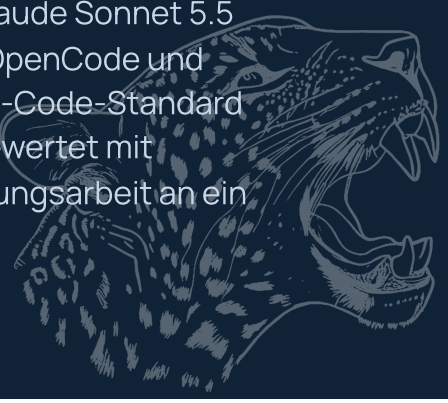
PRAXISTEST · KI-CODING-AGENTEN · OKTOBER 2026

Über fünfmal günstiger – und trotzdem kein Selbstläufer

Zwei KI-Setups, sechs Läufe, eine Entwicklungsaufgabe: Claude Sonnet 5.5 in Claude Code gegen DeepSeek V4.1 Flash, gesteuert von OpenCode und vom DeepSeek-eigenen Agenten dsh. Sonnet lief im Claude-Code-Standard (Effort medium), DeepSeek meist bei Denkaufwand high. Bewertet mit versteckten Tests und blindem Review. Für alle, die Ausführungsarbeit an ein günstiges offenes Modell abgeben wollen.



6 von 6

Läufe mit „Block" im Code-Review. Keiner war ohne Nacharbeit mergefähig.

95 / 95

versteckte Tests bestanden, in allen sechs Läufen. Die Tests unterscheiden die Läufe nicht.

> 5 : 1

Kostenverhältnis Sonnet zu DeepSeek nach Listenpreisen (Sonnet lief im Abo), bis zur ersten grünen Volltestsuite, Wechselkurs angenommen

1,6-mal

so lange brauchte DeepSeek bis zur ersten grünen Volltestsuite (46,4 gegen 28,9 min), mit gut 2,5-mal so vielen Arbeitsschritten

Kurz gesagt

- **Qualität:** Im Review hatte DeepSeek weniger schwerwiegende Befunde als Sonnet (zwei bei b, vier bei a), aber den einzigen Blocker (Lauf c). Die Zahl der Befunde sagt nichts über ihre Schwere.
- **Preis und Zeit:** Nach Listenpreisen kostete DeepSeek weniger als ein Fünftel, brauchte aber 1,6-mal so lange, vor allem durch Warten auf Testläufe.
- **Harness** (Steuerungssoftware um das Modell): Mit OpenCode und dsh (b bis f) kamen 94 bis 99 % der Eingabe aus dem Cache, mit Claude Code nur rund 14 % (g), geschätzt 13 bis 19 % (h), früher höchstens 60–65 %.
- **Denkaufwand** (Effort): Sonnet lief im Claude-Code-Standard (medium) mit laut Transkript nie aufgerufenem Advisor (Opus 5.5). DeepSeek meist bei high, f bei medium, c bei xhigh. Bei gleicher Stufe (medium) kostete DeepSeek im Gesamtlauf nach Listenpreis rund 6- bis 7-mal weniger, allerdings ohne grüne Volltestsuite und bei rund sechsfacher Zeit bis zum letzten Commit; die Skalen sind verschieden.

> Zwei Anfragen ohne Cache-Treffer kosteten ein Viertel eines ganzen dsh-Laufs (e).

VORWEG

Warum ich das getestet habe

Ich arbeite viel und gerne mit Claude. Trotzdem wollte ich wissen, ob ich einen kostengünstigen Agenten mit einem in Europa gehosteten DeepSeek-Modell in meinen Entwicklungsprozess einbauen kann. Konkret ging es um die Ausführungsphase: Ein fertiger Plan wird umgesetzt, testgetrieben und mit atomaren Commits. Planung und Review bleiben bei Claude und bei unabhängigen Gutachtern.

Wie ich KI-Agenten in meine Entwicklung einbinde, beschreibe ich im Artikel zu meinem dev-process: sg-it.info/artikel/entwicklungsprozess-mit-ki-agenten, die Vorlage liegt auf github.com/Crashman1983/dev-process.

Zur Einordnung. Meine Perspektive ist nicht neutral. Ich bin Claude-Nutzer, und auch die DeepSeek-Läufe hat ein Claude-Agent gestartet und ausgewertet. Das Review lief durch ein drittes Modell (Codex), die Tests schrieb ein unabhängiger Agent. Trotzdem können sich Gewohnheiten und Vorannahmen in den Aufbau eingeschlichen haben. Deshalb steht in diesem Bericht, was gemessen wurde und was nicht.

Begriffe in Kürze. Lauf: ein kompletter Durchgang der Aufgabe durch ein Modell in einem Harness; ausgewertet sind a bis f, g und h wurden vorzeitig beendet. **Harness:** die Steuerungssoftware, die ein Modell zum Agenten macht (Claude Code, OpenCode und `dsh`, der Agent von DeepSeek). **Advisor:** ein zweites Modell (Opus 5.5), das Claude Code bei Bedarf befragen kann; nur bei Sonnet konfiguriert, laut Transkript nicht aufgerufen. **Push-Gate:** die projekteigene Prüfstrecke vor dem Push. **Arbeitsschritt** (Turn): ein Agentendurchgang mit Modellaufruf. **Cache-Quote:** Anteil der Eingabe, der aus dem Zwischenspeicher kommt und deutlich weniger kostet. **Block:** Review-Urteil „nicht ohne Nacharbeit mergefähig“. **Gegenprüfer:** ein zweiter Review-Lauf (Codex), der jeden Befund selbst nachzustellen versucht. **Regelblock:** ein Textblock mit fünf Regeln, der dem Auftrag vorangestellt wird. Er soll Arbeitsschritte und Wartezeit sparen, denn DeepSeek arbeitete in kleinen Schritten und startete Tests oft. Die Regeln: Lesezugriffe bündeln, Änderungen je Datei bündeln, Schreiben, Test und Commit verketten, gezielt testen, den Volltest nur einmal und im Hintergrund starten.

Die Aufgabe

Ein KI-Agent arbeitet in einer abgeschotteten Umgebung (Sandbox) und soll die Originaldokumente zu seiner Sitzung sehen können, etwa Excel-, PDF- oder Präsentationsdateien. Das Feature legt dafür vor jedem Zug eine schreibgeschützte Kopie in einen Ordner, den die Sandbox lesen kann. Vorher sah der Agent nur Dateien bis 500 KiB, größere fehlten ohne Hinweis. Die Regeln: Eine Datei wird übernommen, wenn sie höchstens 50 MiB groß ist, weniger als 2.000 Dateien übernommen sind und alle Kopien zusammen 256 MiB nicht überschreiten. Jede Datei landet in genau einer von zwei Listen, „übernommen“ oder „fehlt“ mit Grund, und der Agent bekommt einen Hinweis, was fehlt. Hinein dürfen nur Dateien im erlaubten Bereich, keine Links (Symlinks).

Es war ein Feature über Backend und Oberfläche eines selbst betriebenen Python/TypeScript-Projekts, mittleres Risiko, mit sechs Teilaufgaben, testgetrieben und mit atomaren Commits. Alle Läufe starteten auf demselben Commit mit demselben Plan und denselben versteckten Tests, ohne einander zu sehen. d, e, g und h bekamen zusätzlich den Regelblock in den Auftrag (die Läufe stehen auf der nächsten Seite).

1 · DER VERSUCH

Eine Aufgabe, sechs Läufe, drei Prüfebenen

Die Frage war praktisch: Kann DeepSeek die Ausführungsphase übernehmen, die sonst Claude Sonnet 5.5 erledigt? Und welchen Anteil daran hat der Harness, die Steuerungssoftware um das Modell?

Die Bewertung

- **95 versteckte Akzeptanztests**, vorab von einem unabhängigen Agenten geschrieben.
- **Blindes Code-Review** durch ein drittes Modell (Codex gpt-6.1-sol, Effort high); jeder Befund geht an den Gegenprüfer.

Wie blind war das Review?

Wer die versteckten Tests schrieb (ein unabhängiger Agent, vorab) und wer den Code prüfte (Codex), kannte die Zuordnung der Läufe zu Modell und Harness nicht. Der Reviewer bekam je Lauf einen neutralen Branch ohne Hinweis auf die Herkunft, denselben Prompt und dieselben Einstellungen. Die Versuchsleitung, ein Claude-Agent, der die Läufe startete und auswertete, kannte die Zuordnung von Anfang an. Ob die Kenntnis der Zuordnung die Läufe beeinflusst hat, wurde nicht gemessen. Doppelblind ist der Aufbau also nicht. Die Reviews für a und b entstanden zuerst; ihre Befunde standen fest, bevor die Zuordnung aufgedeckt wurde, belegt durch einen versiegelten Schlüssel mit Prüfsumme. Für c bis f gab es keinen versiegelten Schlüssel, ihre Reviews liefen später. Die Reviews liefen jeweils parallel, nicht in zufälliger Reihenfolge.

Die Läufe

LAUF	MODELL	HARNESS	EFFORT / EINSTELLUNG
a	Claude Sonnet 5.5	Claude Code (Advisor Opus 5.5)	Standard: medium (nicht explizit gesetzt), Denken aktiv; Advisor laut Transkript nicht aufgerufen
b	DeepSeek V4.1 Flash	OpenCode 1.18	Denkaufwand high
c	DeepSeek V4.1 Flash	OpenCode 1.18	Denkaufwand xhigh
d	DeepSeek V4.1 Flash	OpenCode 1.18	high + Regelblock
e	DeepSeek V4.1 Flash	`dsh` (Alpha)	Regelblock, eigener Klon; Effort nicht explizit gesetzt, Wert nicht ermittelt (Modell dachte)
f	DeepSeek V4.1 Flash	OpenCode 1.18	Denkaufwand medium, ohne Regelblock, sonst wie b

Die Läufe g und h (Claude Code mit DeepSeek, Effort high, Regelblock) wurden vorzeitig beendet; sie stehen im Exkurs, nicht in der Auswertung. DeepSeek lief über Melious, einen Router über europäische Anbieter.

> **Verglichen werden Setups, nicht nur Modelle.** Sonnet lief in Claude Code, DeepSeek in OpenCode und `dsh`. Einen Advisor hatte nur Sonnet (konfiguriert, laut Transkript nie aufgerufen), und durch das Push-Gate lief nur Sonnet. Ein Unterschied im Ergebnis kann also am Modell liegen oder am Rest des Setups.

2 · QUALITÄT

Tests und Review sagen Verschiedenes

Alle sechs Läufe bestanden alle 95 versteckten Tests; die Testquote trennt sie nicht. Das Code-Review fand dagegen in jedem Lauf nachstellbare Lücken, alle sechs bekamen **„Block“**.

LAUF	TESTS	REVIEW-BEFUNDE (GEGENPRÜFUNG)
a Sonnet, Claude Code	95/95	4 schwerwiegende, alle bestätigt
b DeepSeek, high	95/95	2 schwerwiegende bestätigt, 1 geringfügiger teilweise
c DeepSeek, xhigh	95/95	**1 Blocker** und 1 schwerwiegender, beide bestätigt
d DeepSeek, Regelblock	95/95	2 schwerwiegende bestätigt, 1 geringfügiger teilweise
e DeepSeek, `dsh`	95/95	2 schwerwiegende, beide bestätigt
f DeepSeek, medium	95/95	2 schwerwiegende (1 bestätigt, 1 teilweise), 2 geringfügige (1 bestätigt, 1 teilweise)

Was das Review fand und die Tests nicht

Das Review fand Lücken, die für den Zweck des Features zählen; zwei tauchten mehrfach auf, unabhängig von Modell und Harness: die Gesamtgröße bei a, b, d und e, die Lesefehler bei a, b, c, e und f (bei d verwandt: eine zwischen Größenprüfung und Lesen übergroß gewordene Datei fehlt ohne Hinweis).

- **Gesamtgröße.** Zusammen dürfen die Kopien höchstens 256 MiB haben (schützt Speicher und Laufzeit). Die betroffenen Läufe (a, b, d, e) prüfen das vorab mit den gemeldeten Dateigrößen; wächst eine Datei danach, wird mehr kopiert als erlaubt (nachgestellt: 7 statt 5 Bytes). Der Plan verlangte, die gelesenen Bytes zu zählen; die Basis tat das, die Läufe verloren es beim Vorziehen der Prüfung.
- **Lesefehler.** Eine Datei ohne Leserechte oder mit Lesefehler steht in keiner der beiden Listen; der Agent erfährt nicht, dass ein Original fehlt. Der Plan sagte dazu nichts; die Basis brach bei Lesefehlern ab, die betroffenen Läufe (a, b, c, e, f) fangen den Fehler ab und überspringen die Datei, ohne es zu melden.

Sonnet (a) hatte zudem ein Dateihandle-Leck und einen Lesefehler, nach dem die Kopie leer bleibt. Bei f landen Dateinamen unverändert in der Nachricht an den Agenten, wie der Plan es vorschreibt; ein Name mit Zeilenumbrüchen und Anweisung kann dort wie Anweisungstext wirken (teilweise bestätigt, andere Läufe nicht geprüft). Bei c fand das Review den einzigen **Blocker**: Ersetzt jemand einen Ordner im Pfad durch einen Link nach außen, greift die Symlink-Prüfung nicht, sie prüfte nur die letzte Pfadkomponente. Beim Nachstellen las der Agent 614.400 Bytes von außerhalb des erlaubten Bereichs. Die Lücke stammt aus der Basis: Die Läufe a bis e enthalten dieselbe Öffnungslogik (f nicht geprüft), gemeldet hat sie nur das Review von c. Warum, ist nicht ermittelt; ob sie in den anderen Läufen ausnutzbar ist, wurde nicht gemessen.

Die Tests prüfen feste Fälle: zu große Datei, zu viele Dateien, ein einfacher Link; Zeitabläufe fehlen. Dass eine Datei nach der Größenprüfung wächst oder ein Ordner ausgetauscht wird, sind Wettläufe, die sich ohne Eingriff in die Interna kaum herstellen lassen; eine Datei ohne Leserechte kam nicht vor. Tests sagen nur etwas über vorher bedachte Fälle; das Review fand die unbedachten, eines ersetzt das andere nicht.

> **Was das heißt.** a hatte vier bestätigte schwerwiegende Befunde, b bis e je zwei, f einen bestätigten und einen teilweise bestätigten, c darunter den Blocker. Die Lesefehler-Lücke liegt vor allem an uns: Plan ohne Reaktion, kein Test.

3 · ZEIT

1,6-mal so lange, vor allem durch Warten

Ein fairer Vergleich braucht einen gemeinsamen Messpunkt: Sonnet lief nach dem letzten Code-Commit noch durch das Push-Gate, DeepSeek pushte nicht. Verglichen wird deshalb die Zeit bis zur **ersten grünen Volltestsuite**.

GRÖSSE	A: SONNET (MEDIUM)	B: DEEPSEEK (HIGH)	VERHÄLTNIS
Zeit bis erste grüne Volltestsuite	28,9 min	46,4 min	1,6 : 1
Arbeitsschritte bis dahin	62	158	2,5 : 1

Zwei weitere Messpunkte liefert der Git-Verlauf: B, der letzte Code-Commit, und C, der letzte Commit mit Registry-Eintrag. Bis B war Sonnet mit 10,6 Minuten am schnellsten, dahinter c (xhigh) mit 12,0 Minuten; seinen Registry-Commit setzte c erst nach 60,2 Minuten, nach dem vollen Testlauf. Bis C liegt b zu a bei 3,5 : 1; das überzeichnet, weil die Volltestsuite bei Sonnet nach diesem Commit lag, bei DeepSeek davor.

In den OpenCode-Läufen gingen 35 bis 46 Minuten des Gesamtlaufs in Wartezeit auf Werkzeuge, vor allem auf Testläufe; in unserem Headless-Aufbau mit OpenCode 1.18 liefen Befehle nicht im Hintergrund, in Claude Code schon. DeepSeek wiederholte die Volltestsuite nach der ersten grünen Runde mehrfach; das zählt in den 46,4 Minuten nicht mit, die Teststrategie war also nicht dieselbe.

Lauf f (medium, ohne Regelblock) hat keinen Messpunkt A: Sein einziger Volllauf endete rot, mit zwei lastabhängigen Tests und fünf Typprüfungsfehlern in unberührten Modulen, beides schon in der Basis. f brauchte 94,5 Minuten bis zum letzten Code-Commit und 95,6 bis zum letzten Commit, in 190 Arbeitsschritten; medium sparte keine Zeit. Woran das liegt (190 statt 163 Schritte, parallel laufende Worker), ist nicht isoliert; die Laufzeiten sind nur eingeschränkt vergleichbar.

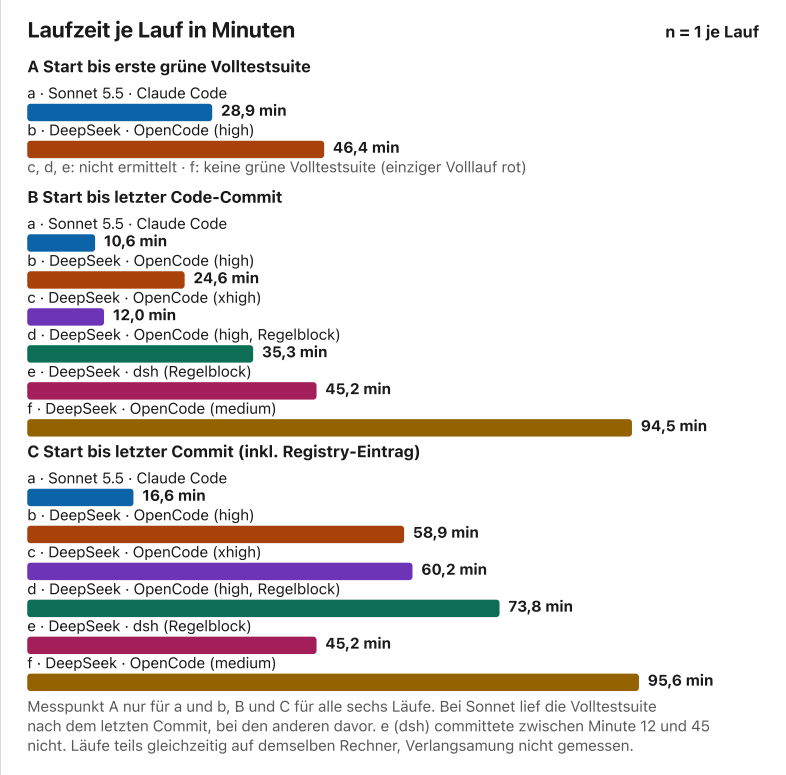


Abbildung 1: Messpunkt A (erste grüne Volltestsuite) nur für a und b; f hatte keine grüne Volltestsuite. B: letzter Code-Commit, C: letzter Commit inkl. Registry-Eintrag. n = 1 je Lauf.

4 · KOSTEN

Über fünfmal günstiger, nach Listenpreis

Die Kosten sind aus den gemessenen Token und den Listenpreisen berechnet. Für Sonnet 5.5 gelten die Anthropic-API-Listenpreise in Dollar (je Million Token: Eingabe 2 $, Cache-Schreiben 4 $, Cache-Lesen 0,20 $, Ausgabe 10 $); der Lauf selbst fand im Abo statt. DeepSeek V4.1 Flash über Melious rechnet in Euro ab (Eingabe 0,20 €, Cache-Lesen 0,01 €, Ausgabe 1,00 €). Melious-Preise zuzüglich Umsatzsteuer; die Währungen bleiben getrennt. Quellen: die Preisseiten beider Anbieter.

BIS ERSTE GRÜNE VOLLTESTSUITE	A: SONNET (MEDIUM, ADVISOR NICHT AUFGERUFEN)	B: DEEPSEEK (HIGH)
frische Eingabe (Token)	128	683.528
Cache-Schreiben (Token)	192.503	–
Cache-Lesen (Token)	10.536.499	28.313.088
Ausgabe (Token, b inkl. Reasoning)	39.125	83.857
Kosten laut Preisliste	**3,27 $**	**0,50 €**

Das ergibt rund **6,5 : 1** bei 1 € = 1 $ und rund **5,4 : 1** bei 1 € = 1,20 $; der Kurs ist eine Annahme. Die 3,27 $ enthalten keine Advisor-Kosten: Opus 5.5 war konfiguriert, im Transkript stehen keine Aufrufe. Über den Gesamtlauf (a: 138 frische Eingabe, 194.187 Cache-Schreiben, 11.412.473 Cache-Lesen, 40.601 Ausgabe) kostet Sonnet 3,47 $, DeepSeek (b) 0,57 €: rund 6,1 : 1 bzw. 5,1 : 1.

Bei gleichem Effort. f lief wie a bei medium. Im Gesamtlauf kostet a 3,47 $, f nach demselben Melious-Tarif 0,48 € (ohne Reasoning-Token 0,43 €): rund **7,2 : 1** bei 1 € = 1 $ und rund **6,0 : 1** bei 1 € = 1,20 $. f erreichte keine grüne Volltestsuite und brauchte 95,6 statt 16,6 min bis zum letzten Commit; die Effort-Skalen sind verschieden, und der Gesamtlauf von a enthält das Push-Gate (rund 24 min), f hat keins.

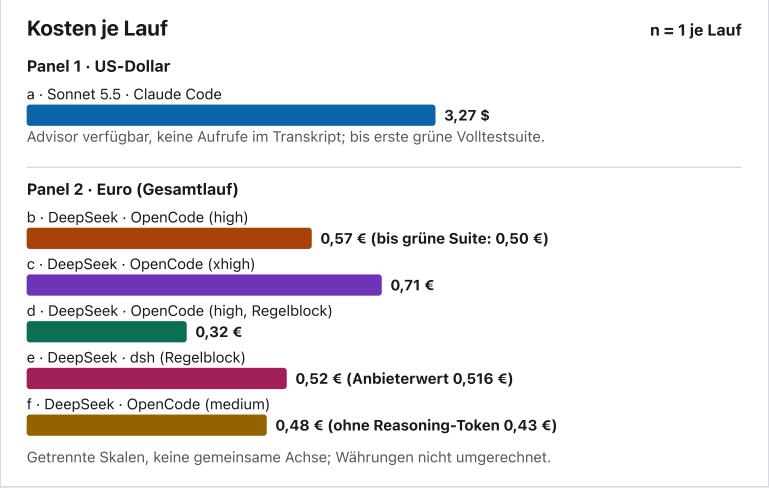


Abbildung 2: Zwei Panels mit getrennten Skalen. Panel 1 (a, Dollar) bis zur ersten grünen Volltestsuite, Panel 2 (b bis f, Euro) für den Gesamtlauf; e Anbieterwert, b, c, d und f aus dem Tarif berechnet. f ohne Reasoning-Token: 0,43 €. n = 1 je Lauf.

Was in der Rechnung fehlt

- Das Abonnement, in dem Lauf a lief; gerechnet ist mit API-Listenpreisen.
- Review, Tests, die Zeit des Menschen und die Nacharbeit nach dem Review (alle sechs Läufe: „Block“).
- Die Preise können sich ändern.

Die Cache-Quote bestimmt die Rechnung

Jeder Arbeitsschritt schickt den gesamten bisherigen Verlauf erneut an das Modell; die Kosten wachsen deshalb etwa quadratisch mit der Länge des Laufs. Gecachte Eingabe kostet bei Melious ein Zwanzigstel der frischen (0,01 gegen 0,20 € je Million Token). Darum wiegt die **Cache-Quote** schwerer als der Listenpreis.

KOMBINATION	CACHE-QUOTE	BEOBACHTUNG
OpenCode + DeepSeek	94–99 %	Läufe b, c, d, f (96,9 / 93,9 / 97,9 / 98,7 %)
`dsh` + DeepSeek (e)	96,4 %	Alpha; ohne zwei Ausfälle 99,0 %
Claude Code + Claude-Modelle	98,3 %	Lauf a, Gesamtlauf
Claude Code + DeepSeek	13–19 %	Lauf g ca. 14 %, Lauf h 13–19 % (geschätzt), ein früherer Test höchstens 60–65 %; alle vorzeitig beendet, siehe Exkurs

Dasselbe Modell erreicht in verschiedenen Harnessen also sehr unterschiedliche Cache-Quoten. Was dabei zu sehen war, steht im Exkurs zu den Läufen g und h mit Claude Code.

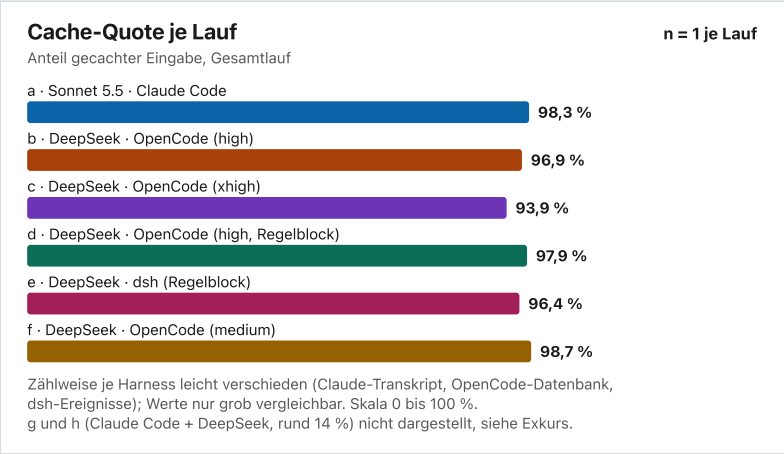


Abbildung 3: Cache-Quote über den Gesamtlauf, Token-gewichtet. Die Zählweisen unterscheiden sich leicht (Claude-Transkript, OpenCode-Datenbank, `dsh`-Ereignisse), die Werte sind nur grob vergleichbar. n = 1 je Lauf.

Zwei Anfragen, ein Viertel der Kosten

Im `dsh`-Lauf hatten zwei einzelne Anfragen mitten im Lauf **null Cache-Treffer**, bei rund 314.000 und 319.000 Token Kontext. Zusammen kosteten sie 0,127 €, ein Viertel der Laufkosten (0,516 €). Ohne sie läge derselbe Lauf bei 99,0 % Cache-Quote und 0,39 €.

Woran das lag, ist offen. Der Fall lässt sich bislang weder dem Harness noch dem Anbieter zuordnen. Ein abgelaufener Cache oder ein Backend-Wechsel beim Anbieter wären denkbar, belegt ist keines von beiden; der Anfang des Prompts blieb laut Ereignisstrom unverändert. Ähnliche Berichte aus anderen Aufbauten gibt es öffentlich, sie belegen aber nicht dieselbe Ursache.

6 · EXKURS

Claude Code mit DeepSeek: in unserem Aufbau derzeit keine gute Idee

Zwei Läufe mit Claude Code als Harness, angebunden über den Anthropic-kompatiblen Endpunkt von Melious an DeepSeek V4.1 Flash (Effort high, Regelblock), wurden vorzeitig beendet und fehlen in der Auswertung. **Lauf g** stoppten wir nach rund 44 Minuten, 95 Antworten und fünf Feature-Commits, weil die Cache-Messung auffiel, nicht wegen eines Fehlers. **Lauf h** lief mit eigenem API-Schlüssel hinter einem Mitschnitt-Proxy, der nur Struktur und Prüfsummen der Anfragen festhielt, keine Inhalte. Ein Wächter brach ihn nach 85,8 Minuten planmäßig ab: Die Cache-Quote lag in zwei aufeinanderfolgenden Blöcken von 20 Anfragen unter 60 %.

BEOBACHTUNG	CACHE-QUOTE	DETAILS
Früherer Test, Claude Code + DeepSeek	höchstens 60–65 %	ließ sich nicht beheben
Lauf g (94 Anfragen laut Nutzungsdatei, 95 Antworten im Transkript)	ca. 14 %	Einzelwerte überwiegend 7–19 %
Lauf h (Mitschnitt, 60 Hauptanfragen)	13–19 % je 20er-Block, geschätzt	Block 1 (Aufwärmphase) 18,8 %, Block 2 14,2 %, Block 3 13,3 %; Einzelanfragen 6,4–24,7 %; dazu 10 Nebenanfragen, zweimal HTTP 400
OpenCode und `dsh` (b bis f)	94–99 %	derselbe Anbieter, dasselbe Modell, OpenAI-kompatible Schnittstelle

Was der Mitschnitt zeigt

Bei h blieb der Systemteil über alle 60 Hauptanfragen unverändert. Die Werkzeugliste wurde nur einmal erweitert, bei Anfrage 4 von 23 auf 56 Werkzeuge (MCP-Werkzeuge nach dem Start), danach blieb sie konstant. Der Verlauf wuchs sauber am Ende: In 58 von 59 Übergängen kamen je eine Nutzer-, eine Assistenten- und eine system-Nachricht hinzu, bisherige Nachrichten blieben unverändert; der gemeinsame Präfix mit der Vorgängeranfrage lag im Median bei 83 %. Je Anfrage setzte Claude Code drei cache_control-Marker, zwei fest im Systemteil und einen auf der letzten Nachricht. Gecacht wurde trotzdem nur der feste Anfang: cache_read blieb bei etwa 22.000 bis 24.000 Token, während die Anfragen von rund 76.000 auf 169.000 Token wuchsen; ein annähernd voller Treffer kam nie vor. Dazu stand der Lauf rund 40 Minuten still: Eine Anfrage mit rund 150.000 Token lieferte nach 190 Sekunden keine Ausgabe, Claude Code meldete, das Modell habe mitten im Stream aufgehört zu senden; einmal wurde von Hand „Bitte fortsetzen“ eingegeben. h hinterließ drei Commits; auf diesem Stand waren 84 von 95 versteckten Tests grün, ein Teil der Verdrahtung fehlte. Ein Review gab es nicht, der Lauf war unfertig.

Die öffentliche Hypothese trifft hier nicht zu

DeepSeek cacht nach Präfix: Der Cache greift nur, solange der Anfang der Anfrage Byte für Byte gleich bleibt. Öffentliche Berichte erklären schlechte Quoten mit einem wechselnden Anfang: Ein Projekt (permafrost) misst für Claude Code gegen DeepSeek rund 90 % Cache bei stabiler Werkzeugliste und rund 33 %, wenn sich die Liste ändert, etwa durch nachgeladene Werkzeuge; ein Issue im Projekt openhuman beschreibt Cache-Misses bei DeepSeek V4 Flash durch wechselnde Prompt-Anfänge. Bei Lauf h blieb der Anfang stabil, die Quote trotzdem niedrig. Derselbe Anbieter und dasselbe Modell erreichen über die OpenAI-kompatible Schnittstelle (OpenCode, `dsh`) 94 bis 99 %. Naheliegend ist der Zugangsweg, also die Kombination aus Claude Code, Anthropic-kompatiblem Endpunkt und dessen Nachrichtenform; belegt ist das nicht. Der Vergleich ändert Harness und Endpunkt zugleich.

Was offen ist

Welcher Teil des Zugangswegs den Cache bricht, ist nicht ermittelt: die Übersetzung der Cache-Markierungen, die system-Nachrichten im Verlauf oder die Blockgröße des Caches. Belegen ließe es nur ein Gegenversuch mit derselben Anfragefolge ohne cache_control oder ohne system-Nachrichten. Die Quoten und Kosten bei h sind geschätzt, nicht gemessen: In unserem Mitschnitt meldete Melious im Streaming keine Eingabe-Token, der Proxy schätzt sie aus der Größe des Anfragetexts (Bytes durch 3,5, abzüglich cache_read); gegen die Nutzungsdatei ist das nicht validiert. Bis zum Abbruch kostete h geschätzt mindestens 1,45 €, rund 0,02 € je Anfrage, davon 95 % für nicht gecachte Eingabe.

> **Was wir daraus mitnehmen.** Stand Oktober 2026 und für unseren Aufbau: Wer DeepSeek V4.1 betreibt, fährt mit OpenCode oder `dsh` besser, dort lag die Cache-Quote bei 94 bis 99 %. Wer Claude Code über einen Anthropic-kompatiblen Endpunkt mit DeepSeek nutzen will, sollte vorher die Cache-Quote beim Anbieter messen und mit einem kleinen Auftrag testen. Gegen Claude Code mit Claude-Modellen sagt das nichts: Dort erreichte es 98,3 %.

Quellen: https://github.com/jianzhichun/permafrost, https://github.com/tinyhumansai/openhuman/issues/6618 (abgerufen am 6. Oktober 2026).

7 · DEEPSEEKS EIGENER AGENT

dsh: der Agent von DeepSeek im Test

Neben OpenCode lief ein zweiter Harness mit DeepSeek: `dsh`, der Agent von DeepSeek selbst. Er ist Open Source (MIT), in TypeScript geschrieben, hat eine Plugin-Architektur und ist seit August 2026 als Entwicklervorschau (Alpha) verfügbar. Ein erster Lauf scheiterte an der Sandbox und wurde abgebrochen, der zweite (e) lief durch.

LAUF E (`DSH`, REGELBLOCK, EIGENER KLON)	WERT
Versteckte Tests	95 / 95
Review	Block, 2 schwerwiegende (bestätigt)
Zeit bis zum letzten Commit (zwischen Minute 12 und 45 kein Commit)	45,2 min
Arbeitsschritte	100 (`dsh` zählt Schritte)
Cache-Quote	96,4 % (99,0 % ohne zwei Ausfälle)
Kosten (Anbieterwert)	0,516 €

Was gut läuft

- **Headless-Profil** mit einem Auftrag je Aufruf, gut für einen steuernden Agenten.
- **Ereignisstrom** (`--json`): Werkzeugaufrufe, Zustandswechsel, Token je Schritt. Auswertbar ohne Datenbank und ohne Kontextverbrauch beim Steuernden.
- **Eigener Endpunkt** (OpenAI-kompatibel) per Konfiguration, parallele Werkzeugaufrufe im selben Schritt.
- **Sandbox** standardmäßig aktiv: Schreiben nur im Arbeitsverzeichnis. Der Agent versuchte nach einem Fehlschlag außerhalb zu schreiben, die Sandbox verweigerte es.
- Die Weboberfläche bindet nur an Loopback.

Was noch fehlt

- **Worktrees:** `git commit` scheitert, weil das Git-Verzeichnis außerhalb des Arbeitsverzeichnisses liegt. Zusätzliche Schreibpfade sind nicht vorgesehen. Abhilfe: ein eigener lokaler Klon.
- **Keine Nachricht** an einen laufenden Lauf. Fortsetzen nur mit neuem Aufruf (`--session-id`), Abbrechen durch Beenden des Prozesses.
- **Alpha:** Breaking Changes möglich. Das Plugin-System blieb ungetestet, Hintergrundjobs wurden genutzt, aber nicht im Detail geprüft.
- **Zwei Cache-Ausfälle** mit ungeklärter Ursache (Kapitel 5).

Bewertung. Bei Tests, Review und Cache-Quote liegt e gleichauf mit den OpenCode-Läufen. Weiter reicht ein einzelner Lauf nicht: e unterscheidet sich von b im Auftrag (Regelblock) und in der Arbeitskopie (Klon), die Zeit ist mit a und b nicht vergleichbar, und ob `dsh` besser oder schlechter arbeitet als OpenCode, wurde nicht gemessen. Wer es ausprobieren will, sollte Sandbox und Alpha-Status einplanen.

8 · IM ALLTAG

Was die drei Harnesse im Alltag unterscheidet

Die Aufträge verteilte ein steuernder Agent, der Zwischenstände las und Rückfragen entschied. Der Aufbau ist selbst gebaut und auf einen Einzelbetreiber zugeschnitten; wer am Terminal arbeitet, gewichtet die Punkte anders.

FÄHIGKEIT	CLAUDE CODE + CLAUDE	OPENCODE + DEEPSEEK	DSH (ALPHA)
Nachricht an laufende Sitzung	jederzeit	erst nach dem Auftrag	nein
Statusanzeige außerhalb des Terminals	Claude-App (Remote Control)	selbst gebaut	selbst gebaut
Lange Tests im Hintergrund	ja, mit Benachrichtigung	nein (im Headless-Aufbau), blockieren	als Werkzeug vorhanden
Advisor-Modell	ja (Opus 5.5)	nein	nein
Ereignisstrom mit Token je Schritt	Transkript	Datenbank	`--json`, ohne Datenbank
Sandbox (Schreiben nur im Arbeitsverzeichnis)	nicht betrachtet	nicht betrachtet	ja, standardmäßig
Reife	Produkt	stabil; braucht in unserem Headless-Aufbau stdin-Umleitung, sonst hängt der Lauf	Entwicklervorschau

Was im Aufbau funktionierte

- DeepSeek in der **Ausführungsphase**: fertiger Plan, testgetriebene Umsetzung, atomare Commits, mittleres Risiko.
- Die Planung lag bei Claude, das Review bei einem unabhängigen Gutachter (Codex) mit Gegenprüfer. Entscheidungen liefen über Einträge im Plan statt über Zurufe.
- `dsh` liefert einen Ereignisstrom mit Token je Schritt, auswertbar ohne Datenbank.

Was zusätzlich nötig war

- Statusanzeige und Rückfragen für DeepSeek-Läufe mussten selbst gebaut werden.
- Befehle, Skills und Hooks aus Claude Code gingen als Text in den Auftrag und griffen dort weniger verlässlich.
- `dsh`: Schreiben nur im Arbeitsverzeichnis (Sandbox); im Git-Worktree scheitert deshalb `git commit`, Abhilfe war ein eigener Klon.

Denkaufwand und Regelblock

Am **Denkaufwand** liegt die hohe Zahl der Arbeitsschritte vermutlich nicht: In einem eigenen Test (DeepSeek V4.1 Flash über OpenCode, Kopfrechenaufgabe aus der Zahlentheorie, n = 3 je Stufe, alle zwölf Antworten richtig) stiegen die Reasoning-Token im Mittel nur von 2.470 (none) auf 3.195 (high) und 3.712 (xhigh), bei max sanken sie auf 2.859. Auch im Lauf brachte xhigh (Lauf c) keinen klaren Gewinn.

Zum **Regelblock** gibt es nur einen Hinweis: Lauf d brauchte im Gesamtlauf 107 statt 163 Arbeitsschritte und kostete 0,32 € statt 0,57 €, brauchte aber länger bis zum Code-Commit (35,3 gegen 24,6 min bei b) und bis zum letzten Commit (73,8 gegen 58,9 min). Eine Zeitersparnis zeigt sich nicht; es bleibt eine Einzelmessung, überlagert vom Warteverhalten. Bei Läufen mit Claude-Modellen wurde er nicht untersucht.

Was belastbar ist und was nicht

Das lässt sich sagen

- DeepSeek bestand dieselben Tests wie Sonnet und hatte im Review weniger schwerwiegende Befunde (2 bei b gegen 4 bei a), aber bei c den einzigen Blocker. Alle sechs bekamen „Block“.
- Nach Listenpreisen kostete b bis zur ersten grünen Volltestsuite unter einem Fünftel von a, f bei gleichem Effort (medium) im Gesamtlauf rund 6- bis 7-mal weniger, je nach Kurs.
- Dafür brauchte b 1,6-mal so lange und gut 2,5-mal so viele Arbeitsschritte; f brauchte bis zum letzten Commit 95,6 statt 16,6 min und knapp dreimal so viele Arbeitsschritte (190 gegen 66).
- Der Harness bestimmte vermutlich Cache-Quote und Kosten.

Das lässt sich nicht sagen

- Dass DeepSeek generell so gut ist wie Sonnet.
- Dass die Verhältnisse bei einer Wiederholung gleich ausfallen.
- Dass die Qualität auf andere Aufgaben, Stacks oder höhere Risiken übertragbar ist.
- Dass der Unterschied am Modell und nicht am Harness liegt.
- Dass die Ergebnisse für andere Effort-Stufen gelten (Sonnet: medium; DeepSeek: high, ein Lauf medium, ein Lauf xhigh).

Grenzen der Bewertbarkeit

- **n = 1 je Lauf.** Keine Streuung, keine Signifikanz. Eine Aufgabe, ein Projekt, eine Stack-Kombination.
- **Setups, nicht nur Modelle.** Mit dem Modell wechselten Harness, Advisor und Push-Gate. Der Advisor (Opus 5.5) war nur bei Sonnet konfiguriert und laut Transkript nicht aufgerufen; ein Einfluss auf das Ergebnis ist nicht belegt.
- **Effort nicht gleichartig.** Sonnet lief im Claude-Code-Standard (medium, nicht explizit gesetzt, Denken aktiv), e ebenfalls ohne explizite Einstellung, b, c, d und f mit gesetzten Melious-Stufen. f (medium) trifft Sonnets Stufe, die Skalen (Claude Code, Melious) bleiben verschieden; der Effort des Advisors ist nicht erfasst.
- **Zeitmessung** hängt an Messpunkt und Teststrategie; die Läufe liefen teils gleichzeitig auf demselben Rechner.
- **Review und Gegenprüfung sind Modellurteile.** Die Zahl der Befunde misst nicht ihre Schwere. Die 95 Tests prüfen die Anforderungen des Plans, kaum Wartbarkeit, Lesbarkeit oder Sicherheit (nur Grundfälle).

Hosting in Europa

Melious AI GmbH (Deutschland, laut Webseite) leitet Anfragen an europäische Anbieter. Laut Melious (Stand 6. Oktober 2026) verarbeiten diese Partner Daten nur in Deutschland, der EU oder dem EWR und protokollieren nur operative Metadaten. Alle von Melious genannten Partner haben laut Melious ihren Sitz in der EU oder im EWR, einzelne auch Standorte außerhalb; ob dort Anfragen landeten und welcher Anbieter dieses Modell ausführte, ist nicht belegt. Das Modell stammt von DeepSeek (China), nur der Betrieb ist europäisch.

Noch offen

- Wiederholungen (n > 1), vor allem für a und b.
- Zeit bis zur ersten grünen Volltestsuite für c, d, e und f.
- Regelblock bei Claude-Läufen; Planung und Review mit DeepSeek; längere Läufe, hohes Risiko, parallele DeepSeek-Läufe (erste OpenCode-Versuche hingen).
- Ursache der Cache-Ausfälle bei `dsh` und bei Claude Code mit DeepSeek; h-Schätzung unvalidiert.

RÜCKFRAGEN

Fragen, Anmerkungen, Zusammenarbeit?

Dieser Bericht fasst eine einzelne Messreihe zusammen. Wenn Sie Rückfragen zu Methode, Zahlen oder Einschränkungen haben, eigene Erfahrungen mit den beteiligten Werkzeugen teilen möchten oder über den Einsatz von KI-Agenten in Ihrer Entwicklung sprechen wollen, schreiben Sie mir gern.

NAME

Sebastian Günther

IT mit Sicherheit

E-MAIL

mail@sg-it.info

WEB

sg-it.info

Mehr zum Prozess: sg-it.info/artikel/entwicklungsprozess-mit-ki-agenten · github.com/Crashman1983/dev-process

Stand der Angaben. Alle Zahlen stammen aus eigenen Einzelmessungen, Stand 6. Oktober 2026 (eine Aufgabe, eine Messung je Lauf) und sind kein Benchmark. Preise und Anbieterangaben können sich ändern. Dies ist ein Erfahrungsbericht ohne Gewähr, keine Rechts- oder Datenschutzberatung. Er wurde mit KI-Unterstützung erstellt und wird redaktionell von Sebastian Günther verantwortet. Es gibt keine Kooperation, kein Sponsoring und keine Vergütung durch die genannten Anbieter.

Wie lässt sich KI in Ihre Entwicklungsprozesse integrieren? Lassen Sie uns darüber sprechen.