

What looking around leaves behind

Lab note 11 16 Oct 2026 9 min read
written by Claude from this repository's data

Before Claude changes code, it looks around: a search, a read, another search. Each step is a model request, and each request re-reads the whole conversation (note 1). What we had not priced is what a step leaves behind: a search's output and the lines it read stay in the conversation, and every later request reads them again until a compaction removes them. This note prices that, shows how it sank the first two versions of a mod, and follows the version that ships, seek, to a rule about where an instruction has to ride.

A token's second life

In the author's own sessions (nearly a thousand, counted only in aggregate), a token added to the conversation was re-read by a median of **143 later requests** before a compaction removed it (interactive sessions; 22 in headless `claude -p` ones). On Opus 5.5 a token costs \$8 per million to write to the cache once and \$0.20 per million each time a request reads it again: 143 re-reads add \$28.60. **About \$37 per million over its life, more than four times what writing it cost.** Over their life, the results of looking around came to about 11% of all the cost of the sessions before the pack was installed (15% in headless ones).

Our benchmark never saw this. A run ends when its task is done, so nothing it added is read again. We added a column, **carry**: the tokens a run added to the context $\times$ 140 later requests $\times$ the cache-read price, beside the run's own cost (`carry.py`). With carry, a run costs roughly what the same task costs inside a session that goes on.

Windows after a search

The first seek came from the same sessions. The most common predictable step after a search was opening a file it had just listed, mostly to read the lines around a hit. So seek attached them to the search's result: a narrow search got the lines around its hits (7 before, 30 after), a broad one its files ranked and the top file's lines, an empty one up to three wider searches.

- **seek 0.1** took only a plain `grep`. Opus's searches are rarely plain: about 100 of 110 sat behind a `cd`, in a chain or in a pipe. It fired in **1 of 36 runs**.
- **seek 0.2** read the hits from any read-only command line's output. Now it fired, and saved requests: 4.9 $\rightarrow$ 4.5 per task in a fresh session (−7.9%). The cost went the other way: **+2.1%** [−1.4, +5.5] fresh; **−4.5%** [−7.4, −1.6] in a long session, where a saved request is dearer.

- **With carry:** **+6.3%** [+1.4, +11.2] fresh, and **-2.2%** [-5.2, +0.7] in the long session, no longer shown.

The windows saved requests by adding text, and the text outlived the requests it saved.

Replayed against what Claude actually did next in the mined sessions, a window held Claude's next read in 0.5 to 12% of searches, depending on its size. At every size, its tokens cost more over their life than the requests it saved. The windows are retired (note 5).

An order in the middle of a search

The same sessions showed where the money sits: from the third consecutive request of looking around on, chains held 15% of requests and 10 to 16% of the cost, mostly in contexts over 400k tokens.

pathfinder watched for exactly that, and at the third such request in a large context told Claude to hand the rest of the search to a cheaper read worker through delegate.

Its plumbing worked; Claude did not follow. In five trial runs it took the order 0 times (in four of them the search was already over). In the measured batch, in a long session that had read click's docs and tests but none of its source, the order came in 6 of 12 runs and Claude handed off **once**. Cost **-3.4%** [-11.3, +5.1], not shown. Into the graveyard.

An order with the prompt

seek 0.3 turned the idea around: say it before the search starts, beside the person's prompt, in the person's name. The text names the cost and asks for lean steps:

The person's setting for looking around in this session (about 356k tokens of context so far): every response you make re-reads all of it, and whatever you read stays in it ... So look around in few, lean steps: put the searches and reads you can already name into one command; when a search shows where to look, read those places in the same step; read only the definitions or lines you need ..., not whole files or long ranges; and do not read again what this session already holds unless it changed.

It comes with the first prompt of a session, and again after a compaction or 150k more tokens of context; in between it stays in the context. (The batches below ran with an earlier wording that called every session "long", a fresh one too; the shipped one gives only the size so far. On the eight TypeScript jobs the two wordings came out the same: **+1.4%** [-4.6, +7.9] between them.) Against the pack alone, in the same batch:

Where	Runs	Cost	With carry
Fresh session: four click tasks and one TypeScript explanation	20 each	-11.9% [-17.2, -5.9]	-18.0% [-23.9, -11.5]
Long session that had read click's docs and tests, not its source	24 each	-10.2% [-17.6, -2.2]	-13.0% [-19.0, -6.8]
Long session that had read all of the source	24 each	-15.9% [-22.5, -8.6]	-17.7% [-24.6, -10.4]
Fresh session, TypeScript (chrono), eight tasks	24 each	-16.4% [-21.8, -10.5]	-23.1% [-28.6, -17.1]

- **Fewer requests, and less left behind.** On TypeScript, 6.3 → 5.3 requests per task and 18.6k → 13.2k tokens added to the context; fresh on click, 14.3k → 10.9k tokens. So carry makes the order look better, not worse: the opposite of the windows.
- **The bigger the job, the more it saved.** The TypeScript feature took 8.3 requests instead of 11.3; writing tests cost 29% less, the rename 26%. The smallest lookups gained nothing and sometimes took a step more (3 requests instead of 2 for the lookup in the docs session).
- **Quality held.** Every run got full marks but two: in the fresh batch, two runs of the TypeScript explanation scored 0.83, one with the order and one without, that task's usual spread.

The map, measured and left off

seek 0.3 had a second part: a code search's result named the definitions that hold its hits, with their lines, in no more tokens than a saved request is worth at the current context. It is the window idea at its leanest. Fresh on click it changed nothing (+1.2%, not shown); on TypeScript -4.5% [-9.9, +1.2]; on top of the order in the docs session -7.9% [-15.1, -0.5], and with carry -5.5% [-11.7, +1.0], not shown. It ships switched off (`map`).

Where an instruction has to ride

The orders and hints this lab has given, side by side:

What	Where it rode	Followed
delegate: build the approved plan through a hand-off	the person's approval of a plan	8 of 8
route: hand this feature work to the worker	the person's prompt	24 of 24
seek: look around in few, lean steps	the person's prompt	44 of 44 runs
pathfinder: hand the rest of this search off	Claude's own search, mid-way	1 of 6
lsp-plus: this LSP call answers your grep (note 2)	a tool's result	0 of 45

An order in the person's name that arrives with the prompt, before Claude has a plan, becomes part of the plan. One that arrives inside Claude's own search, attached to a tool's output, meets a plan already in motion, and Claude finishes what it started. Why is our reading; the counts are not. So seek speaks at the prompt and nowhere else: short, in the person's name, with the reason.

A subagent's task

Two of the pack's instructions ride where a subagent cannot see them. cells' chaining guidance is a section of Claude Code's system prompt, and a subagent Claude starts with the Agent tool has a system prompt of its own. seek's setting rides beside the person's prompts, and a subagent never sees those either: it sees the task Claude wrote for it. In the author's sessions none of 18 subagents had the guidance, and general-purpose subagents ran about a third of the Opus requests.

So both mods now add a line to a subagent's task as it starts (Claude Code's `agent.spawn` hook), under one heading, in the person's name:

The person's settings for this work:

- Fewer round-trips: every response you make re-reads the whole conversation, so a step that needs no judgment should not cost a response of its own. ...
- Lean looking around: whatever you read stays in your context ...

Not for a fork, which has the parent's context and both already; not for a teammate, a resumed subagent or delegate's worker, whose prompt is tuned for its job. Measured with every task handed to one general-purpose subagent, the pack as 0.6.0 ships with `subagents` off and on, 5 tasks × 4 runs per arm, Opus 5.5:

	Off	On	Change
Run cost, main and subagent	\$0.729	\$0.600	-17.6% [-21.0, -14.0]
The two features and the tests	\$0.850	\$0.664	-21.8% [-26.0, -17.4]
The two explanations	\$0.547	\$0.505	-7.8% [-12.3, -2.6]
Subagent requests	10.0	6.9	-31.0% [-36.8, -24.7]
Bash calls chained	88%	99%	

- **The lines arrived.** In 20 of 20 subagents with the option on, in none with it off.
- **Fewer, fuller steps, not more cells.** The subagents put more into each Bash call; run_cell was used less, not more (2.3 → 0.6 calls per subagent).
- **Quality held.** 37 of 40 runs got full marks; three TypeScript explanations scored 0.83, two of them without the settings.
- **What it turned up.** A subagent's transcript records each response's output as the response began, a few tokens, not what it ended with. Our bench had priced every delegate worker that way. These runs are priced in full from Claude Code's own count; the earlier numbers with a worker in them are corrected (CORRECTION-1016.md).

A subagent's task is the only text of the person's that reaches it, so that is where the settings ride.

What this does not show

- **Carry is a model.** It assumes 140 later requests, near the interactive median; headless sessions re-read a median of 22. Without carry, the order still saved 10 to 16%.
- **Three to six runs per task and arm.** The four measurements agree in direction; the sizes have wide ranges.
- **Bench sessions, not real ones.** Every prompt is a complete specification. The long sessions are forks of a prepared session; in the one that had read all of the source, runs whose fork had gone cold were left out (`--warm`).
- **The README's pack-against-plain numbers were measured without seek.** seek's are against the pack, in the same batches.
- **The subagents were ordered.** Every subagent run was told to hand the whole task to one subagent. How often Claude starts one by itself, and for what, decides what the settings are worth in a real session.

Method & limits

Arms, all with the person's installed plugins switched off (note 8): Z, the pack alone; ZKO and ZKM, with seek's order or its map (at every context size); PD, the pack in a session that had read click's `docs/` and `tests/` but nothing under `src/`, and there KDO, KDM and KD (order, map, both) and PFD (pathfinder); PL and KL, the pack without and with seek in a session that had read all of `src/click` (about 220k tokens). seek 0.1 and 0.2 ran as ZK against Z, and 0.2 as KL against PL. SA-off and SA-on: the pack as 0.6.0 ships, every task handed to one general-purpose subagent, cells' and seek's `subagents` off or on; click's feature and explanation, chrono's feature, tests and explanation; checked and summed by `subagent_report.py`. Opus 5.5, click 8.5.0, chrono 2.10.3. Carry, for example: `python3 -I carry.py Z ZKO results-seek03-fresh.jsonl`; 95% bootstrap intervals over runs within tasks, whose ends move by a few tenths of a point with the arms resampled beside. The mined numbers are aggregates; the sessions stay private. Data: `results-seek03-fresh.jsonl`, `results-seek03-docs.jsonl` (pathfinder too), `results-seek03-long.jsonl`, `results-seek03-ts.jsonl`; subagents: `results-subagent.jsonl`; seek 0.1 and 0.2: `results-seek.jsonl`, `results-seek2.jsonl`, `results-seek-long.jsonl`.