NEWS · JUNE 9, 2026

Kann ein 30B-Modell auf Ihrem Mac Ihre AI-Rechnung senken? Zwei lokale LLMs gegen einen Cloud-Faktenprüfer

By Dr. Holger Flick

Die monatliche Rechnung für Cloud-AI ist die neue Stromrechnung — und sie steigt weiter. Da drängt sich eine verlockende Frage immer wieder auf: Wie viel der teuren Arbeit kann ich auf ein Modell verlagern, das kostenlos auf meiner eigenen Hardware läuft — und ist diese Arbeit tatsächlich *gut genug*, um ihr zu vertrauen?

Also habe ich ein Experiment gewagt. Ich ließ zwei lokale LLMs ein vollständiges Security-Review eines echten Next.js-Projekts durchführen und bat anschließend ein Cloud-Modell (Claude Opus 4.8 in Claude Code), **beide Berichte gegen den tatsächlichen Quellcode zu verifizieren** und mir ehrlich zu sagen, ob der lokale Schritt ihm Arbeit erspart hat. Der Clou: Das Cloud-Modell durfte vorher keinen Blick in den Code werfen. Es bekam nur die beiden Markdown-Berichte — und überprüfte dann jede interessante Behauptung mit `grep` und dem Lesen der Dateien.

Das Ergebnis ist interessanter als ein simples „lokal gut" oder „lokal schlecht". Ein Bericht war eine echte Zeitersparnis. Der andere war an einigen Stellen selbstbewusst und nachweisbar falsch. Hier ist die ganze Geschichte.

Die Kontrahenten

Um zu sehen, wie sich lokale Modelle innerhalb der Claude Code CLI schlagen, habe ich **Gemma 4 (31B)** und **Qwen 3.6 (35B)** in der höchsten für Ollama verfügbaren Präzision verwendet. Selbst mit „nur" gut 30 Milliarden Parametern verschlingen diese Modelle bei voller `bf16`-Präzision **jeweils über 60 GB Speicher**. Sie liefen auf einem MacBook Pro M5 Max mit 128 GB 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
```

So starten Sie Claude Code gegen ein lokales Modell:

```
ollama launch claude
```

Oder mit einem festgelegten Modell:

```
ollama launch claude --model <Name des Modells>
```

Ausführung auf der GPU

Stellen Sie auf Apple Silicon sicher, dass das Modell auf der GPU läuft, und verwenden Sie einen **MLX**-Build, sofern einer existiert — MLX-Modelle sind stark für die Apple Neural Engine optimiert und laufen spürbar schneller. Bevorzugen Sie beispielsweise `gemma4-32b-mlx` gegenüber `gemma4-32b`.

Überprüfen Sie mit `ollama ps`, wie das Modell geladen wurde:

```
ollama ps

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

Anmerkung: Für diesen Durchlauf gab es vom Gemma-4-Modell einen MLX-Build, vom Qwen-3.6-Modell jedoch keinen MLX-Build *in dieser Präzision* (in anderen Präzisionsstufen schon). Ich habe bewusst für jedes Modell den besten verfügbaren Build gewählt, daher sind die Laufzeiten **nicht direkt vergleichbar** — und das ist in Ordnung, denn in diesem Experiment geht es um *Genauigkeit*, nicht um Geschwindigkeit.

Der Prompt

Die Aufgabe war ein fokussiertes Security-Review: In einer Next.js-App sind **Server Actions nicht durch Layouts oder Middleware geschützt** — sie sind faktisch öffentliche HTTP-Endpunkte. Wenn eine Action nicht selbst `getSession()` aufruft, kann jeder, der den Endpunkt kennt, sie aufrufen. Genau die Art von systematischem Problem, für dessen Durchforstung ein LLM gut geeignet sein sollte.

> 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`

Beide Modelle lieferten innerhalb von etwa 10 Minuten. (Sie liefen unbeaufsichtigt, und gelegentlich musste ich einen Schritt bestätigen, daher nenne ich keine sekundengenauen Zeiten.)

Das Ergebnis: zwei Markdown-Berichte.

- **Gemma 4** erstellte einen knappen Bericht auf Dateiebene — eine kategorisierte Liste der Action-Dateien ohne Auth-Prüfung plus solide generische Empfehlungen.
- **Qwen 3.6** lieferte ein deutlich ambitionierteres Audit: Tabellen pro Funktion, Zeilennummern, Schweregrad-Einstufungen, eine zusammenfassende Zählung und eine priorisierte Fix-Liste. Es las sich wie ein professioneller Pentest-Bericht.

Auf den ersten Blick gewinnt Qwen haushoch. Aber „wirkt autoritativ" und „ist korrekt" sind nicht dasselbe — und genau darum geht es im nächsten Schritt.

Der Vergleich

Ich bat Claude Opus 4.8 (via Claude Code), die beiden Berichte zu vergleichen und abzuschätzen, ob der lokale Schritt ihm Token ersparen würde. Der naheliegende Einwand — „das Cloud-Modell schummelt doch einfach und nutzt seine eigene Analyse" — greift hier nicht: Claude Code protokolliert jede Datei, die es anfasst, und um die Behauptungen zu *verifizieren*, hat es den Quellcode bewusst erneut gelesen und `grep` ausgeführt. Diese Verifikation ist die eigentliche Arbeit — und genau das, was wir messen wollen.

> @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.

Das Urteil

Das Cloud-Modell nahm keinen der beiden Berichte für bare Münze. Es prüfte die konkreten, falsifizierbaren Behauptungen stichprobenartig gegen den Code. Das kam dabei heraus.

Was tatsächlich stimmt (verifiziert)

Der **Kernbefund ist korrekt und wirklich wertvoll**: Viele Server Actions in der `(protected)`-Routengruppe verlassen sich ausschließlich auf Layout/Middleware und rufen nie selbst `getSession()` auf. Direkt verifiziert:

- `users/actions.ts` — keinerlei Auth-Referenzen. `createUser`, `updateUser`, `resetUserPassword`, `deleteUser` sind alle ohne Authentifizierung erreichbar. Echt und kritisch.
- `app/api/migration/run/route.ts` — der `POST`-Handler führt `TRUNCATE` und eine vollständige Re-Migration aus, **ganz ohne** `getSession()`. Echt — und der beängstigendste Punkt auf der Liste.
- `app/api/generate-bulk-reports/route.ts` und `app/api/tinker/bulk-pdf-stream/route.ts` — nicht authentifizierte `GET`-Handler, die sensible Daten auslesen und ausgeben. Echt.
- `entries-actions.ts` — ein subtiler Fall: `createEntry` ruft zwar `getSession()` auf, prüft aber **nie** `isLoggedIn`. Es macht nur `...(session.userId ? { createdBy: ... } : {})`, sodass ein nicht authentifizierter Aufrufer den Eintrag trotzdem anlegt — das Audit-Feld fällt einfach weg. Als korrekt verifiziert.

Wo Qwen falsch lag (verifiziert falsche Behauptungen)

Hier rächt sich der autoritativ wirkende Bericht. Qwen behauptete selbstbewusst, in formatierten Tabellen:

- **H5 —** `widgets/[id]/comments/actions.ts`**:** „All functions have NO `getSession()` call." **Falsch.** `createComment`, `updateComment`, `addFollowUp`, `resolveFollowUp`, `editFollowUp` und `deleteFollowUp` rufen alle `getSession()` auf. Die Datei enthält sieben Auth-Referenzen.
- **H7 —** `records/[id]/items/[itemId]/actions.ts`**:** Dieselbe Behauptung „All functions have NO `getSession()`". **Falsch** — mindestens sechs Funktionen rufen es auf.

Das sind keine Kleinigkeiten. Ein selbstbewusstes, tabellarisch formatiertes „NO" für Code, der die Auth-Prüfung *sehr wohl* durchführt, ist die schlimmste Art von Fehler: Handelt man blind danach, „repariert" man bereits korrekten Code — und, noch zersetzender, man beginnt auch den Zeilen zu misstrauen, die *stimmen*.

Der direkte Vergleich

	Gemma 4	**Qwen 3.6**
Kernbefund korrekt		
Granularität	Nur Dateiebene	Tabellen pro Funktion, Zeilennummern
	lalle drei übersehen	gefunden, die wertvollsten Treffer

API-Routen (migration, bulk-reports, tinker)		
Subtiler Bug: `isLoggedIn` nicht geprüft	Übersehen	erkannt (korrekt)
`lib/`-Actions außerhalb von `(protected)`	Übersehen	gefunden
False Positives („Auth fehlt", obwohl vorhanden)	Geringes Risiko (vager, Dateiebene)	mindestens zwei als falsch bestätigt
Zeilennummern-Angaben	Keine	Ja (teils verschoben oder erfunden)

Spart der lokale Schritt dem Cloud-Modell Token?

Die ehrliche Antwort lautet: **„Ja, aber nur bei einem der beiden — und nur als Lead-Generator, niemals als Ground Truth."**

Qwen 3.6: unterm Strich hilfreich. Es förderte die drei nicht authentifizierten API-Routen und den subtilen `isLoggedIn`-Bug zutage — die beiden wertvollsten Befunde des gesamten Reviews, und genau die Dinge, deren Entdeckung eine Untersuchung ohne Vorwissen die meisten Token kosten würde. Das ist eine echte Ersparnis in der *„Wo soll ich überhaupt suchen?"*-Phase. Aber wegen seiner selbstbewussten falschen Negative kann das Cloud-Modell **keiner einzelnen Tabellenzeile trauen** und muss jede nachprüfen. Qwen spart also die Discovery-Phase, nicht die Verifikationsphase. Als abzuarbeitende Checkliste behandelt ist es ein Gewinn; als Wahrheit behandelt eine Belastung.

Gemma 4: nur geringfügig hilfreich. Es erfasste das Konzept und lieferte eine saubere Dateiliste mit wenigen False Positives — aber es bleibt oberflächlich. Keine API-Routen, kein subtiler Bug, keine Details pro Funktion. Es sagt dem Cloud-Modell ungefähr das, was ein einzelnes `grep -L getSession 'app/**/actions.ts'` sagen würde — also ein einziger Tool-Aufruf. Zu wenig, um den Workflow zu verändern.

Die unbequeme Meta-Lektion: Ein Bericht, der derart autoritativ *aussieht*, aber teilweise falsch ist, kann **gefährlicher sein als gar kein Bericht**, weil das polierte Tabellenformat zu blindem Copy-Paste-Vertrauen einlädt. Der Wert ist real — aber er hängt vollständig von einem disziplinierten Alles-verifizieren-Durchgang obendrauf ab.

Erkenntnisse

1. **Lokale Modelle sind exzellente Späher, aber unzuverlässige Richter.** Ein 30B-Modell auf einem Mac kann ein leistungsfähigeres Modell absolut auf die richtigen Dateien und die nicht offensichtlichen Bugs stoßen. Endgültige Urteile zu fällen — daran scheitert es.
2. **Selbstbewusstsein ist keine Genauigkeit.** Qwens Formatierung auf Pentest-Niveau machte seine falschen Zeilen genauso überzeugend wie seine richtigen. Politur ist kein Beweis.
3. **Der Gewinner-Workflow ist hybrid.** Lassen Sie den günstigen lokalen Durchlauf Leads generieren und geben Sie Cloud-Token dann fürs *Verifizieren und Handeln* aus, nicht fürs *Entdecken*. In dieser Arbeitsteilung steckt die tatsächliche Ersparnis.
4. **Behalten Sie immer einen Faktenprüfer im Prozess.** Der nützlichste Schritt in diesem gesamten Experiment war keines der beiden lokalen Reviews — sondern das Cloud-Modell zu zwingen, die Behauptungen gegen den Quellcode zu bestätigen, bevor es ihnen glaubt.

Wenn Sie aus alledem ein einziges vertrauenswürdiges Artefakt mitnehmen wollen, ist der Weg klar: Nehmen Sie Qwens Bericht als Lead-Liste, verifizieren Sie jede Datei erneut und ergänzen Sie die wenigen Punkte, die Gemma gefunden hat und Qwen nicht. Diese deduplizierte, *verifizierte* Liste ist das, was es wert ist, committet zu werden — nicht einer der beiden Rohberichte.

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 —