Why Gemma 4's Apache 2.0 License Matters More Than Its Benchmarks

April 11, 2026 / Gavin Jackson

gemma google ai llm apache-2 ollama local-ai open source

Google's latest open models aren't just capable — they're actually open.

Google dropped Gemma 4 last week, and while the tech press is busy comparing benchmark scores, I think they're missing the bigger story. Yes, the models are impressive — the 26B Mixture-of-Experts variant punches well above its weight class, and the 31B Dense model currently sits at #3 on the Arena AI open-source leaderboard. But the real headline here is buried in the licensing fine print: **Gemma 4 is released under a true Apache 2.0 license.**

This is a bigger deal than it sounds.

The Licensing Problem Nobody Talks About

If you want the primary source on the release itself, Google's official announcement is here: Gemma 4: Byte for byte, the most capable open models. For the license text itself, see the official Apache License 2.0.

For years, "open" AI models have come with strings attached. Meta's Llama models have a custom license with usage restrictions. Previous Gemma versions used Google's own Gemma Terms of Use, which included clauses that made enterprises nervous. Even "open weight" often meant "you can download it, but good luck using it commercially without legal review."

The Apache 2.0 license changes the game completely. It means:

- **No usage restrictions** — Use it for whatever you want. Commercial products? Fine. Competitive services? Go ahead.
- **No attribution requirements beyond the license** — Just include the license file and you're good.
- **Patent protection** — Google explicitly grants patent rights to users.
- **True forkability** — You can modify, redistribute, and even sell your derivatives.

Google acknowledged this directly in their announcement: "We listened closely to what innovators need next to push the boundaries of AI." Translation: developers were tired of legal uncertainty, and Google finally responded.

Real-World Performance: My Testing

I spent the weekend running Gemma 4 on my local workstation to see how these models perform in practice. My setup:

- **OS:** Windows 11
- **GPU:** NVIDIA RTX 5070 Ti (16GB VRAM)
- **Runtime:** Ollama
- **Models tested:** 12B and 27B parameter variants

Getting Started with Ollama

If you want the model listing itself, Ollama's official library page is here: gemma4 on Ollama.

If you want to try Gemma 4 locally, Ollama makes it trivial. Here's what I ran:

```
# Install Ollama (Windows powershell)
irm https://ollama.com/install.ps1 | iex

# Run the Gemma 4 model
ollama run gemma4
```

That's it. Ollama handles the download, quantization, and serving automatically.

The 12B Model: A Pleasant Surprise

Running the 12B parameter model on the 5070 Ti was genuinely impressive. Token generation was blazingly fast — we're talking noticeably quicker than similarly-sized Llama variants I've tested on the same hardware. For local inference without hitting an API, the speed-to-quality ratio feels like a breakthrough.

Initial testing on coding tasks, reasoning problems, and general chat produced solid results. The model feels "smart" in a way that some larger models don't — it's not just regurgitating training data, it's actually following instructions and reasoning through problems.

The 27B Model: VRAM Reality Check

The 27B parameter model was a different story. It ran, but slowly. The issue is VRAM — 16GB just isn't enough to hold the full model in memory without aggressive quantization. For the 27B to shine, you'd want 32GB+ VRAM (think RTX 5090 or professional cards).

This isn't a Gemma-specific problem — it's physics. Larger models need more memory. But it's worth knowing before you get excited about running the biggest variant on consumer hardware.

Using Gemma 4 with OpenClaw for a Fully Offline Agent (updated 4/12/2026)

For background on why OpenClaw is interesting as an agent runtime, I wrote more about that in OpenClaw, Bob, and a Small Taste of the Future. If you want the official project links, start with the OpenClaw GitHub repository and the OpenClaw docs.

One of the more interesting things I tested was pairing Gemma 4 with OpenClaw for a fully offline agent workflow. If your goal is private, local-first automation without sending prompts or files to a cloud API, this is a genuinely compelling setup.

The nice part is that OpenClaw now has first-class Ollama support, so getting it talking to a local Gemma 4 instance is much easier than it used to be.

OpenClaw + Ollama Setup

The official docs that matter most here are OpenClaw's Getting Started guide, the Ollama provider docs, and the OpenClaw FAQ entry that documents the local model flow.

The simplest path is:

```
# Make sure Ollama is installed and the model is available locally
ollama pull gemma4

# Install OpenClaw
curl -fsSL https://openclaw.ai/install.sh | bash

# Run onboarding and choose Ollama
openclaw onboard
```

During onboarding, OpenClaw can detect your local Ollama server, discover installed models, and let you pick `gemma4` as the default. If you want a local-only setup, choose the `Local` option rather than `Cloud + Local`.

If you prefer to do it manually, the important bits are:

- Make sure Ollama is running and reachable on `http://127.0.0.1:11434`
- Set any non-empty Ollama API key value for OpenClaw, for example `OLLAMA_API_KEY=ollama-local`
- Use the native Ollama endpoint, **not** the OpenAI-compatible `/v1` endpoint
- Set your default model to `ollama/gemma4`

That looks roughly like this:

```
export OLLAMA_API_KEY="ollama-local"
openclaw models list
openclaw models set ollama/gemma4
```

And if you want to sanity check the local runtime before blaming OpenClaw:

```
ollama list
curl http://127.0.0.1:11434/api/tags
```

That last detail about the endpoint matters more than it should. OpenClaw works best with Ollama's native API, so you want `http://127.0.0.1:11434` and not `http://127.0.0.1:11434/v1`. Using `/v1` can break tool calling, which is exactly the kind of bug that makes local agents feel flaky.

How It Felt in Practice

In my testing, Gemma 4 paired with OpenClaw felt surprisingly responsive for a fully offline agent running on commodity hardware. It didn't feel quite as capable as Kimi K2.5 or Codex for more complex agentic work, especially when tasks required longer chains of reasoning or cleaner tool use, but it was still far better than I expected for a model I could run entirely on my own machine.

That's the part I keep coming back to: this is no longer a science project. You can stand up a private, local agent stack with Ollama and OpenClaw, point it at Gemma 4, and get something genuinely useful without needing datacenter-class hardware.

Why Apache 2.0 Changes Everything

Let's talk about what this license shift actually means for the industry:

For Startups

Previously, if you wanted to build a product on an open model, you needed legal review of custom licenses. Apache 2.0 is boring, standard, and well-understood. Your lawyers already know it. This removes friction and lets small teams move faster.

For Enterprise

Big companies have been cautious about "open" AI because of license uncertainty. Apache 2.0 is enterprise-friendly and has been battle-tested for decades. Expect to see Gemma 4 showing up in more commercial products simply because the legal risk dropped to near zero.

For the Open Source Ecosystem

Google says the Gemmaverse already includes more than 100,000 model variants, which suggests there is already meaningful community momentum behind the family. With Gemma 4 now under Apache 2.0, that ecosystem can accelerate further. Developers can fine-tune, merge, and redistribute derivatives with far fewer licensing constraints than custom "source-available" model licenses, while still complying with standard Apache notice requirements. We might see Gemma 4 become a popular base for downstream variants in the same way Llama-family models did, but with a much cleaner licensing story.

For Google

This is the interesting part. Google isn't giving up control out of altruism — they're making a strategic bet. By releasing truly open models, they:

1. **Undercut competitors** who are still using restrictive licenses
2. **Build ecosystem lock-in** through tooling and integration (Gemma works great with Google's AI stack)
3. **Establish standards** — if Gemma 4 becomes the default "safe choice" for open models, Google shapes the direction of on-device and local AI

It's a long game, and it's smart.

The Bottom Line

Gemma 4's benchmark scores will get the headlines, but the Apache 2.0 license is what matters long-term. It represents a shift in how major AI labs think about openness — from "open-ish with caveats" to "actually open, go build something."

For developers, this is what we've been asking for. A capable model we can run locally, modify freely, and ship in products without legal anxiety.

Google finally delivered. Let's see what the community builds with it.

References and Further Reading

- Google's Gemma 4 announcement
- gemma4 in the Ollama model library
- OpenClaw GitHub repository
- OpenClaw documentation
- OpenClaw Getting Started
- OpenClaw Ollama provider docs
- OpenClaw FAQ: self-hosted models and Ollama
- Apache License 2.0
- Apache License FAQ

Related Posts

- OpenClaw, Bob, and a Small Taste of the Future
- OpenClaw Alternatives and the Dream of Extensible AI Agents
- Using Codex to Rebuild My Blog and Import 15+ Years of Blogger Posts

Hardware tested: RTX 5070 Ti (16GB), AMD Ryzen 9 7950X, 64GB DDR5
OS: Windows 11
Software: Ollama
Models: gemma4:12b, gemma4:27b