NEWS · JULY 14, 2026

Qwen3.6 hat mir einen Festgeld-Rechner geschrieben — und warum ich Claude trotzdem die Mathematik prüfen ließ

By Dr. Holger Flick



Ein Modell, klein genug, um vollständig auf meinem Laptop zu laufen, hat aus einem einzigen Satz einen funktionierenden Festgeld-Rechner mit Diagramm gebaut. Es war schnell, es sah großartig aus, und die Arithmetik stimmte. Es war aber auch — ganz still — das falsche Werkzeug. Und das zu erkennen, brauchte eine zweite Meinung aus der Cloud.

Alle sprechen über Qwens neueste Veröffentlichung — Qwen3.6, das 27-Milliarden-Parameter-Modell des

Qwen-Teams von Alibaba. Die Geschichte lautet: Ein 4-Bit-quantisierter Build läuft auf Hardware, die Sie ohnehin besitzen, und erzeugt wirklich guten Code, besonders in JavaScript und Python. Ich wollte zwei Dinge wissen: Stimmt diese Geschichte, und wie weit vertraue ich der Ausgabe? Also gab ich dem Modell eine kleine, ehrliche Aufgabe — einen Rechner für ein Certificate of Deposit (CD, im deutschen Raum am ehesten Festgeld) — und übergab das Ergebnis dann Claude in der Cloud als Schiedsrichter.

Was folgt, ist die gesamte Schleife: ein schnelles lokales Modell, ein plausibler-aber-falscher erster Entwurf, drei Iterationsrunden und ein Cloud-Modell, das genau das fing, worauf es ankam. Die Pointe ist nicht „lokal schlecht, Cloud gut" — sie ist das Gegenteil. Lokale KI ist inzwischen gut genug, um ein tägliches Werkzeug zu sein. Man muss nur weiterhin die Teile prüfen, die einem wehtun können.

> **Ich habe denselben Test auf einem zweiten lokalen Modell gemacht**

> Nach diesem Beitrag gab ich den *identischen* Prompt einem völlig anderen lokalen Modell — Ornith 1.0 35B, einem Mixture-of-Experts-Modell — und verglich die beiden direkt. Kurz gesagt: Es machte den *exakt gleichen* Fachfehler und reparierte sich dann beeindruckend selbst. Der vollständige Vergleich steht hier: Zwei lokale Coding-Modelle, ein Prompt.

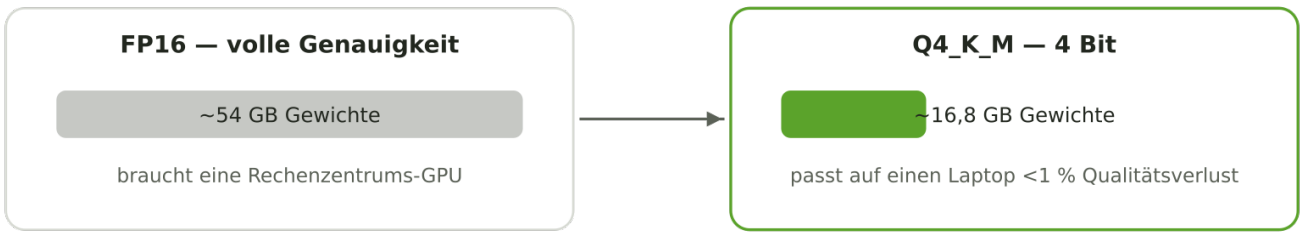
Das Modell: klein genug für den Schreibtisch, gut genug, um zu zählen

Vor der Aufgabe ein Wort dazu, was „läuft lokal" 2026 tatsächlich bedeutet, denn genau die Zahlen sind der Punkt.

Qwen3.6-27B ist ein dichtes 27-Milliarden-Parameter-Modell, veröffentlicht am 22. April 2026 unter der freizügigen Apache-2.0-Lizenz. „Dicht" (dense) bedeutet, dass jeder Parameter bei jedem Token aktiv ist — im Gegensatz zu einem Mixture-of-Experts-Modell (MoE), das pro Token nur einen Ausschnitt seiner selbst aktiviert. Auf SWE-bench Verified, einem gängigen Benchmark für agentisches Programmieren, erreicht es **77,2 %**, gegenüber 75,0 % bei der Vorgängerversion — und übertrifft dabei bemerkenswerterweise Alibabas eigenes, deutlich größeres 397B-MoE-Modell in diesem Benchmark (Quelle).

Der Grund, warum es auf einen Schreibtisch passt, ist die **Quantisierung**: das Speichern der Gewichte des Modells mit geringerer numerischer Genauigkeit. Ein Gewicht in voller Genauigkeit (FP16) belegt 16 Bit; ein 4-Bit-quantisiertes Gewicht belegt vier. Der empfohlene `Q4_K_M`-Build von Qwen3.6-27B wiegt etwa **16,8 GB** und läuft in rund **18 GB** RAM oder VRAM — bei unter 1 % Qualitätsverlust gegenüber voller Genauigkeit (Quelle).

Lassen Sie mich den Kompromiss visuell zeigen, denn er ist der ganze Grund, warum dies möglich ist.



Quantisierung tauscht einen Hauch Qualität gegen ein Modell, das in Verbraucherspeicher passt

Die Erkenntnis aus dem Diagramm: Der 4-Bit-Build hat etwa ein Drittel des Speicherbedarfs, und die Qualität, die man aufgibt, ist klein genug, um für die meisten Aufgaben ein Rundungsfehler zu sein. Genau das macht aus einem Forschungslabor-Artefakt etwas, das man an einem Dienstag laufen lässt.

Zur Einordnung: Ich habe dies auf einem Apple MacBook Pro mit dem M5 Max und 128 GB RAM ausgeführt. Apple Silicons Unified-Memory-Architektur — bei der CPU und GPU einen gemeinsamen Pool aus Speicher mit hoher Bandbreite nutzen, statt Daten über einen Bus zu kopieren — ist ein echter Vorteil für große Modelle, weil das gesamte Modell in einem Speicher liegt, den die GPU direkt erreicht. Aber beachten Sie die ehrliche Einordnung: **Sie brauchen meine Maschine nicht.** Mit ~18 GB läuft dieses Modell auf einer einzelnen

High-End-Verbraucher-GPU (einer RTX 5090) oder einem Mittelklasse-Mac (Quelle). Meine 128 GB sind für dieses spezielle Modell überdimensioniert; sie bedeuten nur, dass ich nicht darüber nachdenken muss.

> **Wie sich lokal kurz gegen die Cloud schlägt**
> Die heutigen Spitzen-Cloud-Modelle — Claude Opus 4.8 und Sonnet 5 oder OpenAIs GPT-5.6 — arbeiten mit **1-Millionen-Token**-Kontextfenstern und sind in einer Größenordnung trainiert, die kein lokales Modell erreicht. Ein 27B-Modell auf Ihrem Schreibtisch spielt ein anderes Spiel: kleinerer Kontext, weniger Weltwissen, aber *privat, am Rand kostenlos und immer verfügbar*. Der Rest dieses Beitrags handelt davon, wo dieser Unterschied tatsächlich beißt.

Ein faires Wort zu den Alternativen

Qwen ist nicht die einzige glaubwürdige lokale Option, und sie zu erwähnen kostet nichts, schafft aber Vertrauen.

Wenn Sie Modelle wie dieses selbst ausführen möchten, ist das Open-Source-Tooling hervorragend: Ollama und llama.cpp laden beide `Q4_K_M`-GGUF-Builds direkt, und LM Studio bietet Ihnen eine grafische Oberfläche, falls Sie kein Terminal anfassen möchten. Andere starke lokale Familien in derselben Gewichtsklasse gibt es ebenfalls — Metas Llama, Mistral, Googles Gemma. Der Punkt dieses Beitrags ist nicht „Qwen gewinnt"; er ist, dass *jede* dieser Optionen die Schwelle von Spielzeug zu Werkzeug überschritten hat. Wählen Sie nach Passung: Lizenz, Sprachstärke und was Ihre Hardware fassen kann.

Die Aufgabe: ein Satz, eine funktionierende App

Ich hielt den Prompt bewusst lässig — so, wie eine beschäftigte Person tatsächlich fragt — um zu sehen, wie viel das Modell selbst ergänzen würde.

Genau das habe ich getippt:

```
implement a simple webpage in html with vanilla javascript and css. no
dependencies. to enter a money value and a cd percentage with how much
interest you earn in each month with a bar chart. i wanna be able to enter
the number of months with the cd percentage that a bank gives me. like you
get a 4.5% CD
```

Das Modell erzeugte eine einzige, in sich geschlossene Seite — HTML, CSS und JavaScript in einer Datei, kein Build-Schritt. Es verzinste die Zinsen monatlich und rendert ein Balkendiagramm. Das selbst gebaute Diagramm überlappte und sah grob aus, also fügte ich eine Nachfrage hinzu:

```
please use chart.js for the chart
```

Das ist eine größere Sache, als es klingt. Chart.js ist eine leistungsfähige, aber große Bibliothek mit Dutzenden Diagrammtypen und einer tiefen Konfigurationsoberfläche; sie von Hand korrekt zu verdrahten, kostet echte Zeit. Das Modell tauschte in einem einzigen Zug ein sauber konfiguriertes Chart.js-Balkendiagramm ein — genau die Art „Ich kenne diese Bibliothek, damit Sie es nicht müssen"-Hebelwirkung, die diese Werkzeuge lohnenswert macht.

Das Ergebnis sah großartig aus.



Das sieht auf den ersten Blick großartig aus, aber der Schein trügt — eine saubere Oberfläche und ein Chart.js-Balkendiagramm, das die Zinsen Monat für Monat anwachsen zeigt.

Das Diagramm zeigt, wie sich die Zinsen über die Monate zum eingegebenen Satz ansammeln. Sauberes Layout, sinnvolle Beschriftungen, eine funktionierende Live-Berechnung. Hätte ich hier aufgehört, hätte ich es ausgeliefert. Genau das ist die Falle.

Der Schiedsrichter: Übergabe an die Cloud

Ich konnte nicht leicht mit bloßem Auge beurteilen, ob das *Finanzmodell* stimmte — die Arithmetik sah in Ordnung aus, aber „sieht in Ordnung aus" ist genau der Fehlermodus, den ich jagte. Also machte ich einen Screenshot der laufenden App und bat Claude in der Cloud, sie zu prüfen. Entscheidend: **Der Schiedsrichter sah nie meinen Quellcode** — nur das gerenderte Ergebnis —, was widerspiegelt, wie man jede Blackbox-Ausgabe auf Plausibilität prüfen würde.

Das Urteil war schärfer, als ich erwartet hatte. Die Arithmetik stimmte; die *fachliche Modellierung* nicht.

> **Claudes Prüfung (Auszug)**
> Die Arithmetik geht auf — es ist die finanzielle Modellierung, die wackelt. Es hat einen korrekten **Zinseszins**-Rechner geschrieben und ihn als **Festgeld**-Rechner beschriftet. Das ist ein sehr verbreiteter LLM-Fehlermodus: Es implementiert die Lehrbuchformel, auf die es gemustert hat, und überspringt das fachspezifische Detail, das eine Mathe-Übung von einem Finanzwerkzeug trennt.

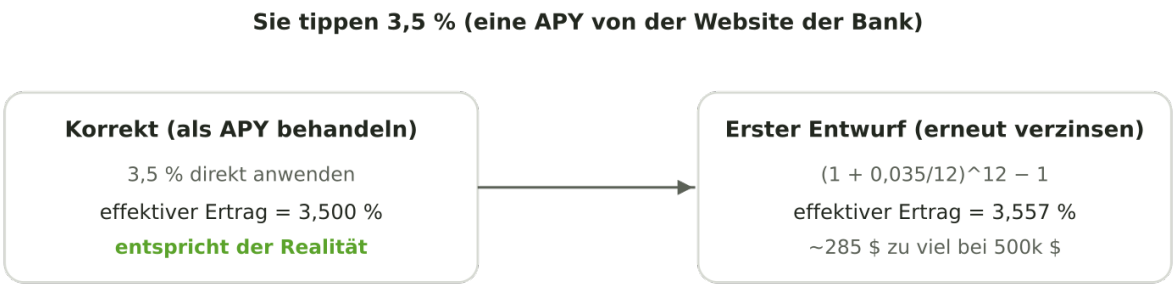
Die konkret bemängelten Lücken waren allesamt real, und sie laufen alle auf eine Sache hinaus: Ein Festgeld ist nicht einfach „Kapital, monatlich verzinst". Lassen Sie mich die wichtigste davon aufdröseln.

Warum „Festgeld" `„Zinseszins"

Diese Unterscheidung ist das Herz des ganzen Beitrags und verdient daher eine klare Erklärung und eine Quelle.

Banken bewerben Festgeld über die **APY** — Annual Percentage Yield, den *effektiven* Jahresertrag, in dem der Zinseszins bereits eingerechnet ist. Ein **nomineller Satz** (in diesem Kontext manchmal APR genannt) ist der Schlagzeilen-Satz *bevor* der Zinseszins angewandt wird (Regions Bank erklärt es hier; Investopedia zur APY). Weil die APY den Zinseszins bereits enthält, ist sie stets e dem nominellen Satz.

Der erste Entwurf nahm den eingegebenen Satz und verzinste ihn monatlich — behandelte also eine *beworbene APY* so, als wäre sie ein *nomineller Satz*. Das zählt den Zinseszins doppelt.



Der Fehler: ein Satz, der bereits eine APY ist, dann erneut verzinst

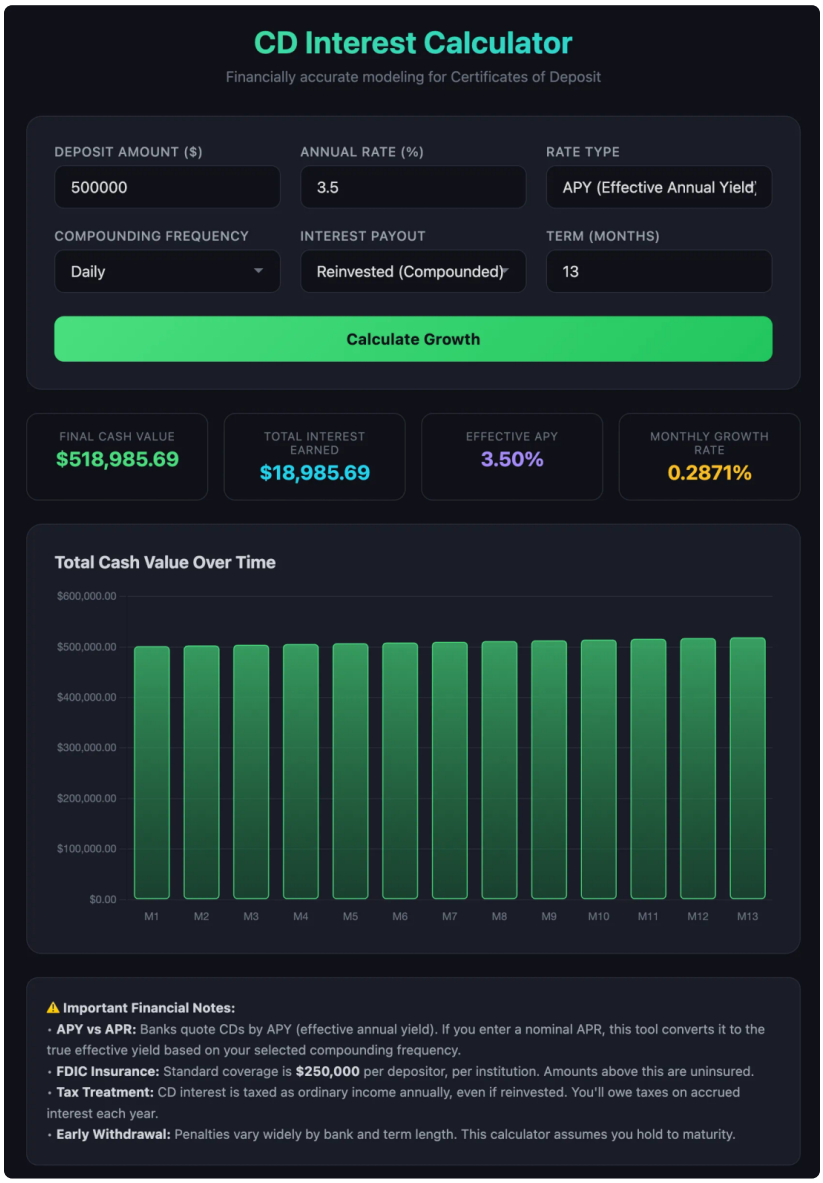
Lesen Sie das Diagramm so: Die Zahl, die das Modell *falsch* hatte, weicht gar nicht so stark ab — etwa 285 $ bei einer halben Million Dollar Einlage. Genau das macht sie gefährlich. Es ist kein Absturz und keine offensichtliche Halluzination; es ist eine plausible Antwort, die ein Laie niemals hinterfragen würde. Die anderen von Claude bemängelten Lücken — keine Auswahl der Verzinsungsfrequenz, keine Option für einfache Zinsen bzw. Auszahlung, keine Vorfälligkeitsentschädigung, keine Steuer- oder Einlagensicherungshinweise und eine Diagramm-Y-Achse, die bei 500.000 $ begann und einen Zuwachs von 3,9 % zu einer Treppe aufbauschte — sind allesamt Varianten desselben Themas: **die Lehrbuchformel wurde implementiert; die Fachlichkeit nicht.**

Die Korrektur: die Kritik zurück ins lokale Modell füttern

Hier ist der Teil, der mich wirklich beeindruckt hat. Ich fügte Claudes Kritik direkt in den Chat des lokalen Modells ein und bat um eine Überarbeitung.

Es wurde nicht defensiv und wiegelte nicht ab. Es erzeugte eine korrigierte Version, die **jeden** gültigen Punkt aufgriff:

- eine Auswahl **APY vs. nomineller Satz** mit der korrekten Umrechnungsmathematik,
- eine Wahl der **Verzinsungsfrequenz** (täglich, monatlich, vierteljährlich, jährlich, bei Fälligkeit),
- einen **Auszahlungsmodus** (reinvestiert/verzinst vs. monatlich ausgezahlt, was linear wächst),
- eine bei **0 $ verankerte Y-Achse**, um die visuelle Verzerrung auszumerzen,
- und eingebettete Hinweise zu **Einlagensicherung und Steuerbehandlung**, wobei Vorfälligkeitsentschädigungen ehrlich als zu bankspezifisch zum Modellieren gekennzeichnet wurden.



Der überarbeitete Rechner — nun ein echtes Festgeld-Werkzeug mit Auswahl für Satztyp und Verzinsungsfrequenz, kein Zinseszins, der ein Festgeld-Etikett trägt.

Sie können beide Versionen selbst öffnen und direkt vergleichen:

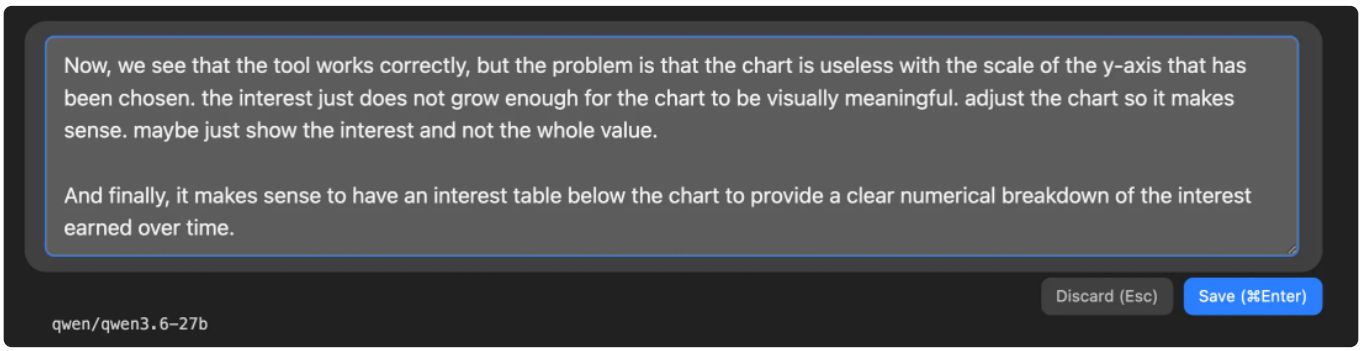
- Der erste, subtil falsche Entwurf: /examples/cd-incorrect.html
- Die korrigierte Version: /examples/cd-fixed.html

Das ist eine saubere Iterationsschleife: lokales Modell entwirft, Cloud-Modell kritisiert, lokales Modell repariert — und die Reparatur saß tatsächlich.

Noch eine Runde: das Diagramm die Wahrheit sagen lassen

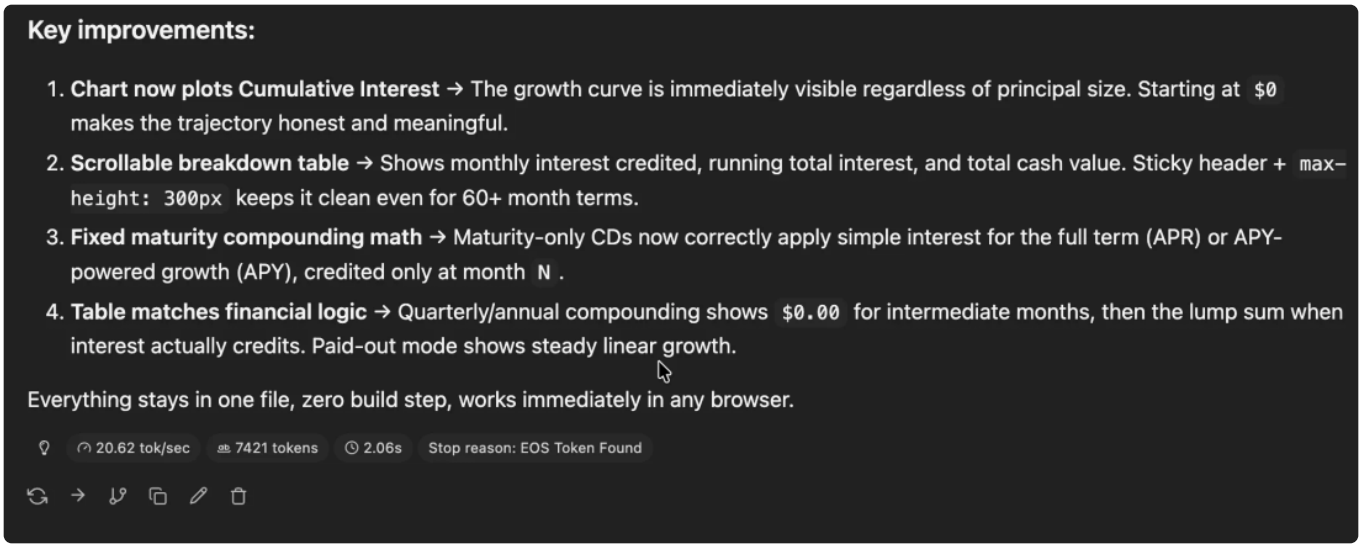
Korrekte Mathematik ließ ein Bedienbarkeitsproblem zurück, das zu zeigen sich lohnt, weil es eine andere *Art* von Korrektur ist.

Mit der nun ehrlich bei 0 $ liegenden Y-Achse bewegt sich der Gesamtsaldo über die Laufzeit kaum — ein Zuwachs von 3,5 % ist auf einer Skala von 500.000 $ eine nahezu flache Linie. Das Diagramm war zugleich korrekt und nutzlos. Also bat ich das Modell, statt des Gesamtsaldos die *verdienten Zinsen* darzustellen und darunter eine numerische Aufschlüsselungstabelle hinzuzufügen.



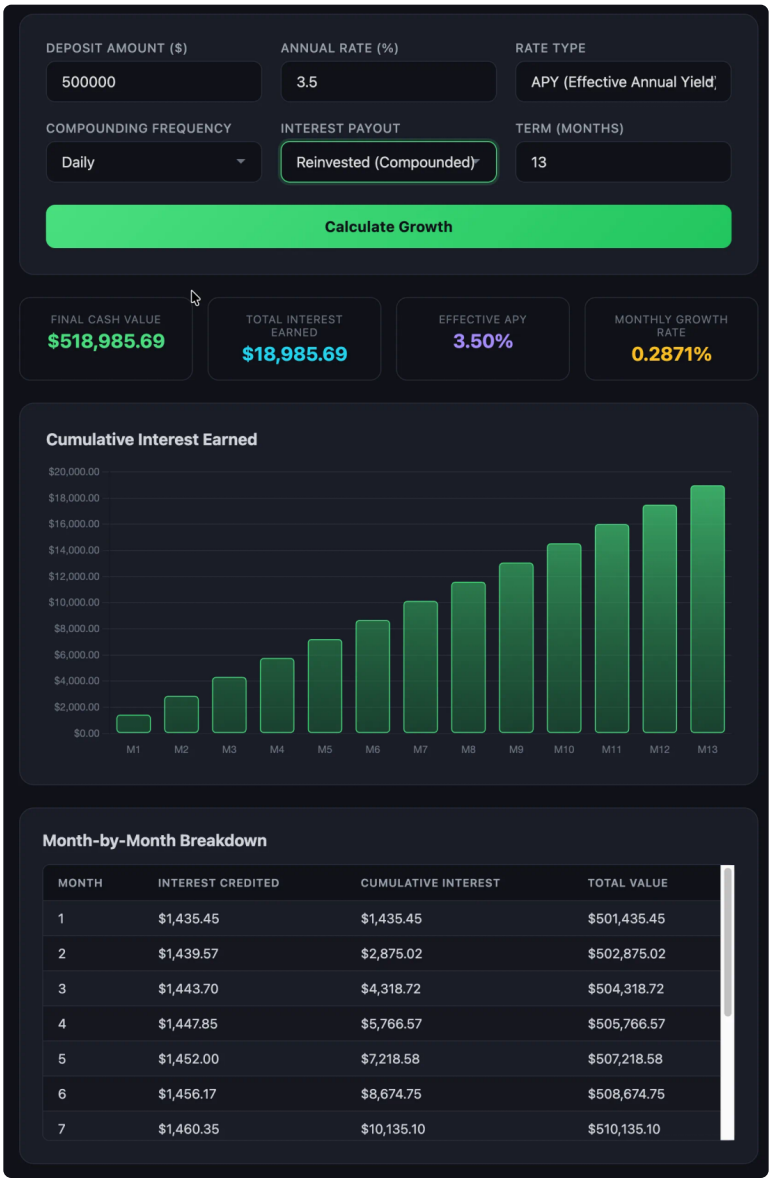
Der Prompt an das lokale Modell, das Diagramm auf verdiente Zinsen umzuskalieren und darunter eine Zins-Aufschlüsselungstabelle hinzuzufügen.

Es erledigte beides in einem Zug. Beim Zusehen bekommt man ein Gefühl für den lokalen Kompromiss: Der Token-Durchsatz lag bei rund **20–30 Token/Sekunde**. Das ist brauchbar, aber es gibt einen Vorbehalt, der ehrlich benannt gehört — unter Dauerlast wird der M5 Max des MacBook Pro warm und drosselt, und die Generierung verlangsamt sich weiter. Ein Desktop wie ein Mac Studio, mit weit mehr thermischem Spielraum, hält seine Geschwindigkeit deutlich besser. Mobile Inferenz hat eine thermische Steuer.



Die finale Antwort generiert lokal — man beachte den Durchsatz von ~20–30 Token/Sekunde auf dem Laptop.

Das Endergebnis ist ein Festgeld-Rechner, der zugleich korrekt und lesbar ist.



Der fertige Rechner: reines Zinsdiagramm, eine monatliche Aufschlüsselungstabelle und ehrliche Skalierung.

Probieren Sie die fertige Version hier aus: /examples/cd-final.html

Ich schickte auch diese zurück an den Cloud-Schiedsrichter. Diesmal verifizierte er die Mathematik unabhängig und gab grünes Licht — das monatliche Wachstum aus einer APY von 3,5 % von 1,035^(1/12) 1 = 0,28709 %, ein Zins im ersten Monat von 1.435,45 $ und ein Endsaldo von ~518.985 $, der die APY-Korrektur korrekt widerspiegelt (etwa 308 $ niedriger als der fehlerhafte Entwurf). Er merkte sogar an, dass der Rechner die Zinsen jeden Monat auf den Cent gerundet gutschreibt und auf dem gerundeten Saldo weiterverzinst — was echte Banken tun, es zählt also *für* das Werkzeug, nicht dagegen.

> **Wo dies immer noch zu kurz greift**
>
> Ehrlichkeit schneidet in beide Richtungen. Das fertige Werkzeug ist ein solider *Wachstums*-Rechner, aber es modelliert immer noch keine Vorfälligkeitsentschädigungen, keine auf Zinsen fällige Steuer (die jährlich anfällt, selbst wenn Sie das Geld nie anrühren) und keine Grenzen der Einlagensicherung. Als Benchmark des Modells ist das in Ordnung. Als Finanzberatung ist es das nicht — und weder das lokale Modell noch der Cloud-Schiedsrichter ersetzen eine Fachperson, wenn echtes Geld im Spiel ist.

Eine erwähnenswerte Entwicklung (14. Juli 2026)

Seit dieser Beitrag erschien, hat sich der weitere Kontext um Qwen verändert. In einem Brief an den Bankenausschuss des US-Senats legte Anthropic offen, was es als die größte je dokumentierte gegnerische

Distillationskampagne bezeichnete — knapp 28,8 Millionen unautorisierte Claude-Austausche über rund sechs Wochen im Jahr 2026, über etwa 25.000 betrügerische Konten, die es Betreibern zuschreibt, die es mit Alibabas Qwen-Labor in Verbindung bringt; es brachte die Sache vor den Kongress statt vor Gericht und merkte an, dass das Verhalten nach geltendem Recht nicht eindeutig illegal sei (CNBC, Tom's Hardware).

Ich erwähne es, weil es zum Hintergrund jedes Modells der Qwen-Abstammung gehört — dieser Beitrag handelt vom Engineering auf Ihrem Schreibtisch, und die politischen Fragen sind ein eigenes Gespräch. Ich habe dieses Experiment außerdem auf einem zweiten lokalen Modell wiederholt; der Vergleich steht in Zwei lokale Coding-Modelle, ein Prompt.

Fazit

Gemessen daran, wo lokale Modelle noch vor einem Jahr standen, ist diese ganze Übung eine gute Nachricht — mit einer angehängten Disziplin.

- **Lokale KI hat die „tägliches Werkzeug"-Linie überschritten.** Ein 27B-Modell, das in ~18 GB passt, 77,2 % auf SWE-bench Verified erreicht und seinen eigenen Code aus einer Kritik heraus repariert, ist keine Spielerei. Es ist wirklich nützlich, es wird immer besser, und am Rand ist es kostenlos.
- **Privatsphäre und Kontrolle sind echte Vorteile.** Nichts in dieser Sitzung verließ meine Maschine. Kein Prompt, kein Code, keine Daten gingen an Dritte — was in dem Moment zählt, in dem Ihre Eingaben sensibel sind.
- **Aber Präzision ist immer noch die Stärke der Cloud — und Verifikation ist Ihre.** Die Lücke war nicht die Arithmetik; es war das Lesen meiner Absicht. Ich sagte „Festgeld", und das lokale Modell gab mir Zinseszins. Ein Spitzen-Cloud-Modell fing mit seinem weit umfangreicheren Training und seinem 1-Millionen-Token-Kontext den fachlichen Fehler, den „sieht-richtig-aus-ist-falsch"-Ausgaben zu verbergen gebaut sind. Hätte ich präziser prompten können? Sicher. Aber wenn ich *Festgeld* sage, ist die Erwartung einer *Festgeld-Berechnung* nicht unangemessen.
- **Der Arbeitsablauf ist die Lektion.** Entwerfen Sie lokal für Geschwindigkeit und Privatsphäre; verifizieren Sie kritische Logik unabhängig — idealerweise mit einem stärkeren Modell, das Ihren Code nie gesehen hat — bevor Sie irgendetwas Finanziellem, Sicherheitsrelevantem oder anderweitig Teuer-zu-Verpatzendem vertrauen.

> Lokale Modelle wurden schnell genug, um Ihr Werkzeug zu entwerfen, und demütig genug, um es zu reparieren. Sie sind noch nicht präzise genug, um die Prüfung zu überspringen. Nutzen Sie sie großzügig — verifizieren Sie die Teile, die Ihnen wehtun können.

Wenn Sie einen lokal-zuerst-KI-Arbeitsablauf für Ihr eigenes Produkt abwägen und einen zweiten Blick darauf möchten, wo Sie ihm vertrauen und wo Sie verifizieren sollten, lassen Sie uns sprechen.

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 —