

tokenscope

See where your AI coding tool's
tokens actually go

1%

of a request is what you
actually typed

39%

of cost comes from output,
which is only 8% of tokens

5×

one file read, billed five times
over five turns

Contents

1	What it solves	why logs can't tell you, but a proxy can
2	Install	one command, optional exact tokenizer
3	60-second tour	offline, free, before you commit
4	Wire up your agent	four scenarios, copy-paste ready
5	Read the results	terminal reports and browser dashboard
5b	How long you have left	quota, burn rate and proactive alerts
6	Set prices	make the cost column mean something
7	Privacy & credentials	what lands on your disk
8	Command reference	every subcommand and flag
9	Troubleshooting	the four most common problems

1 What it solves

Most of what you pay an AI coding tool goes somewhere you never see. In a single request, the requirement you typed is roughly **1%** of the payload. The rest is tool descriptions, conversation history and files that were read in — all sent on your behalf by the framework, and re-sent on every turn.

Three concrete findings

Your words are 1%

The other 99% is the system prompt, tool schemas, past turns and tool output.

Output is small but pricey

Model replies are 8% of tokens yet 39% of cost — output bills at 5x input.

The same content billed again

Reading one file cost 657 tokens, but five turns later you paid 3,300.

The third one matters most because it overturns a common assumption: people reach for a cheaper model, when dropping files you no longer need or turning on prompt caching saves far more.

Why not parse logs

Every framework logs differently, and most record only a final token count with no breakdown. But every agent has to reach a model over HTTP — put a reverse proxy in the middle and one implementation covers all of them, with access to the **full request body**. That is what makes a breakdown possible at all.

The proxy is fully transparent

Status codes, response headers and response bytes pass through unchanged; SSE streams are relayed chunk by chunk with no buffering. Accounting happens on the side, and a parse failure never affects your agent. Tests enforce this, including byte-identical streaming responses.

What it supports

Any agent that honours a base URL. Verified with Claude Code, Aider, Cline, Continue and Kiro CLI. Upstream can be a cloud API (Anthropic, OpenAI, a self-hosted corporate gateway) or a local Ollama / LM Studio. Multiple agents can be tracked at once and are kept separate in the reports.

2 Install

Requires Python 3.10 or newer. A virtual environment is recommended.

```
python3 -m venv .venv
. .venv/bin/activate
pip install -e .
```

Verify:

```
tokenscope --version
```

Optional: exact tokenizer

```
pip install tiktoken
```

It works without it

The estimate only decides the **relative share** of each component. Absolute numbers always come from the usage the upstream reports, and components are scaled proportionally to it — so a systematic bias in the heuristic largely cancels out. tiktoken improves precision; the startup panel shows which mode is active.

Make the command global

Commands inside a virtualenv are not on your PATH. To avoid activating it every time:

```
mkdir -p ~/.local/bin
ln -sf "$PWD/.venv/bin/tokenscope" ~/.local/bin/tokenscope
```

Note

The symlink points into the project's virtualenv, so do not move or delete the project directory. Also confirm `~/.local/bin` is on your PATH.

Language

All output is available in English and Chinese, auto-detected from `TOKENSCOPE_LANG` or your system locale:

```
tokenscope report --lang en      # one-off
export TOKENSCOPE_LANG=en        # for every command
```

3 60-second tour

Before wiring up your real traffic, walk through the reports with built-in sample data. **No network, no API key, no cost**, and it cleans up after itself.

```
tokenscope demo
```

Six tables in sequence, each introduced with one plain sentence about the question it answers:

1 • Overview	total spend, split across your tools
2 • Breakdown	what actually consumed the money. The core of the tool
3 • Tool cost	which tool calls cost most, variable vs fixed
4 • Amplification	the counter-intuitive one: content billed over and over
5 • Sessions	which concrete task the money went to
6 • Per-turn detail	every request in one task, with caching explained

To keep the sample data around, or try it in the dashboard:

```
tokenscope demo --keep ~/ts-demo.db
tokenscope report --db ~/ts-demo.db --view amplify
tokenscope web --db ~/ts-demo.db
```

4 Wire up your agent

Point the agent's base URL at the local proxy; the proxy forwards to wherever it went before:

```
agent -> tokenscope (localhost:8787) -> your original model service
```

Scenario A: Claude Code on the official API

1 Redirect its config through the proxy

```
tokenscope hook claude --agent claude-code
```

This edits `env.ANTHROPIC_BASE_URL` in `~/.claude.json` and records the original upstream in `~/.tokenscope/hooks.json`. Credentials and model names are untouched.

2 Start recording (keep this terminal open)

```
tokenscope serve
```

No `--upstream` needed: hook already stored the original, and serve picks it up automatically.

3 In another terminal, work as usual

```
claude
```

No environment variables required. The serve terminal prints one line per request.

4 Restore when done

```
tokenscope unhook claude
```

The proxy must stay up while hooked

The config points at `127.0.0.1:8787`, so Claude Code cannot connect if the proxy is not running. Run `tokenscope hook --status` if unsure.

Scenario B: Claude Code behind a corporate gateway

Identical to scenario A. `hook` reads whatever gateway URL is currently in your config and records it; `serve` then points there. Gateways with a path prefix (e.g. `http://gw.corp/anthropic`) are supported.

Why the config file must be edited

The `env` block in Claude Code's config takes precedence **over** shell environment variables. Plain `export ANTHROPIC_BASE_URL=...` is silently ignored — the tool keeps working but traffic bypasses the proxy, which looks exactly like "monitoring is broken". This is the single most common trap.

Scenario C: local Ollama

```
# terminal 1
tokenscope serve --preset ollama

# terminal 2
export ANTHROPIC_BASE_URL=http://127.0.0.1:8787/a/claude-code
export ANTHROPIC_AUTH_TOKEN=ollama
claude --model qwen3-coder
```

Ollama v0.14.0 and later implement the Anthropic Messages API natively, so Claude Code talks to it directly with no translation layer.

Scenario D: Aider and other OpenAI-compatible tools

```
# terminal 1
tokenscope serve --preset ollama

# terminal 2
export OPENAI_API_BASE=http://127.0.0.1:8787/a/aider/v1
# or, when Aider talks straight to Ollama
export OLLAMA_API_BASE=http://127.0.0.1:8787/a/aider
aider --model ollama_chat/qwen3-coder
```

Ready-made wiring commands for every framework:

```
tokenscope env
```

About the `/a/<name>` prefix

The `/a/<name>` segment decides which agent the traffic is filed under in the reports; it is stripped before forwarding. Requests without a prefix land under `default`, or you can set the `X-Tokenscope-Agent` header.

Mixing upstreams

Cloud by default, one agent to a local server:

```
tokenscope serve --preset anthropic --route aider=http://127.0.0.1:11434
```

5 Read the results

Terminal

```
tokenscope report           # all four views
tokenscope report --view components # what consumed the money
tokenscope report --view amplify  # resend amplification only
tokenscope report --agent claude-code # one agent
tokenscope report --days 1       # last day only
tokenscope report --json         # machine readable

tokenscope sessions         # what you worked on, and what it cost
tokenscope session a3c530d6  # per-turn detail (prefix is enough)
```

Tables adapt to terminal width: on narrow screens explanatory columns are dropped rather than wrapped.

Browser dashboard

```
tokenscope web
```

Defaults to `http://127.0.0.1:8788` and opens your browser. The dashboard draws token share and cost share side by side; click any session for per-turn detail and a prompt-growth curve. The top right filters by agent and time range, switches language, and exports JSON. Sessions support deep links (`#session=<id>`), so URLs are shareable.

No external resources

No CDN, no web fonts, no analytics. Charts are hand-written SVG and the whole dashboard is one self-contained HTML file. It works offline, and it never contacts a third party just to show you your own bill.

The dashboard has no authentication

It binds to `127.0.0.1` by default. Anyone who can reach the port can read your task titles and cost data. Binding to `0.0.0.0` requires an explicit `--host` and prints a security warning.

Where the data lives

`~/.tokenscope/usage.db` main SQLite database; `-wal` / `-shm` belong with it

`~/.tokenscope/hooks.json` hook state: config path and original base URL

`~/.tokenscope/pricing.json` the model prices you filled in

Change the location with `--db /path/to.db` per command, or `export TOKENSCOPE_HOME=~/.somewhere` for the whole directory. Wipe everything with `tokenscope reset --yes`.

5b How long you have left

Earlier chapters answer "where the money went". This one answers "how long can I keep going". There are three tiers of data, and which one applies depends on whether your upstream reports quota information.

Three tiers

1 • Reported by upstream	On a Pro/Max subscription talking directly to the official API, response headers carry utilisation and reset times for the 5-hour and 7-day windows; pay-per-token accounts get per-minute request and token allowances
2 • Locally declared	Self-hosted gateways usually strip those headers. Declare your own budget and tokenscope tracks a rolling window from its own records
3 • Trend only	With no quota data at all you still get burn rate and "how big the prompt will be five turns from now"

A subtlety on subscriptions

Both the 5-hour and the 7-day window must have room, but which one is authoritative is stated by the `representative-claim` response header. tokenscope follows it rather than guessing — ignoring that field is what blocks people who still have 98% of their 5-hour window left.

Terminal status bar

```
tokenscope live           # keeps refreshing
tokenscope live --once    # print once and exit
tokenscope live --interval 5 # refresh every 5 seconds
```

Shows used share, remaining, and reset countdown per window, plus current burn rate, projected time until you hit the cap, and the current task's turns and accumulated cost.

Declare your own budget

The format is `<window>=<amount>`, accepted by both `serve` and `live`:

```
tokenscope serve --limit 5h=200k # 200k tokens per 5 hours
tokenscope serve --limit 1d=$5   # $5 per day
tokenscope live  --limit 7d=10M  # 10M tokens per 7 days
```

The rolling window starts at the first request inside it, not on the hour — matching how Anthropic's 5-hour window actually behaves.

Proactive alerts

While `serve` is running it speaks up in three situations, so you do not have to check:

- the authoritative window drops below 20% remaining (below 8% escalates to critical)
- at the current rate you will hit the cap within 20 minutes (6 minutes escalates)
- the session's prompt is on track to grow more than 1.5x over the next five turns

The same alert repeats at most once every 5 minutes. Alerts that fire constantly are alerts nobody reads. Disable them with `--no-alerts`.

Burn rate needs enough samples

Agents often fire several requests within a second or two. Extrapolating "per minute" from a 1.5-second sample yields absurd numbers, so no rate is shown until the sample spans at least 20 seconds — better nothing than a figure that leads you to a wrong decision.

Live panel in the dashboard

The browser dashboard shows the same data graphically at the top, refreshing every three seconds (paused while the tab is in the background). Quota bars change colour with remaining share, and the authoritative window is badged.

6 Set prices

The built-in price table is a **reference only** and may be out of date. Reports state plainly which models use built-in defaults and which have no price at all (counted as zero) — nothing is presented as free when it is not.

```
tokenscope pricing          # writes ~/.tokenscope/pricing.json
tokenscope pricing --path  # print the path only
```

Add entries keyed by model-name prefix, in **USD per million tokens**:

```
{
  "your-org/your-model": {
    "input": 3.0,
    "output": 15.0,
    "cache_read": 0.3,
    "cache_write": 3.75
  }
}
```

Longer prefixes win, so `claude-3-5-sonnet` is not shadowed by `claude`.

Local models cost \$0, but the counts still matter

Token volume directly determines VRAM use, whether you overflow the context window, and inference time. The cost column reflects cloud calls only — for local models, read tokens and latency.

Budget guard

```
tokenscope serve --budget-usd 2.0          # warn only
tokenscope serve --budget-usd 2.0 --block-over-budget  # reject once over
```

Warning is the default. `--block-over-budget` returns 429 once a session exceeds the budget, which surfaces as an error inside your agent — enable it only if that interruption is acceptable.

7 Privacy & credentials

A proxy inevitably sees your token in transit; that is what a proxy is. What it does not do is parse it, log it, persist it, or send it to any third party. Here is how to check that yourself.

One command

```
tokenscope audit --self-test
```

It injects a fake token, runs a full request through the proxy, then checks whether that token appears in the database, the logs, the report output or the error response, and confirms there is exactly one outbound destination. Entirely local, and each check states how it was performed.

See what is on disk

```
tokenscope audit
```

Lists every file the tool has written, its permissions, and whether it contains credentials or prompt text. Rows with loose permissions are flagged in red. It also lists **every address the proxy has connected to** — every outbound request is recorded, including failed ones, so the list is complete.

Key facts

No copy of your credentials	hook does not back up the config by default. Before writing it verifies the diff is <i>exactly</i> the base URL and refuses otherwise
No loosened permissions	config rewrites inherit the original mode; database and state files are 0600
No message bodies stored	only token counts and components. The one exception is the task title (first 70 characters), which <code>--no-titles</code> disables
One destination only	the sole outbound connection is the upstream you configured

Limits of a self-test

The self-test is run by this program, and a malicious build could make it lie. It proves this code does not leak along this path, and it will catch a tampered install, but it does not replace reading the code or capturing packets. For a stronger guarantee, read `tokenscope/proxy.py` (about 350 lines) or diff the traffic yourself.

Keep prompt text off disk entirely

```
tokenscope serve --no-titles
```

Reports then show only a session hash; all other statistics are unaffected.

8 Command reference

Command	What it does
tokenscope demo	zero-config tour on sample data
tokenscope serve	start the recording proxy
tokenscope web	open the browser dashboard
tokenscope live	terminal status bar: quota left and time remaining
tokenscope hook <target>	route an agent through the proxy (reversible)
tokenscope hook --status	show current hook state
tokenscope unhook [target]	restore config; omit target for all
tokenscope report	the reports
tokenscope sessions	session list
tokenscope session <prefix>	per-turn detail for one session
tokenscope audit	files written and destinations contacted
tokenscope audit --self-test	credential-leak self check
tokenscope env	print wiring commands per framework
tokenscope pricing	create the editable price file
tokenscope reset --yes	wipe collected data

Common serve flags

Flag	Meaning
--preset	upstream preset: anthropic / ollama / openai / lmstudio
--upstream URL	upstream address directly
--route AGENT=URL	per-agent upstream, repeatable
--port N	listen port, default 8787
--budget-usd N	per-session budget cap
--block-over-budget	reject instead of warn when over budget
--limit SPEC	declare a local budget, e.g. 5h=200k or 1d=\$5

--no-alerts	no proactive quota or growth alerts
--no-titles	do not record task titles
--quiet	no per-request log lines
--db PATH	database path

Global flags

Flag	Meaning
--lang zh en	output language; accepted before or after the subcommand
--version	print version

Environment variables

Variable	Meaning
TOKENSCOPE_LANG	output language, zh or en
TOKENSCOPE_HOME	data and config directory, default ~/.tokenscope

9 Troubleshooting

tokenscope: command not found

The command lives in the virtualenv and is not on your PATH. Activate the environment (`.venv/bin/activate`) or create the symlink from chapter 2.

The agent works but the report is empty

Traffic is not going through the proxy — by far the most common problem. Check in order:

- Is the proxy up? `curl -s http://127.0.0.1:8787/___tokenscope/health` should return `{"ok":true,...}`
- Is the hook active? `tokenscope hook --status`
- If you set variables by hand: was the export in the **same terminal**, and does the agent's config file override it? Claude Code does.
- Same database? If serve used `--db` and report did not, the data is in two different places.

The agent cannot connect

Most likely the config is still hooked but the proxy is down. Either start `tokenscope serve` or run `tokenscope unhook claude`.

Every cost shows \$0

The model is not in the built-in price table; reports say so explicitly. Add prices as described in chapter 6. Token and cache statistics are unaffected.

Code changed but behaviour did not

serve is a long-running process. Restart it after editing code.

Verify the whole chain yourself

Confirm the install without spending real quota:

```
python scripts/simulate_agent.py # fake upstream + proxy + two agents, multi-turn
python scripts/showcase.py       # every report view at once
pytest -q                       # the test suite
```