

Borrowing the good ideas

Lab note 4 15 Oct 2026 4 min read
written by Claude from this repository's data

The code graph in the last note saved nothing we could measure on our bench, alone or beside the pack. That does not make their ideas wrong. Two of them are good, and we tried to carry them over in a form that fits how Claude actually works: answers placed where it already looks, from sources that are never stale.

Idea 1: a pointer before the first search

Trail-Code's prompt hook adds a few likely places to a prompt before Claude starts, so it can skip the first search. The step before you know a symbol is the one a language server cannot help with, because it needs a symbol to ask about.

Our first version failed in calibration. We started with what the independent evaluation found strongest (Trail-Code #117): BM25 keyword ranking over whole files (Robertson & Zaragoza 2009), term counts from `git grep`, so there is no index to keep fresh. It speaks only when the best file is clearly ahead of the field.

Before spending a single benchmark run, we looked at what it would attach to 13 prompts: our seven task prompts and six everyday ones ("commit and push this", "run the tests and tell me what fails", "fix the typo in the README"...).

- The first version pointed "commit and push this" at an example program that happens to define `commit()` and `push()` functions.
- After teaching it to skip paths and flags, to weigh code over prose, and to need a rare code identifier from the prompt, it went quiet on 11 of 13 prompts. On the other two it pointed at the wrong code: once at the word `inner`, taken from a formatting example in the prompt ("nested functions as `outer.inner`"), and once at `pytest`.
- Silent where it should speak, confidently wrong where it spoke. A pointer that is wrong costs more than no pointer: Claude reads the wrong file first.

What survived is the deterministic half. When a prompt spells out a code name (`format_filename` , `ParamType` , `echo()`), scout looks up where that name is defined, with one `git grep` per name, and attaches the definition if there are one to three of them in code files (the code's own ahead of tests and examples):

```
Where the code names in this prompt are defined (git grep):
`resolve_color_default`: src/click/globals.py:54 `def resolve_color_default(color: bc
```

On the same 13 prompts it spoke on 4, correctly each time, and said nothing on the other 9. A name that is defined nowhere (a new one, a word in backticks) or in many places gets no pointer. Neither does a file Claude has already read, nor anything in a session past 80k tokens of context, where Claude has most likely read the code.

One caution we owe the reader: we calibrated on the same prompts we then benchmarked on. The surviving rule has no threshold we tuned, which limits the risk, but it is not a held-out test.

Idea 2: the blast radius after an edit

Trail-Code prints the callers of what you just edited, from its graph. Our version asks the live language server, and only when an edit changed a function's parameters, the moment callers may break:

```
`resolve_color_default`'s parameters changed; per the language server it is called from
src/click/exceptions.py: __init__ (48)
src/click/termui.py: _readline_prompt (99), get_pager_file (362), progressbar (581)
src/click/utils.py: echo (342)
```

To give it a fair chance we added a task built for it: make `resolve_color_default`'s parameter keyword-only and rename it, so three positional callers must change, checked by a hidden test and a scan for stale calls.

What it measured

96 runs of the ideas, all correct (`results-ideas.jsonl`), beside the pack's own runs. The first round left every interval across zero, so we ran scout against no scout again on the five tasks where it speaks, for eight runs per task:

	Change in cost	95% interval
The pack with both ideas, against the pack, 8 tasks	-2.5%	-6.9% to +2.0%
Taking scout out, on the 5 tasks where it speaks	+3.8%	-1.9% to +10.3%
Taking blast out, on the signature task	+3.7%	-2.0% to +11.0%

On the signature task itself, Claude needed 4.2 requests with the pack, 3.4 with both ideas, 5.2 with only the blast radius, and 6.5 with Trail-Code.

Twice the runs, the same answer: a small saving in the direction we hoped, never a wrong result, and an interval that still crosses zero. By our own rule that is not enough to ship. We measured once more on 30 real bug reports, whose text names functions far more often than our prompts: scout spoke on 20 of them and saved nothing (+7.3% against the pack [-6.7%, +20.7%]; note 9). Both ideas stay out of the pack.

Why the blast radius barely matters here was visible in the transcripts: before editing a function, Claude greps for it, and lsp-plus already answers that grep with the same list of callers. By the time the signature changes, Claude knows who calls it. An idea that is valuable in a tool without lsp-plus is redundant in a pack with it. It stays in lsp-plus, off by default.

What this changed

- Calibrate before you benchmark. Thirteen prompts and a script cost nothing and stopped a feature that would have spent money being wrong.
- Prefer the deterministic version of an idea. "Where is the name you typed defined" can be wrong only if the code changed in the last second; "which file is most similar to your words" can be wrong in a hundred ways.
- An idea's value depends on the pack around it. Measure it as an addition to what already ships, not on its own.

Method & limits

Arms: Z (the pack), N (the pack with scout and lsp-plus's `blast` on), N-scout (without scout), N-blast (without blast). Opus 5.5, click 8.5.0, fresh sessions; four runs per task and arm, eight for N and N-scout on the five tasks where scout speaks. Scout's calibration script and prompts are in the lab. Data:

`results-ideas.jsonl`, the pack's runs in `results-trail.jsonl`.

Sources

- trailhq, Trail-Code: prompt hook and post-edit blast radius; issue #117.
- Robertson, S., & Zaragoza, H. (2009). The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval*, 3(4), 333–389.