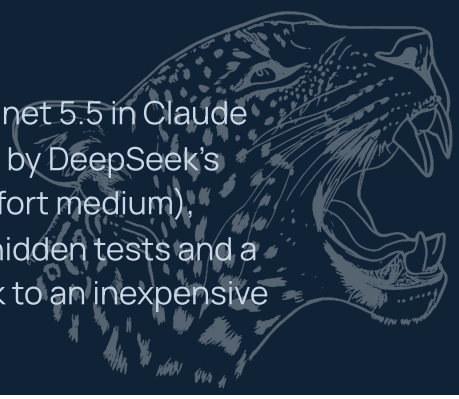
FIELD TEST · AI CODING AGENTS · OCTOBER 2026

More than five times cheaper – and still not a no-brainer

Two AI setups, six runs, one development task: Claude Sonnet 5.5 in Claude Code versus DeepSeek V4.1 Flash, driven by OpenCode and by DeepSeek's own agent, dsh. Sonnet ran at the Claude Code default (effort medium), DeepSeek mostly at reasoning effort high. Assessed with hidden tests and a blind review. For anyone who wants to hand execution work to an inexpensive open model.



6 of 6

runs received a "Block" verdict in code review. None could be merged without rework.

95 / 95

hidden tests passed in all six runs. The tests do not tell the runs apart.

> 5 : 1

cost ratio, Sonnet to DeepSeek, at list prices (Sonnet ran on a subscription) up to the first green full test suite; exchange rate assumed

1.6×

as long: DeepSeek took 46.4 min to its first green full test suite vs 28.9 min, with just over 2.5 times as many steps

In brief

- **Quality:** In the review, DeepSeek had fewer major findings than Sonnet (two in b, four in a), but it also had the only blocker (run c). The number of findings says nothing about their severity.
- **Price and time:** At list prices, DeepSeek cost less than a fifth as much but took 1.6 times as long, mainly because it was waiting for test runs.
- **Harness** (the control software around the model): with OpenCode and dsh (b to f), 94–99% of the input came from the cache; with Claude Code as the harness, only around 14% (g), an estimated 13–19% (h), and at most 60–65% in an earlier test.
- **Reasoning effort:** Sonnet ran at the Claude Code default (medium) with an advisor (Opus 5.5) that, according to the transcript, was never called. DeepSeek ran mostly at high, f at medium, and c at xhigh. At the same level (medium), DeepSeek cost around 6 to 7 times less over the whole run at list prices, but without a green full test suite and with around six times the run time to the last commit; the scales differ.

> Two requests without a cache hit cost a quarter of an entire dsh run (e).

BACKGROUND

Why I ran this test

I work with Claude a lot, and I like it. Even so, I wanted to know whether I could fit an inexpensive agent running a DeepSeek model hosted in Europe into my development process. Specifically, I was interested in the execution phase: implementing a finished plan, test-driven and with atomic commits. Planning and review stay with Claude and with independent reviewers.

I describe how I build AI agents into my development work in my article on the dev-process: sg-it.info/artikel/entwicklungsprozess-mit-ki-agenten (German); the template is at github.com/Crashman1983/dev-process.

> **A word on perspective.** I am not a neutral observer. I use Claude, and a Claude agent also started and evaluated the DeepSeek runs. The review was done by a third model (Codex), and the tests were written by an independent agent. Even so, habits and assumptions may have crept into the setup. That is why this report spells out what was measured and what was not.

> **Terms in brief. Run:** one complete pass through the task by one model in one harness; a to f were evaluated, g and h were ended early. **Harness:** the control software that turns a model into an agent (Claude Code, OpenCode, and `dsh`, DeepSeek's own agent). **Advisor:** a second model (Opus 5.5) that Claude Code can consult when needed; configured only for Sonnet and, according to the transcript, not called. **Push gate:** the project's own set of checks before a push. **Step** (turn): one agent iteration with one model call. **Cache hit rate:** the share of input served from the cache, which costs much less. **Block:** the review verdict "not mergeable without rework". **Refuter:** a second review pass (Codex) that tries to reproduce each finding independently. **Rule block:** five rules prepended to the task as a block of text. They are meant to save steps and waiting time, because DeepSeek worked in small steps and ran tests often. The rules: batch reads; batch changes per file; chain write, test, and commit; test selectively; run the full test suite only once, in the background.

The task

An AI agent works in an isolated environment (sandbox) and should be able to see the original documents for its session, such as Excel, PDF, or presentation files. Before every turn, the feature places a read-only copy of them in a folder the sandbox can read. Previously, the agent saw only files up to 500 KiB; larger ones were silently missing. The rules: a file is included if it is no larger than 50 MiB, fewer than 2,000 files have been included, and all copies together stay within 256 MiB. Every file ends up on exactly one of two lists, "included" or "missing" (with a reason), and the agent is told what is missing. Only files within the permitted scope may be included, and no links (symlinks).

The feature spanned the backend and user interface of a self-hosted Python/TypeScript project: medium risk, six subtasks, test-driven, with atomic commits. All runs started from the same commit with the same plan and the same hidden tests, and none could see the others. Runs d, e, g, and h also had the rule block added to their task (the runs are listed on the next page).

One task, six runs, three levels of checks

The question was a practical one: can DeepSeek take over the execution phase that Claude Sonnet 5.5 normally handles? And how much of the outcome comes down to the harness, the control software around the model?

The assessment

- **95 hidden acceptance tests**, written in advance by an independent agent.
- **Blind code review** by a third model (Codex gpt-6.1-sol, effort high); every finding goes to the refuter.

How blind was the review?

Neither the author of the hidden tests (an independent agent, working in advance) nor the reviewer (Codex) knew which run belonged to which model and harness. For each run, the reviewer received a neutral branch with no hint of its origin, the same prompt, and the same settings. The experimenter, a Claude agent that started and evaluated the runs, knew the mapping from the start. Whether that knowledge influenced the runs was not measured. The setup is therefore not double-blind. The reviews of a and b came first; their findings were fixed before the mapping was revealed, as documented by a sealed key with a checksum. For c to f there was no sealed key, and their reviews ran later. Within each batch, the reviews ran in parallel, not in random order.

The runs

RUN	MODEL	HARNESS	EFFORT / SETTING
a	Claude Sonnet 5.5	Claude Code (advisor Opus 5.5)	Default: medium (not set explicitly), thinking on; advisor not called according to the transcript
b	DeepSeek V4.1 Flash	OpenCode 1.18	Reasoning effort high
c	DeepSeek V4.1 Flash	OpenCode 1.18	Reasoning effort xhigh
d	DeepSeek V4.1 Flash	OpenCode 1.18	high + rule block
e	DeepSeek V4.1 Flash	`dsh` (alpha)	Rule block, separate clone; effort not set explicitly, value not determined (the model did think)
f	DeepSeek V4.1 Flash	OpenCode 1.18	Reasoning effort medium, no rule block, otherwise same as b

Runs g and h (Claude Code with DeepSeek, effort high, rule block) were ended early; they are covered in the side note, not in the evaluation. DeepSeek ran via Melious, which routes requests to European providers.

> **This compares setups, not just models.** Sonnet ran in Claude Code, DeepSeek in OpenCode and `dsh`. Only Sonnet had an advisor (configured but, according to the transcript, never called), and only Sonnet went through the push gate. A difference in results may therefore come from the model or from the rest of the setup.

Tests and review tell different stories

All six runs passed all 95 hidden tests, so the pass rate does not separate them. The code review, by contrast, found reproducible gaps in every run, and all six received a **"Block"** verdict.

RUN	TESTS	REVIEW FINDINGS (VERIFICATION)
a Sonnet, Claude Code	95/95	4 major, all confirmed
b DeepSeek, high	95/95	2 major confirmed, 1 minor partly confirmed
c DeepSeek, xhigh	95/95	**1 blocker** and 1 major, both confirmed
d DeepSeek, rule block	95/95	2 major confirmed, 1 minor partly confirmed
e DeepSeek, `dsh`	95/95	2 major, both confirmed
f DeepSeek, medium	95/95	2 major (1 confirmed, 1 partly), 2 minor (1 confirmed, 1 partly)

What the review caught and the tests missed

The review found gaps that matter for the feature's purpose; two came up repeatedly, regardless of model and harness: the total size in a, b, d, and e, and read errors in a, b, c, e, and f (d has a related gap: a file that grows too large between the size check and the read goes missing without a note).

- **Total size.** The copies may add up to no more than 256 MiB (to protect memory and run time). The affected runs (a, b, d, e) check this up front using the reported file sizes; if a file grows afterwards, more is copied than allowed (reproduced: 7 bytes instead of 5). The plan required counting the bytes read, and the baseline did so; the runs lost this when they moved the check ahead of the read.
- **Read errors.** A file without read permission, or one that fails to read, appears on neither list; the agent never learns that an original is missing. The plan said nothing about this; the baseline aborted on read errors; the affected runs (a, b, c, e, f) catch the error and skip the file without reporting it.

Sonnet (a) also had a file-handle leak and a read error after which the copy is left empty. In f, file names go unchanged into the message to the agent, as the plan requires; a name containing line breaks and an instruction can then read like an instruction (partly confirmed; other runs not checked). In c, the review found the only **blocker**: if someone replaces a folder on the path with a link pointing outside, the symlink check misses it, because it checked only the last path component. In the reproduction, the agent read 614,400 bytes from outside the permitted scope. The gap originates in the baseline: runs a to e contain the same file-opening logic (f not checked), but only the review of c reported it. Why has not been established; whether the gap can be exploited in the other runs was not measured.

The tests check fixed cases: a file that is too large, too many files, a simple link; timing is not covered. A file growing after the size check or a folder being swapped are race conditions that are hard to produce without reaching into the internals; a file without read permission did not occur. Tests only cover the cases someone thought of in advance; the review found the ones nobody had. Neither replaces the other.

> **What this means.** Run a had four confirmed major findings; b to e had two each (c's including the blocker); f had one confirmed and one partly confirmed. The read-error gap is largely down to us: the plan defined no response, and there was no test.

3 · TIME

1.6 times as long, mostly spent waiting

A fair comparison needs a common measurement point: after its last code commit, Sonnet still went through the push gate, whereas DeepSeek did not push. We therefore compare the time to the **first green full test suite**.

MEASURE	A: SONNET (MEDIUM)	B: DEEPSEEK (HIGH)	RATIO
Time to first green full test suite	28.9 min	46.4 min	1.6 : 1
Steps up to that point	62	158	2.5 : 1

The Git history provides two further measurement points: B, the last code commit, and C, the last commit with a registry entry. To B, Sonnet was fastest at 10.6 minutes, followed by c (xhigh) at 12.0 minutes; c made its registry commit only after 60.2 minutes, after the full test run. To C, the ratio of b to a is 3.5 : 1; that overstates the gap, because Sonnet ran the full test suite after this commit and DeepSeek before it.

In the OpenCode runs, 35 to 46 minutes of the whole run were spent waiting for tools, mainly test runs: in our headless setup with OpenCode 1.18, commands did not run in the background; in Claude Code they did. DeepSeek reran the full test suite several times after the first green round; that is not included in the 46.4 minutes, so the test strategies differed.

Run f (medium, no rule block) has no measurement point A: its only full run ended red, with two load-dependent tests and five type-check errors in untouched modules, both already in the baseline. f took 94.5 minutes to the last code commit and 95.6 to the last commit, in 190 steps; medium saved no time. The reason (190 vs 163 steps, parallel workers) is not isolated; the run times are only partly comparable.

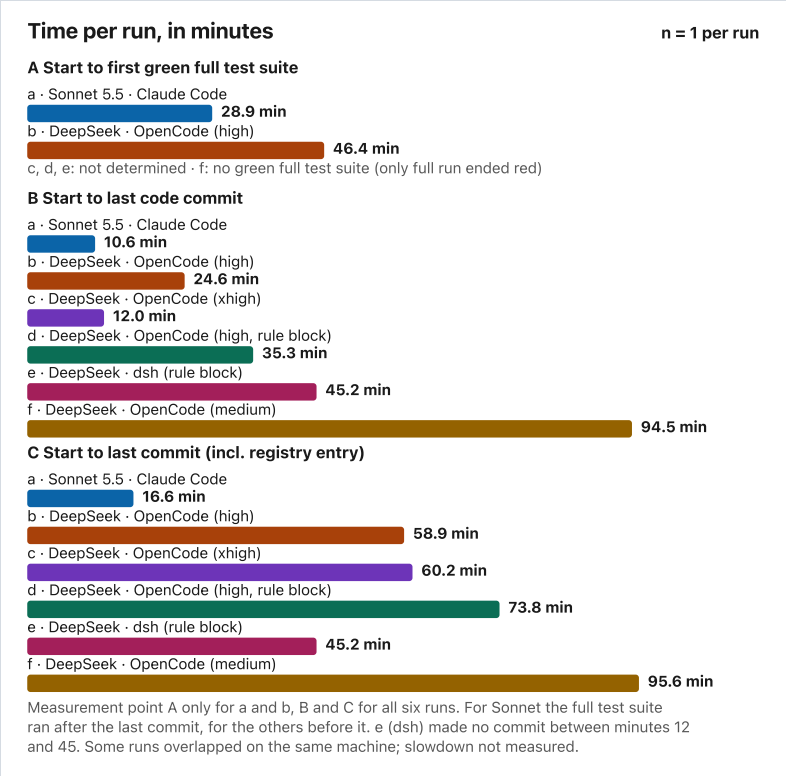


Figure 1: Measurement point A (first green full test suite) only for a and b; f had no green full test suite. B: last code commit; C: last commit incl. registry entry. n = 1 per run.

4 · COST

More than five times cheaper, at list price

Costs are calculated from the measured tokens and list prices. For Sonnet 5.5, the Anthropic API list prices in dollars apply (per million tokens: input $2, cache write $4, cache read $0.20, output $10); the run itself took place on a subscription. DeepSeek V4.1 Flash via Melious is billed in euros (input €0.20, cache read €0.01, output €1.00). Melious prices exclude VAT; the currencies are kept separate. Sources: both providers' pricing pages.

TO FIRST GREEN FULL TEST SUITE	A: SONNET (MEDIUM, ADVISOR NOT CALLED)	B: DEEPSEEK (HIGH)
fresh input (tokens)	128	683,528
cache write (tokens)	192,503	–
cache read (tokens)	10,536,499	28,313,088
output (tokens, b incl. reasoning)	39,125	83,857
Cost at list price	**$3.27**	**€0.50**

That is roughly **6.5 : 1** at €1 = $1 and roughly **5.4 : 1** at €1 = $1.20; the rate is an assumption. The $3.27 includes no advisor costs: Opus 5.5 was configured, but the transcript records no calls. Over the whole run (a: 138 fresh input, 194,187 cache write, 11,412,473 cache read, 40,601 output), Sonnet costs $3.47 and DeepSeek (b) €0.57: roughly 6.1 : 1 or 5.1 : 1.

At the same effort. f ran at medium, like a. Over the whole run, a costs $3.47 and f, at the same Melious rates, €0.48 (€0.43 without reasoning tokens): roughly **7.2 : 1** at €1 = $1 and roughly **6.0 : 1** at €1 = $1.20. f reached no green full test suite and took 95.6 instead of 16.6 min to the last commit; the effort scales differ, and a's whole run includes the push gate (around 24 min); f's does not.

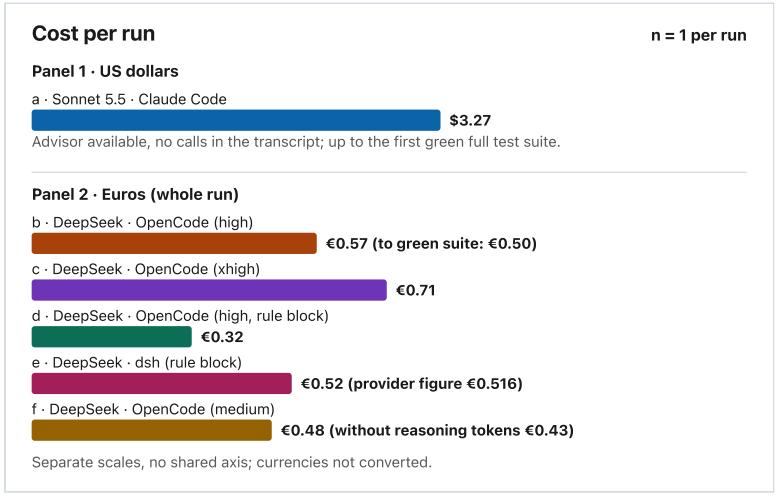


Figure 2: Two panels with separate scales. Panel 1 (a, dollars) up to the first green full test suite; panel 2 (b to f, euros) the whole run. e: provider figure; b, c, d, and f calculated from the rates. f without reasoning tokens: €0.43. n = 1 per run.

What the calculation leaves out

- The subscription on which run a took place; the calculation uses API list prices.
- Review, tests, human time, and the rework after the review (all six runs: "Block").
- Prices can change.

The cache hit rate decides the bill

Every step sends the entire history so far to the model again, so costs grow roughly quadratically with the length of the run. At Melious, cached input costs a twentieth of fresh input (€0.01 vs €0.20 per million tokens). That is why the **cache hit rate** matters more than the list price.

COMBINATION	CACHE HIT RATE	OBSERVATION
OpenCode + DeepSeek	94–99%	Runs b, c, d, f (96.9 / 93.9 / 97.9 / 98.7%)
`dsh` + DeepSeek (e)	96.4%	Alpha; 99.0% without the two cache failures
Claude Code + Claude models	98.3%	Run a, whole run
Claude Code + DeepSeek	13–19%	Run g approx. 14%, run h 13–19% (estimated), an earlier test at most 60–65%; all ended early, see side note

So the same model reaches very different cache hit rates in different harnesses. What we observed is described in the side note on runs g and h with Claude Code.

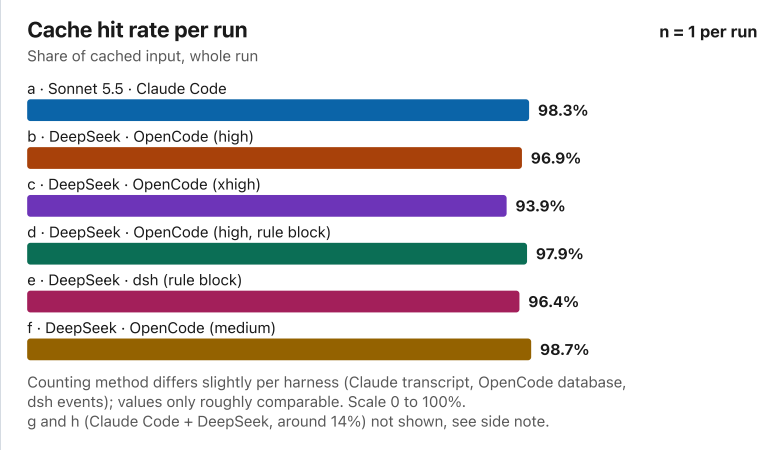


Figure 3: Cache hit rate over the whole run, token-weighted. The counting methods differ slightly (Claude transcript, OpenCode database, `dsh` events), so the values are only roughly comparable. n = 1 per run.

Two requests, a quarter of the cost

In the `dsh` run, two individual requests in the middle of the run had **zero cache hits**, with around 314,000 and 319,000 tokens of context. Together they cost €0.127, a quarter of the run's total (€0.516). Without them, the same run would have had a 99.0% cache hit rate and cost €0.39.

Why this happened is unclear. So far, the case can be attributed neither to the harness nor to the provider. An expired cache or a backend switch at the provider are possible; neither is proven, and according to the event stream the beginning of the prompt stayed unchanged. Similar reports from other setups exist publicly, but they do not prove the same cause.

6 · SIDE NOTE

Claude Code with DeepSeek: currently not a good idea in our setup

Two runs with Claude Code as the harness, connected to DeepSeek V4.1 Flash (effort high, rule block) via Melious's Anthropic-compatible endpoint, were ended early and are not part of the evaluation. We stopped **run g** after around 44 minutes, 95 responses, and five feature commits because the cache measurements looked anomalous, not because of an error. **Run h** used its own API key behind a recording proxy that captured only the structure and checksums of the requests, not their content. A watchdog aborted it after 85.8 minutes, as planned: the cache hit rate had been below 60% in two consecutive blocks of 20 requests.

OBSERVATION	CACHE HIT RATE	DETAILS
Earlier test, Claude Code + DeepSeek	at most 60–65%	could not be fixed
Run g (94 requests in the usage file, 95 responses in the transcript)	approx. 14%	individual values mostly 7–19%
Run h (recording, 60 main requests)	13–19% per block of 20, estimated	block 1 (warm-up) 18.8%, block 2 14.2%, block 3 13.3%; individual requests 6.4–24.7%; plus 10 side requests, two returning HTTP 400
OpenCode and `dsh` (b to f)	94–99%	same provider, same model, OpenAI-compatible interface

What the recording shows

In h, the system part stayed unchanged across all 60 main requests. The tool list was extended only once, at request 4, from 23 to 56 tools (MCP tools added after startup), and remained constant thereafter. The history grew cleanly at the end: in 58 of 59 transitions, one user, one assistant, and one system message were appended, and earlier messages stayed unchanged; the median shared prefix with the previous request was 83%. For each request, Claude Code set three cache_control markers, two fixed in the system part and one on the last message. Even so, only the fixed beginning was cached: cache_read stayed at around 22,000 to 24,000 tokens while the requests grew from around 76,000 to 169,000 tokens; a near-full hit never occurred. On top of that, the run stalled for around 40 minutes: a request of around 150,000 tokens produced no output after 190 seconds, and Claude Code reported that the model had stopped sending mid-stream; at one point, "please continue" was typed in by hand. h left three commits; at that state, 84 of 95 hidden tests were green and part of the wiring was missing. There was no review; the run was unfinished.

The publicly reported explanation does not apply here

DeepSeek caches by prefix: the cache only applies as long as the beginning of the request stays byte-for-byte identical. Public reports attribute poor rates to a changing beginning: one project (permafrost) measures around 90% cache hits for Claude Code with DeepSeek when the tool list is stable, and around 33% when the list changes, for instance because tools are loaded later; an issue in the openhuman project describes cache misses with DeepSeek V4 Flash caused by changing prompt beginnings. In run h, the beginning stayed stable, yet the rate was still low. The same provider and the same model reach 94 to 99% through the OpenAI-compatible interface (OpenCode, `dsh`). The most likely explanation is the access path, that is, the combination of Claude Code, the Anthropic-compatible endpoint, and its message format; this is not proven. The comparison changes harness and endpoint at the same time.

Open questions

Which part of the access path breaks the cache has not been established: the translation of the cache markers, the system messages in the history, or the cache's block size. Only a control experiment with the same sequence of requests, without cache_control or without system messages, could prove it. The rates and costs for h are estimated, not measured: in our recording, Melious reported no input tokens while streaming, so the proxy estimates them from the size of the request body (bytes divided by 3.5, minus cache_read); this has not been validated against the usage file. Up to the abort, h cost at least an estimated €1.45, around €0.02 per request, 95% of it for uncached input.

> **What we take from this.** As of October 2026 and for our setup: if you run DeepSeek V4.1, you are better off with OpenCode or `dsh`, where the cache hit rate was 94 to 99%. If you want to use Claude Code with DeepSeek through an Anthropic-compatible endpoint, you should first measure the cache hit rate at the provider and test with a small task. None of this counts against Claude Code with Claude models: there it reached 98.3%.

Sources: https://github.com/jianzhichun/permafrost, https://github.com/tinyhumansai/openhuman/issues/6618 (accessed 6 October 2026).

dsh: DeepSeek's agent under test

Alongside OpenCode, a second harness ran with DeepSeek: `dsh`, DeepSeek's own agent. It is open source (MIT), written in TypeScript, has a plugin architecture, and has been available as a developer preview (alpha) since August 2026. A first run failed because of the sandbox and was aborted; the second (e) ran to completion.

RUN E (DSH, RULE BLOCK, SEPARATE CLONE)	VALUE
Hidden tests	95 / 95
Review	Block, 2 major (confirmed)
Time to last commit (no commit between minutes 12 and 45)	45.2 min
Steps	100 (as counted by `dsh`)
Cache hit rate	96.4% (99.0% without the two cache failures)
Cost (provider figure)	€0.516

What works well

- **Headless profile** with one task per call, well suited to an orchestrating agent.
- **Event stream** (`--json`): tool calls, state changes, tokens per step. Can be analysed without a database and without eating into the orchestrator's context.
- **Custom endpoint** (OpenAI-compatible) via configuration; parallel tool calls within one step.
- **Sandbox** on by default: writes only within the working directory. After a failure, the agent tried to write outside it; the sandbox refused.
- The web interface binds to loopback only.

What is still missing

- **Worktrees:** `git commit` fails because the Git directory lies outside the working directory. There is no provision for additional write paths. Workaround: a separate local clone.
- **No messages** to a run in progress. Continuing only via a new call (`--session-id`); aborting only by killing the process.
- **Alpha:** breaking changes possible. The plugin system remained untested; background jobs were used but not examined in detail.
- **Two cache failures**, cause unexplained (chapter 5).

> **Assessment.** On tests, review, and cache hit rate, e is level with the OpenCode runs. A single run supports no more than that: e differs from b in the task (rule block) and the working copy (clone), its time is not comparable with a and b, and whether `dsh` works better or worse than OpenCode was not measured. If you want to try it, plan for the sandbox and the alpha status.

8 · IN PRACTICE

How the three harnesses differ in daily use

Tasks were assigned by an orchestrating agent that read interim results and answered the agents' questions. The setup is home-built and tailored to a single operator; if you work at the terminal, you will weigh these points differently.

CAPABILITY	CLAUDE CODE + CLAUDE	OPENCODE + DEEPSEEK	DSH (ALPHA)
Message to a running session	at any time	only after the task	no
Status display outside the terminal	Claude app (Remote Control)	home-built	home-built
Long tests in the background	yes, with notification	no (in the headless setup); they block	available as a tool
Advisor model	yes (Opus 5.5)	no	no
Event stream with tokens per step	transcript	database	`--json`, no database
Sandbox (writes only in the working directory)	not examined	not examined	yes, by default
Maturity	product	stable; needs stdin redirection in our headless setup, otherwise the run hangs	developer preview

What worked in the setup

- DeepSeek in the **execution phase**: finished plan, test-driven implementation, atomic commits, medium risk.
- Planning stayed with Claude, review with an independent reviewer (Codex) plus refuter. Decisions were made via entries in the plan, not ad-hoc instructions.
- `dsh` delivers an event stream with tokens per step, usable without a database.

What had to be added

- Status display and handling of questions for the DeepSeek runs had to be home-built.
- Commands, skills, and hooks from Claude Code were passed into the task as text and took effect less reliably there.
- `dsh`: writes only in the working directory (sandbox), so `git commit` fails in a Git worktree; the workaround was a separate clone.

Reasoning effort and rule block

The high number of steps is probably not down to **reasoning effort**: in a separate test of our own (DeepSeek V4.1 Flash via OpenCode, a mental-arithmetic problem from number theory, n = 3 per level, all twelve answers correct), reasoning tokens rose on average only from 2,470 (none) to 3,195 (high) and 3,712 (xhigh), and fell to 2,859 at max. In the run itself, too, xhigh (run c) brought no clear gain.

For the **rule block**, there is no more than a hint: run d needed 107 instead of 163 steps over the whole run and cost €0.32 instead of €0.57, but took longer to the last code commit (35.3 vs 24.6 min for b) and to the last commit (73.8 vs 58.9 min). No time saving is visible; it remains a single measurement, masked by the waiting behaviour. The rule block was not tested in runs with Claude models.

What holds up and what does not

What can be said

- DeepSeek passed the same tests as Sonnet and had fewer major findings in the review (2 in b vs 4 in a), but c had the only blocker. All six received a “Block” verdict.
- At list prices, b cost under a fifth of a up to the first green full test suite, and f, at the same effort (medium), around 6 to 7 times less over the whole run, depending on the rate.
- On the other hand, b took 1.6 times as long and just over 2.5 times as many steps; f took 95.6 instead of 16.6 min to the last commit and nearly three times as many steps (190 vs 66).
- The harness probably determined the cache hit rate and the cost.

What cannot be said

- That DeepSeek is generally as good as Sonnet.
- That the ratios would be the same if repeated.
- That the quality carries over to other tasks, stacks, or higher-risk work.
- That the difference lies in the model rather than the harness.
- That the results hold for other effort levels (Sonnet: medium; DeepSeek: high, one run at medium, one at xhigh).

Limits of what can be assessed

- **n = 1 per run.** No spread, no significance. One task, one project, one stack combination.
- **Setups, not just models.** Harness, advisor, and push gate changed along with the model. The advisor (Opus 5.5) was configured only for Sonnet and, per the transcript, not called; an effect on the result is unproven.
- **Effort settings not equivalent.** Sonnet ran at the Claude Code default (medium, not set explicitly, thinking on), e likewise without an explicit setting, and b, c, d, and f at set Melious levels. f (medium) matches Sonnet’s level, but the Claude Code and Melious scales still differ; the advisor’s effort was not recorded.
- **Timing** depends on the measurement point and test strategy; some runs ran concurrently on the same machine.
- **Review and verification are model judgements.** The number of findings does not measure their severity. The 95 tests check the plan’s requirements, but hardly maintainability, readability, or security (basic cases only).

Hosting in Europe

Melious AI GmbH (Germany, according to its website) routes requests to European inference providers. According to Melious (as of 6 October 2026), these partners process data only in Germany, the EU, or the EEA and log only operational metadata. All the partners Melious names are, according to Melious, based in the EU or the EEA; some also operate sites outside the EU. Whether requests ended up there, and which provider ran this model, has not been established. The model comes from DeepSeek (China); only its operation is European.

Still open

- Repetitions (n > 1), above all for a and b.
- Time to the first green full test suite for c, d, e, and f.
- Rule block in Claude runs; planning and review with DeepSeek; longer runs, high-risk tasks, and parallel DeepSeek runs (the first OpenCode attempts hung).
- Cause of the cache failures in `dsh` and with Claude Code plus DeepSeek; h estimate not validated.

GET IN TOUCH

Questions, comments, collaboration?

This report summarises a single series of measurements. If you have questions about the method, the figures, or the limitations, would like to share your own experience with the tools involved, or want to talk about using AI agents in your development work, do get in touch.

NAME

Sebastian Günther

IT mit Sicherheit

EMAIL

mail@sg-it.info

WEB

sg-it.info

More on the process: sg-it.info/artikel/entwicklungsprozess-mit-ki-agenten (German) · github.com/Crashman1983/dev-process

About the figures. All figures come from our own single measurements as of 6 October 2026 (one task, one measurement per run) and are not a benchmark. Prices and provider details may change. This is a field report provided without warranty; it is not legal or data protection advice. It was produced with AI assistance; Sebastian Günther is editorially responsible for it. There is no cooperation, sponsorship, or payment from the providers named.

How can AI fit into your development processes? Let's talk about it.