

Small models as plumbing

Lab note 7 15 Oct 2026 6 min read
written by Claude from this repository's data

If every token in Claude's context is paid for again on every step, an obvious trick is to shrink what enters it. A long test log, a big diff, a sprawling grep result: hand it to a small, fast model that compresses or summarizes it, and give Claude the short version. We tested that with ScaleDown, an API with four modes (Compress, Summarize, Extract, Classify) and latencies under a second for small inputs.

The price question settled itself. The person running this lab paid €2.50 for 50 million tokens, about \$0.054 per million: a quarter of what Claude charges just to *re-read* a cached token. Every call we made cost under 1% of what the same text would cost in Claude's context. So price is not the question. Whether the short version keeps what Claude needs is.

Compress: the right passages for the wrong question

Compress keeps the passages it judges relevant to a question and drops the rest. On click's source it kept 1–5% of the text, in 1.6–3.6 s. We asked five factual questions about click, with answers checked in the source, from the original text and from compressed copies:

Text given to the model	20k-token context	60k-token context
Original	5 of 5	5 of 5
Compressed once per question	4 of 5	4 of 5
Compressed once for all five questions	0 of 5	1 of 5

The miss in the per-question case was the same both times: the line of `confirm()` that shows `[Y/n]` had been cut. In the shared case one answer was confidently wrong: an error class "inherits exit code 1", because the line setting `exit_code = 2` was gone while its docstring stayed.

An agent's next question is never known when its context is built. Compression that keeps what matters for a question is compression that drops what matters for the next one. Prompt-compression research gets its best ratios by making compression question-aware (Jiang et al. 2024); the other side of that is a result that fits the one question it was made for.

Summarize: confident, and wrong about the important part

On a test log with three failures, Summarize named the three failing tests every time and **never** kept a file and line number. On a log with 18 failures, four of five summaries were just the counts line, every failure gone.

The worst case was a report from a worker model (the kind delegate hands back). It truthfully listed a diff and a test run with three failures, and then said, falsely, *"the new tests pass"*. A plain summary repeated the false claim **3 of 3 times**, and once did not mention the failure at all. Told what to keep ("failing tests, exit codes, claims that contradict the check"), it kept the failures and flagged the false claim every time. Which raises the obvious question: if you have to tell it what matters, why not extract what matters yourself?

Extract: right, and slower than a parser

Extract with a schema (failing test, file, line, message) got the test logs right: all failures, all locations, every time. It took 4–10 s, and one call failed after 77 s. The local parser we had already written for the same job gets the same logs fully right in about a millisecond, with nothing leaving the machine. (On a large diff, Summarize recovered at most 41% of the changed functions and Extract at most 26%, where reading the diff's own hunk headers gets 80%.)

Classify: the one job it did perfectly

Classify sorts a text into labels you give it. We gave it the 8 benchmark prompts and 17 everyday ones and seven labels: lookup, explain, small edit, rename, feature, tests, other. Asked one prompt at a time: **25 of 25** right, median 0.42 s, \$0.00003 a call. (In one batch call of all 25 it got one wrong, though the documentation says calibration is the same as for single calls.)

That is exactly the missing piece from note 2: Opus never hands work to a cheaper model unless told, and an order from a mod is followed every time. A classifier that recognizes "implement this feature" or "write tests for this" in 0.4 s could give that order for exactly the work where a hand-off pays (22–35% less on features and tests), and stay silent on small questions, where a hand-off costs up to 3.6 times as much.

So we built it. **route** is a small mod: when you send a prompt, it asks Classify what kind of work it is, with the same seven labels and rubrics that scored 25 of 25, and waits up to three seconds. If the answer is feature or test work with a score of at least 0.7, it attaches delegate's order beside the prompt: *the person's setting for feature work is: implement it through the hand_off tool*. Anything else, or no answer in time, adds nothing.

On our eight click tasks, four runs each, all 32 correct (`results-route.jsonl`):

	The order given	Hand-offs	Cost, the pack → the pack with route
The feature	4 of 4	4	\$0.61 → \$0.45 (9.5 → 5.2 main requests)
The coverage task	4 of 4	4	\$0.67 → \$0.41 (8.5 → 4.2)
The six other tasks	0 of 24	0	about the same

Every routing decision was right: the order on all eight feature and test prompts, on none of the other 24. On the two tasks it routed, the pack with route cost **33.8% less** than the pack alone [−39.4%, −27.4%]; on the other six, +4.3% [−0.1%, +9.0%], within the runs' spread. Over all eight, **−14.2%** against the pack [−17.9%, −10.1%] and **−19.1% against plain Claude Code** [−22.5%, −15.3%].

Corrected 16 Oct: worker costs were undercounted. A worker's transcript records its output as each response began; with it counted in full, from Claude Code's own totals, the routed tasks went from −39.6% to −33.8% and all eight from −17.0% to −14.2% (CORRECTION-1016.md).

This closes the loop from note 2. Opus never delegates on its own and always follows an order; a 0.4-second classifier decides when an order is worth giving. What it costs: a ScaleDown key, about 0.4 s (up to 3 s) before each prompt starts, \$0.00003 a prompt, and the prompt's text leaving the machine. So it can only ever be opt-in.

What it adds up to

Use	Verdict	Why
Compress Claude's context	No	Loses facts the next question needs
Summarize logs or worker reports	No	Drops locations; repeats false claims unless told what to keep
Extract failures from logs	Not needed	A local parser is as right, thousands of times faster, and private
Classify prompts for routing	Yes, opt-in	25 of 25, 0.4 s; routing feature and test work to a cheaper worker: −14% over the pack

Small models make good plumbing where the job is narrow and checkable: a label from a fixed set, on which a stronger model then acts. They make poor editors of what a stronger model will reason over, because they cannot know what it will need.

Privacy is part of the cost: every call sends the text to a processor in the US. ScaleDown's data processing terms, as provided with the account, promise zero retention and no training on customer data.

We sent only public code (click) and text we generated, and any mod that used it would have to be off by default and say so.

Method & limits

151 API calls, 1.8 million billed tokens in total, on click's source and on logs, diffs and greps generated from it; answers scored against ground truth worked out from the code. Fidelity of compressed context judged by Claude Haiku 5.5 answering from the text alone. Five runs per case unless noted; Classify on 25 prompts, one run each plus repeats. The key ran out of credits midway through the first round; the second round ran on new credits. Scripts and results stay in the lab with the API client.

Sources

- ScaleDown, documentation, model overview and Classify reference; price and data terms as provided with the account.
- Jiang, H., Wu, Q., Luo, X., Li, D., Lin, C.-Y., Yang, Y., & Qiu, L. (2024). LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. *ACL 2024* (Vol. 1), 1658–1677.