NEWS · JUNE 9, 2026

Can a 30B Model on Your Mac Cut Your AI Bill? Two Local LLMs vs. One Cloud Fact-Checker

By Dr. Holger Flick

The monthly bill for cloud AI is the new electricity bill — and it keeps climbing. So a tempting question keeps coming up: how much of the expensive work can I push onto a model running on my own hardware, for free — and is that work actually *good enough* to trust?

So I ran an experiment. I had two local LLMs perform a full security review of a real Next.js project, then asked a cloud model (Claude Opus 4.8 inside Claude Code) to **fact-check both reports against the actual source code** and tell me, honestly, whether the local step saved it any work. The twist: the cloud model wasn't allowed to peek at the code first. It only got the two Markdown reports — and then verified every interesting claim with `grep` and file reads.

The result is more interesting than a simple "local good" or "local bad." One report was a genuine time-saver. The other was confidently, verifiably wrong in places. Here's the whole thing.

The contenders

To see how local models hold up inside the Claude Code CLI, I used **Gemma 4 (31B)** and **Qwen 3.6 (35B)** at the highest precision available for Ollama. Even at "only" 30-something billion parameters, full `bf16` precision means these models eat **over 60 GB of memory each**. They ran on a MacBook Pro M5 Max with 128 GB of unified memory.

```
ollama list

NAME                          ID              SIZE     MODIFIED
gemma4:31b-mlx-bf16           cd34f05c33e9    62 GB    About an hour ago
qwen3.6:35b-a3b-coding-bf16   8d3c7ad727e7    70 GB    16 hours ago
```

To launch Claude Code against a local model:

```
ollama launch claude
```

Or pin a specific model:

```
ollama launch claude --model <name of model>
```

Running on the GPU

On Apple Silicon, make sure the model runs on the GPU and use an **MLX** build when one exists — MLX models are heavily optimized for the Apple Neural Engine and run noticeably faster. For example, prefer `gemma4-32b-mlx` over `gemma4-32b`.

Verify how the model loaded with `ollama ps`:

```
ollama ps

NAME                    ID              SIZE     PROCESSOR    CONTEXT    UNTIL
gemma4:31b-mlx-bf16    cd34f05c33e9    61 GB    100% GPU     262144     4 minutes from now
```

Note: for this run the Gemma 4 model had an MLX build, but the Qwen 3.6 model did not have an MLX build *at this precision* (other precision levels do). I deliberately picked the best available build for each, so the wall-clock times are **not directly comparable** — and that's fine, because this experiment is about *accuracy*, not speed.

The prompt

The task was a focused security review: in a Next.js app, **Server Actions are not protected by layouts or middleware** — they're effectively public HTTP endpoints. If an action doesn't call `getSession()` itself, anyone who knows the endpoint can invoke it. That's exactly the kind of systemic issue an LLM should be good at sweeping for.

> this is a nextjs project and it uses server-side code. especially server-actions need to be authenticated. please review the code and create a report in markdown if there are missing calls to check if the user is authenticated in your opinion. in a nutshell: review that only authenticated users can access protected areas. note that there are token-authenticated pages (a few). name the report `claude_review_<name of model>.md`

Both models delivered within about 10 minutes. (I ran them unattended and occasionally had to confirm a step, so I'm not quoting times to the second.)

The output: two Markdown reports.

- **Gemma 4** produced a concise, file-level report — a categorized list of which action files lack auth, plus solid generic recommendations.
- **Qwen 3.6** produced a far more ambitious audit: per-function tables, line numbers, severity ratings, a summary count, and a prioritized fix list. It read like a professional pentest report.

On first glance, Qwen wins by a mile. But "looks authoritative" and "is correct" are not the same thing — which is the entire point of the next step.

The comparison

I asked Claude Opus 4.8 (via Claude Code) to compare the two reports and estimate whether the local step would save it tokens. The obvious objection — "the cloud model will just cheat and use its own analysis" — doesn't hold here: Claude Code logs every file it touches, and to *fact-check* the claims it deliberately re-read the source and ran `grep`. That verification is the work, and it's exactly what we want to measure.

> @claude*review*gemma4.md and @claude*review*qwen3.md have been created with local LLMs with those respective models. Compare the created results with regards to level of detail and accuracy and if you - as a cloud LLM - would be able to act on this and this would lead to saving tokens as without these reports you

> would need to investigate this yourself. Give me an honest estimate if this local step is helpful for both, one of them or not at all.

The verdict

The cloud model didn't take either report at face value. It spot-checked the specific, falsifiable claims against the code. Here's what it found.

What's actually true (verified)

The **core finding is correct and genuinely valuable**: many Server Actions in the `(protected)` route group rely solely on the layout/middleware and never call `getSession()` themselves. Verified directly:

- `users/actions.ts` — zero auth references. `createUser`, `updateUser`, `resetUserPassword`, `deleteUser` are all reachable unauthenticated. Real and critical.
- `app/api/migration/run/route.ts` — the `POST` handler runs `TRUNCATE` and a full re-migration with **no `getSession()` at all**. Real, and the scariest item on the list.
- `app/api/generate-bulk-reports/route.ts` and `app/api/tinker/bulk-pdf-stream/route.ts` — unauthenticated `GET` handlers that read and dump sensitive data. Real.
- `entries-actions.ts` — a subtle one: `createEntry` *does* call `getSession()`, but it **never checks** `isLoggedIn`. It just does `...(session.userId ? { createdBy: ... } : {})`, so an unauthenticated caller still creates the entry — the audit field is simply dropped. Verified correct.

Where Qwen was wrong (verified false claims)

This is where the authoritative-looking report bites back. Qwen confidently stated, in formatted tables:

- **H5 —** `widgets/[id]/comments/actions.ts`**:** "All functions have NO `getSession()` call." **False.** `createComment`, `updateComment`, `addFollowUp`, `resolveFollowUp`, `editFollowUp`, and `deleteFollowUp` all call `getSession()`. The file has seven auth references.
- **H7 —** `records/[id]/items/[itemId]/actions.ts`**:** Same "All functions have NO `getSession()`" claim. **False** — at least six functions call it.

These aren't nitpicks. A confident, table-formatted "NO" for code that *does* check auth is the worst kind of error: act on it blindly and you'd "fix" already-correct code, and — more corrosively — you'd start distrusting the rows that *are* right.

Head-to-head

	Gemma 4	Qwen 3.6
Core finding correct		
Granularity	File-level only	Per-function tables, line numbers
API routes (migration, bulk-reports, tinker)	missed all three	found, the highest-value catches
Subtle `isLoggedIn`-not-checked bug	missed	caught (correct)
`lib/` actions outside `(protected)`	missed	found

False positives ("auth missing" when present)	Low risk (vaguer, file-level)	at least two confirmed false
Line-number citations	None	Yes (some off or invented)

Does the local step save the cloud model tokens?

The honest answer is **"yes, but only one of them, and only as a lead generator — never as ground truth."**

Qwen 3.6: net helpful. It surfaced the three unauthenticated API routes and the subtle `isLoggedIn` bug — the two highest-value findings in the whole review, and exactly the things a cold investigation would burn the most tokens discovering. That's a real saving on the *"where do I even look?"* phase. But its confident false negatives mean the cloud model **cannot trust any individual row** and must re-verify each one. So Qwen saves the discovery phase, not the verification phase. Treated as a checklist-to-confirm, it's a win; treated as truth, it's a liability.

Gemma 4: marginally helpful. It got the concept and a clean, low-false-positive file list right — but it's shallow. No API routes, no subtle bug, no per-function detail. It tells the cloud model roughly what a single `grep -L getSession 'app/**/actions.ts'` would, which is one tool call. Not enough to change the workflow.

The uncomfortable meta-lesson: a report that *looks* this authoritative but is partly wrong can be **more dangerous than no report**, because the polished table format invites copy-paste trust. The value is real — but it's entirely conditional on a disciplined verify-everything pass on top.

Takeaways

1. **Local models are excellent scouts, unreliable judges.** A 30B model on a Mac can absolutely point a more capable model at the right files and the non-obvious bugs. Letting it render final verdicts is where it falls down.
2. **Confidence is not accuracy.** Qwen's pentest-grade formatting made its wrong rows just as convincing as its right ones. Polish is not evidence.
3. **The winning workflow is hybrid.** Run the cheap local sweep to generate leads, then spend cloud tokens *verifying and acting*, not *discovering*. That division of labor is where the money actually gets saved.
4. **Always keep a fact-checker in the loop.** The single most useful step in this entire experiment wasn't either local review — it was forcing the cloud model to confirm the claims against source before believing them.

If you want one trustworthy artifact out of all this, the move is clear: take Qwen's report as the lead list, re-verify each file, and merge in the few things Gemma caught that Qwen didn't. That deduplicated, *verified* list is the thing worth committing — not either raw report.

INDEPENDENT · AD-FREE · READER-SUPPORTED

Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

Ways to support

KO-FI

Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

Support on Ko-fi —