that can conflict on a kernel update. Use `--no-dkms`. |
| cmake flag name drift | Current llama.cpp uses `-DGGML_HIP=ON`. Older guides use `-DGGML_HIPBLAS=ON`. Probe `CMakeLists.txt` to confirm. Wrong flag = CPU-only binary, no HIP devices found. |
| spirv-headers missing | Vulkan build fails at ~47% with `'spv' is not a class, namespace, or enumeration`. spirv-tools does not pull spirv-headers as a dependency. Install explicitly. |
| iGPU enumeration on AM5 | `HIP_VISIBLE_DEVICES=0` is essential. ROCm enumerates the Raphael iGPU (Device 1) alongside the dGPU (Device 0). Without the pin, decode speed drops to single digits. |
| dmesg restricted on Ubuntu 24.04 | `kernel.dmesg_restrict=1` by default. Running dmesg without sudo silently returns nothing. Use: `sudo dmesg \| grep -iE "amdgpu\|error\|warning"` |
| tmux not pre-installed | Ubuntu 24.04 minimal does not include tmux. Run: `sudo apt -y install tmux` — before attempting the server launch. |
| huggingface-cli deprecated | Use `hf` in HF Hub 1.x. `hf auth login --token "..."` — use the `--token` flag; masked-input paste silently drops tokens. |
| VRAM ceiling at Q4_K_M | Q4_K_M for a 26B MoE pushes VRAM use over the 16 GB ceiling. The model spills to system RAM, dropping decode from 40 tok/s to 2 tok/s. IQ3_M is the right choice for 16 GB; Q4_K_M is for 24 GB+ cards. |
| KV cache default is FP16 | Default `--cache-type-k` is FP16. Setting `q8_0` halves KV cache memory at negligible quality cost — the difference between fitting at 64K context and not. |
| Q8_0 V-cache requires Flash Attention | `--cache-type-v q8_0` will not allocate without `--flash-attn on`. The flag combination is interdependent. |
| Q4_0 KV cache silently fails on HIP | `--cache-type-k q4_0` and `--cache-type-v q4_0` produce no error but the context loads at 512 tokens regardless of `--ctx-size`. The HIP backend in llama.cpp commit b9031-bf76ac77b does not appear to |

| Issue | What works |
|---|---|
| | support Q4_0 KV cache. Q8_0 is the lowest reliably-supported KV cache quantisation on AMD as of testing. |
| 262K context with –parallel 4 silently fails | Requesting `--ctx-size 262144` with the default 4 parallel slots causes llama.cpp to load with `n_ctx = 512` while reporting "no changes needed" in `params_fit`. Memory headroom is not the issue — there are 4–5 GB unused VRAM in the failed attempts. The fix is `--parallel 1`. The full native context loads correctly with single-slot allocation. This appears specific to the parallel-slot logic at maximum context. |
| Compute buffers scale gently with context | Compute buffers hold at 532 MiB at 64K and 131K, then grow to 984 MiB at 262K. Flash Attention's working space is not flat across context sizes. Worth budgeting for at maximum context. |
| Two identical NVMe drives at install | The Ubuntu installer shows model + size + device path. With two `CT1000T500SSD8` at 931.5G the fields are visually identical, and device names (`nvme0n1` vs `nvme1n1`) can swap between sessions. The installer GUI does not show serials. Recommended: physically disconnect drives you want to preserve before running an install on a multi-drive system. Five extra minutes of cable work; saves an evening of recovery. |
| IDE compatibility varies | Cursor 3.2.16 does not currently route to local custom models. Antigravity works but is slow on single-card local hardware due to agent paradigm. Continue.dev is the next test candidate. |

## 13. Practitioner Caveats

A section on what local LLM at this hardware tier delivers in practice, written by someone who pays for frontier cloud daily and uses both.

### What This System Does Well

Gemma 4 26B A4B at IQ3_M, with 8 active experts of 128 totalling ~3.8B active parameters, is competent in its weight class. It writes a function. It summarises a paragraph. It handles a well-defined classification task. It runs offline, with no API costs, no rate limits, and no data leaving the hardware. For privacy-bound work, narrow tasks, and unmetered iteration it is genuinely useful. None of that is small.

### Where Its Limits Show

The active-parameter capacity is in the range of a small dense model, and the effective reasoning depth follows from that. Sustained coherence over long outputs, holding three or four constraints in working memory simultaneously, the kind of inference that frontier cloud models do without thinking — these all show their limits within ten minutes of real use. The Hirokawa article on antigravitylab.net flags this directly: Gemma needs prompts

that are dramatically more explicit than what works with frontier models. Users coming from Claude or GPT will need to recalibrate their prompting style.

## Three Common Claims, with Context

**"Privacy."** Genuinely material for regulated work, confidential client material, or anything that should not pass through a third-party API. Less material for the chatting-about-code, drafting-emails workload that makes up most cloud LLM use under enterprise terms. Worth being clear which category your use sits in.

**"No fees."** Operationally true once built; the upfront hardware cost (£1,500+), electricity under load, build and maintenance time, and the opportunity cost of using a less capable tool are all real. £20/month for Claude Pro is in the same financial bracket as a year of electricity for the GPU.

**"Frontier-class at home."** Benchmark scores for open models are often within a few points of frontier closed models on standard tests. Real-world capability gaps are wider than the benchmarks suggest. The gap is most visible in sustained reasoning work, less visible in short-form generation.

> *You can run a 26B MoE at 40 tok/s on a 16 GB consumer card. You can also reach the model's full native 262K context on the same hardware. Both are hard-won and worth documenting. The model is competent within its weight class. It is not a substitute for frontier cloud reasoning, and that distinction is worth holding in mind when reading any local-LLM literature, including this one.*

## Where Local LLM Fits

In a portfolio that includes a frontier cloud subscription, local LLM keeps a real chair. It handles work that should not go through a cloud API. It is always available, regardless of internet, rate limits, or service status. It supports unmetered experimentation — trying twenty prompt variations to refine a workflow, learning the dynamics of an open model, building skills against an endpoint with predictable latency. These are real capabilities that did not exist on hardware this affordable two years ago.

Used for what it is rather than what the broader literature sometimes implies, this build is genuinely useful. The point of writing it up is to give other AMD builders the same baseline.

---

## 14. Summary

| Component | Specification | Role |
|---|---|---|
| GPU | Sapphire Nitro+ RX 9060 XT 16 GB (gfx1200) | Primary compute |
| CPU | Ryzen 7 7700 (Zen 4, 8c/16t) | MoE expert offload |
| RAM | 32 GB DDR5-6000 | CPU-side experts; bandwidth |
| Quant | IQ3_M / IQ3_S mix, 12.35 GiB | Fits 16 GB VRAM |
| KV cache | Q8_0 (8-bit) | Halves cache memory; lowest reliable type on HIP |
| Backend | ROCm 7.2.1 + llama.cpp HIP (b9031-bf76ac77b) | AMD-native inference |
| Context | 64K production / 131K long / 262K full native | Three validated tiers |
| Decode | 40 / 37 / 26.74 tok/s | At each tier respectively |
| Prefill | 600–1,100+ tok/s warm | Fast prompt ingestion |
| IDE | Open question | Cursor: limited. AG: slow on this tier. Continue: untested. |

**For AMD builders:** Pin your devices. Quantise your KV cache to Q8_0 with `--flash-attn on`. Build from source for gfx1200. Use `--no-dkms`. Install spirv-headers before the Vulkan build. Use `--parallel 1` if you want full 262K context. And if you ever expand to a second NVMe, physically disconnect the first drive before running the installer.

The hardware is ready. The software is catching up. What works, works — and the parts that are still in motion are honestly named here so other builders can plan around them.

---

## 15. The Twilight Zone

Two recent announcements from Google point at a near-future where the numbers in this article look conservative. Worth flagging both, while being clear that nothing in this section has been measured on this hardware — only mathematically projected from what others have published.

### Multi-Token Prediction Drafters (Released 05/05/2026)

Google's MTP drafters for the Gemma 4 family use speculative decoding with KV-cache sharing between target and drafter models. The announcement headline cites up to 3× speedup; the actual benchmark chart in the same blog post tells a more nuanced story. The published table covers seven hardware/model combinations:

| Hardware / Model | Speedup |
|---|---|

| Hardware / Model | Speedup |
|---|---|
| Gemma 4 26B / Nvidia A100 | 1.5× |
| Gemma 4 E2B / Samsung S26 | 1.8× |
| Gemma 4 E4B / Samsung S26 | 2.2× |
| Gemma 4 31B / Apple M4 | 2.5× |
| Gemma 4 E2B / Pixel TPU | 2.8× |
| Gemma 4 31B / Nvidia A100 | 3.0× |
| Gemma 4 E4B / Pixel TPU | 3.1× |

Mean: 2.41×. Median: 2.5×. Range: 1.5–3.1×. Coefficient of variation: ~24%. That spread is well above measurement noise and tells a real story.

**Why the variance is wide:** speculative decoding only helps when there is idle compute capacity to spare during single-token decode. The standard inference loop on most hardware is memory-bandwidth-bound rather than compute-bound — the GPU spends most of each token's time fetching weights from VRAM, while the compute units sit partly idle. MTP captures that idle compute by speculating ahead and verifying in parallel. The speedup is therefore proportional to how much idle compute headroom exists during single-token decode.

That depends on three things:

- **Memory bandwidth relative to compute throughput.** A100 has ~2 TB/s and ~78 TFLOPS FP16. Mobile hardware is an order of magnitude lower on both. Bandwidth-to-compute ratios vary substantially across this hardware list.
- **Model size relative to that ratio.** Bigger models create more memory pressure per token. The doubling from 1.5× (26B on A100) to 3.0× (31B on A100) on the same hardware shows the larger model exposing more idle compute that MTP can use.
- **Architecture-specific scheduling.** TPUs are designed around batched parallel computation, which is exactly what MTP's verification step does. Pixel TPU shows the highest speedups because the workload pattern aligns with the architecture's strengths.

**Where the RX 9060 XT lands:** ~456 GB/s memory bandwidth, ~26 TFLOPS FP16. Bandwidth-to-compute ratio: ~17.5 GB/TFLOP — slightly more compute-bound than A100 (~26 GB/TFLOP) but still memory-pressured at this model size. IQ3_M quantisation reduces bandwidth pressure further, which cuts both ways. A reasonable projection range is 1.5–2.0×, with 2.5× as the optimistic ceiling.

## TurboQuant (Released 24/03/2026)

Google Research's TurboQuant compresses KV cache to 3 bits with reported zero accuracy loss, using polar-coordinate quantisation combined with a Johnson-Lindenstrauss residual error correction. Tested on Gemma and Mistral. The performance claim is up to 8× speedup over unquantised keys on H100 hardware.

For the build documented here, a 3-bit KV cache would compress to roughly 37% of the current Q8_0 size:

| Context | Q8_0 today | TurboQuant projected | Saved |
|---|---|---|---|
| 64K | 1,158 MiB | ~430 MiB | ~730 MiB |
| 131K | 1,838 MiB | ~690 MiB | ~1,150 MiB |
| 262K | 2,879 MiB | ~1,080 MiB | ~1,800 MiB |

That headroom matters. It would either allow 262K to run with `--parallel 4` (multi-slot at full native context), or leave room for an MTP drafter alongside the main model, or both.

## If Both Land in llama.cpp

Combined, MTP and TurboQuant on this hardware could plausibly mean:

| Today (verified) | Conservative (1.5×) | Mid (2.0×) | Optimistic (2.5×) |
|---|---|---|---|
| 64K @ 40 tok/s | 60 | 80 | 100 |
| 131K @ 37 tok/s | 55 | 74 | 92 |
| 262K @ 26.74 tok/s | 40 | 53 | 67 |

Worth pausing on this projection range. Even at the conservative 1.5× lower bound, 262K context decodes at 40 tok/s — the same speed as today's 64K production figure, but at four times the context. The 2.5× optimistic ceiling puts 262K at 67 tok/s, which as recently as 2024 was data-centre territory. None of this has been measured on this build yet. But the projections are conservative against the published chart, the underlying mechanism is well-understood, and 16 GB consumer AMD hardware sits closer to genuinely useful long-context inference than most observers — local-first or cloud-first — currently price into their thinking.

## The 31B Dense Question

Look at the A100 numbers again: 1.5× for 26B MoE, 3.0× for 31B Dense — same hardware, doubled gain. The bigger dense model exposes more memory pressure for MTP to capture back. If that pattern holds on consumer AMD, a 24 GB AMD card running Gemma 4 31B Dense with both MTP and TurboQuant could see substantially higher speedups than the 26B build documented here.

The MoE architecture trades total parameters for active parameters; the 31B Dense uses every parameter on every token, with the reasoning capacity that implies. At 16 GB with IQ3_M and current quantisation, 31B Dense does not fit usefully. With TurboQuant's KV cache compression and MTP's speedup, it might.

The build process documented here would translate directly. The hardware is the constraining piece — a 24 GB AMD card would bring 31B Dense well within the practical envelope and let a properly resourced AMD builder map territory this article cannot reach. If anyone in the AMD Dev Group community has access to hardware in that class and would like to compare notes on a 31B build, that is the experiment that would close this loop.

## The Twilight Zone Summary

| Today (verified on this build) | Speculative |
|---|---|
| 64K @ 40 tok/s, parallel=4 | 64K @ 60–100 tok/s with MTP |
| 131K @ 37 tok/s, parallel=4 | 131K @ 55–92 tok/s with MTP |
| 262K @ 26.74 tok/s, parallel=1 | 262K @ 40–67 tok/s, parallel=4, with MTP + TurboQuant |
| 26B MoE on 16 GB | 31B Dense on 24 GB |

None of the speculative column has been measured on this build. All of it is conditional on llama.cpp implementing MTP and TurboQuant on the HIP backend, which is itself conditional on continued ROCm investment from AMD. Worth following.

---

## Related Reading

This article is part of an ongoing series on local AI infrastructure and enterprise technology. The companion essay series **Logical Zombies** at zombies.ie covers AI and enterprise technology from a practitioner perspective.

---

*Article_ver07 | 09/May/2026 | Des Donnelly | https://www.dd.ie/*