

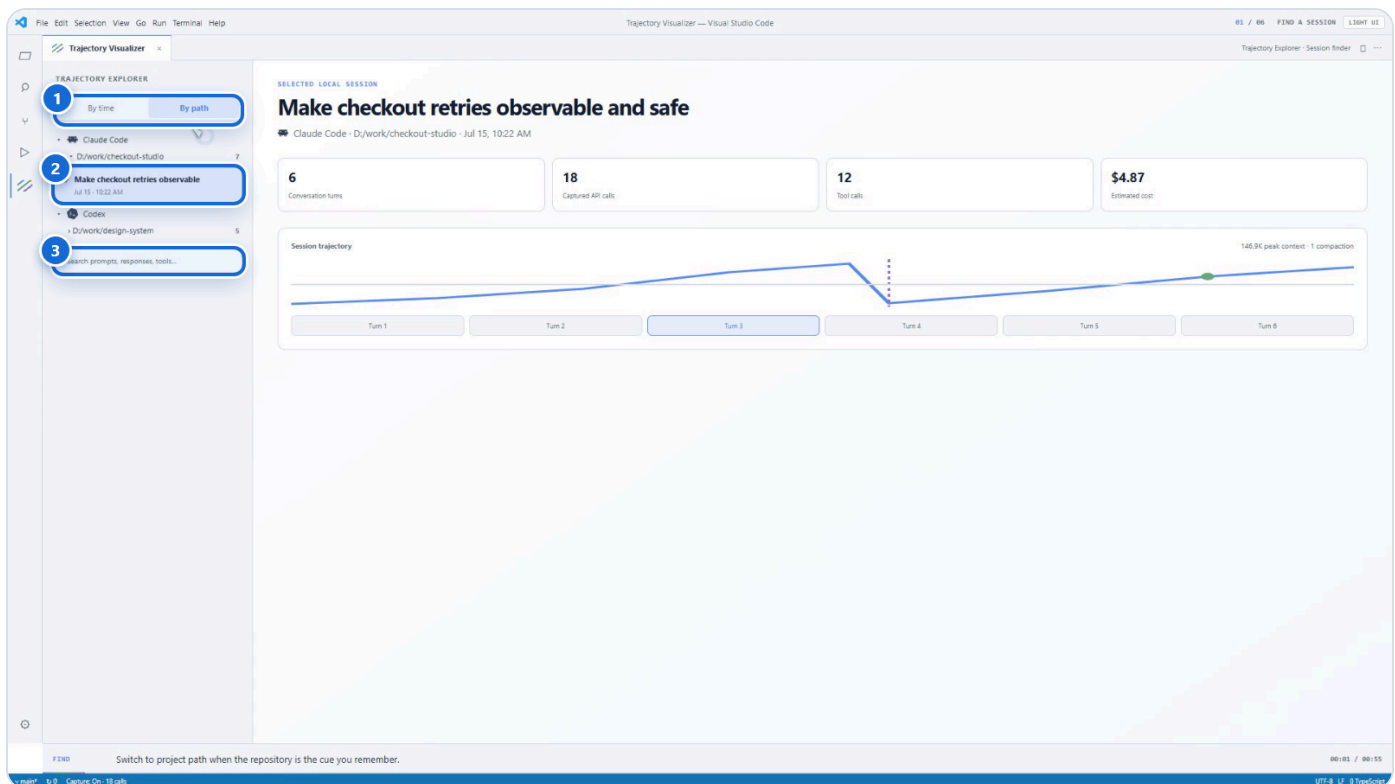
Coding Agent Trajectory Visualizer — Usage Guide

Coding Agent Trajectory Visualizer turns local Claude Code and Codex session logs into a searchable, readable record of what the agent did: prompts, final answers, intermediate reasoning messages, tools, results, compactions, subagents, context growth, and personal usage patterns.

Claude Code and Codex sessions can both be viewed. Full API Capture is an optional Claude Code feature and is off by default.

1. Find and open the right session

Open **Trajectory Visualizer** from the VS Code Activity Bar. The Trajectory Explorer can organize sessions by time or project path, and search across prompts, responses, tools, file names, and errors.



1. Choose **By time** when you remember when the work happened, or **By path** when you remember the repository.
2. Select a session to open its Timeline. Claude Code and Codex keep their own provider logos.
3. Use search when all you remember is a phrase, command, file, or error.

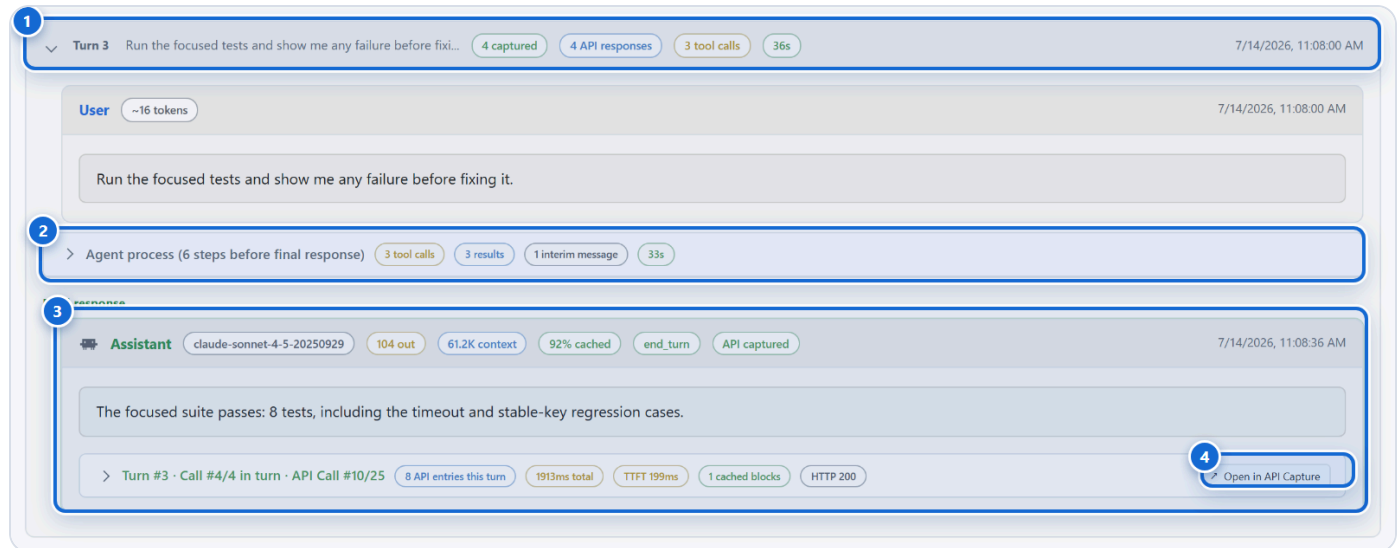
Useful sidebar commands:

- **Refresh Trajectories** rescans local session files.
- **Collapse to Default View** closes expanded sidebar groups without changing the selected grouping mode.
- **Open Usage Dashboard** opens the calendar, habits, achievements, and usage statistics.
- **Open Agent Setup** shows Memory, Skills, Hooks, and MCP across projects.
- **Open Usage Guide** opens this file at any time.

2. Read a Timeline Turn

Timeline is the default session view. The header summarizes the model, duration, Turns, tool calls, token use, estimated cost, and captured API calls. The controls can search Prompt, Response, or All text; filter by date; reverse order; collapse expanded content; reset the view; and page through long sessions.

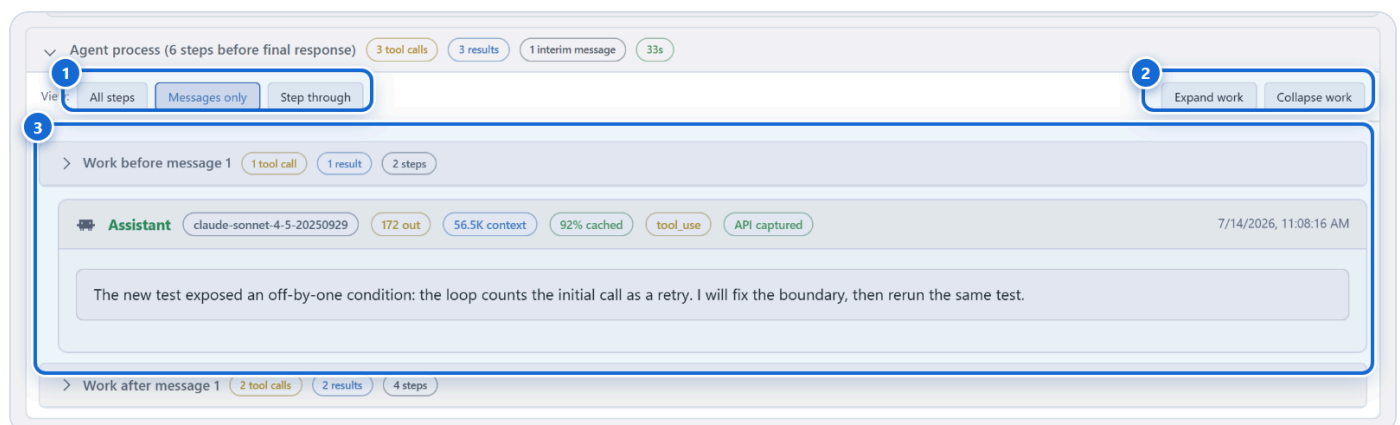
Select a Turn header to load it. Long Turns are loaded only when opened, so large sessions do not create every detail in the page at once. Images attached to Claude Code or Codex prompts appear as bounded thumbnails inside the User message; select a thumbnail to inspect the original image full screen. Image bytes follow the same on-demand Turn loading path.



1. Open or close the complete Turn from its header.
2. **Agent process** contains everything between the prompt and final answer.
3. The prompt and final response remain the quickest reading path. Use **Read full screen** on long content instead of fighting terminal scrollbar.
4. An inline captured call can open the complete request in **API Capture**.

Model, effort, output-token, context, cache, stop-reason, and API-captured badges stay beside the response that produced them. Date separators and idle-gap markers make resumed or multi-day sessions easier to follow.

Inspect the agent process



1. **Messages only** is the readable default. **All steps** shows the exact sequence, while **Step through** keeps one step visible and moves with Previous/Next.

2. Expand or collapse all grouped work in Messages-only mode.
3. Open a **Work before/after message** group to see the assistant tool call followed by its Tool Result.

The screenshot displays a user interface for viewing tool calls and their results. It is divided into three main sections, each marked with a blue circle and a number:

- Section 1:** A dropdown menu labeled "Tool: PowerShell" is expanded, showing two tabs: "Formatted" (selected) and "Raw input". The "Formatted" view displays a JSON object:

```
{  "command": "npm test -- retry.test.ts",  "timeout": 120000}
```
- Section 2:** A horizontal bar representing a captured response. It contains the text "Turn #3 · Call #1/4 in turn · API Call #7/25" followed by several performance metrics in colored pills: "2 API entries this turn" (blue), "1502ms total" (orange), "TTFT 172ms" (orange), "1 cached blocks" (green), and "HTTP 200" (blue). On the right, there is a button labeled "Open in API Capture".
- Section 3:** A "Tool Result" panel. The title bar shows "Tool Result" on the left and the timestamp "7/14/2026, 11:08:12 AM" on the right. The main content area shows a red header "Tool Result (ERROR)" and a text block:

```
FAIL test/payments/retry.test.ts
Expected attempts: 3
Received attempts: 4

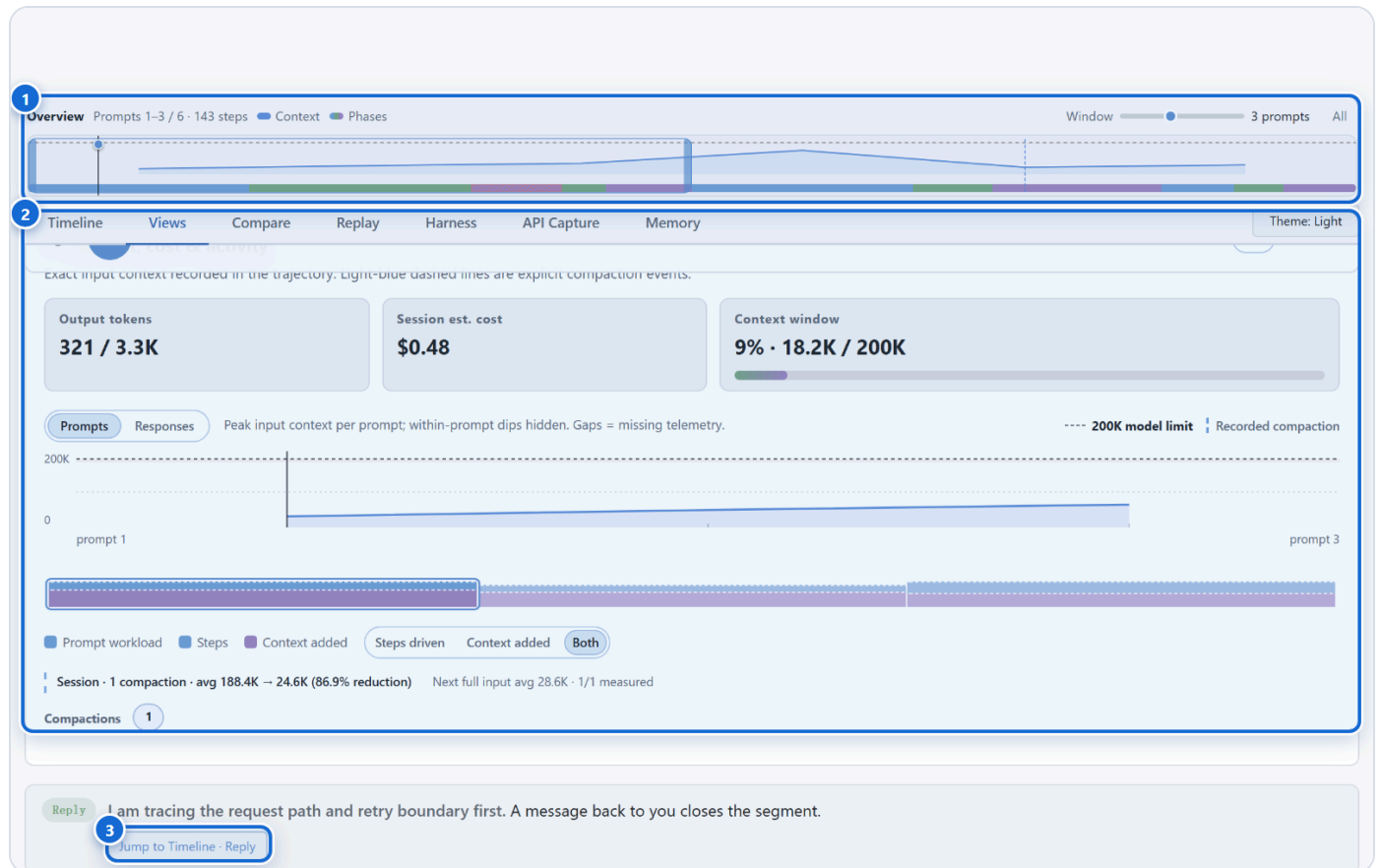
Tests: 1 failed, 7 passed
```

 The text "101 chars" is visible on the right side of the result area.

1. Tool inputs have a structured **Formatted** view and the exact **Raw input** JSON.
2. Each captured response shows **Turn #** , **Call # in turn** , and its session-wide **API Call #** . **Open in API Capture** keeps that identity.
3. Tool Results preserve command output and make failures visually distinct. Long values expose their own bounded reader or full-screen reader.

3. Understand the run with Views

Views links several visualizations to one playhead and one movable context window.



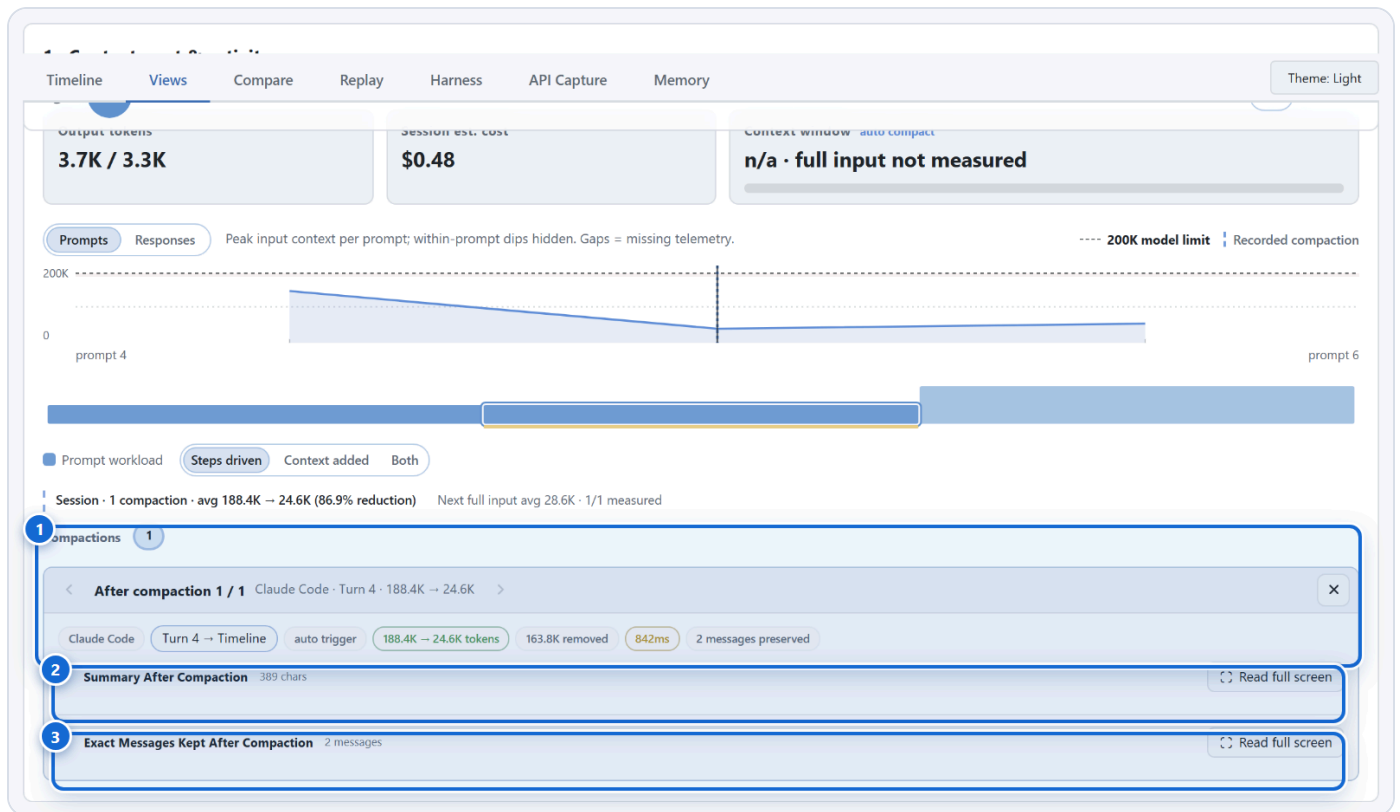
1. Play, scrub, or resize the Overview window to choose the part of the session you want to study.
2. Context and workload, compactions, Tool use, common next-tool patterns, and the Phase map all follow the same selected window. Workload can show Steps driven, Context added, or Both.
3. A selected prompt, response, tool block, failure, or phase can jump back to its exact Timeline Turn and step.

Tool-use blocks are chronological. Selecting a group keeps the surrounding history visible with a mask; selecting one exact block can reveal its matching tool call or result in Timeline.

The Phase map groups consecutive activity with fixed rules:

- **Explore:** reading and searching.
- **Implement:** edits and writes.
- **Verify:** tests and execution used to check the change.
- A prompt, reply, or rule boundary starts a new phase. Failure styling marks a phase containing an error.

Inspect a compaction



1. Select a recorded compaction to open its token reduction and source Turn.
2. For Claude Code, **Summary After Compaction** is the readable replacement summary carried into later context when Claude records it.
3. For Claude Code, **Exact Messages Kept After Compaction** are original messages preserved verbatim beside that summary.
4. For Codex, **Replacement Context** shows the readable messages retained in `replacement_history`. Codex stores the generated compaction summary as encrypted content, so the extension labels its presence and size but cannot display summary text that is not present in the local log. Each long readable value has its own full-screen reader.

This makes it possible to check what the agent retained, what it summarized, and whether important instructions survived compression.

4. Capture and inspect Claude API requests

API Capture is optional and starts **Off**. Click the VS Code status-bar item **Capture: Off** to turn it on; click **Capture: On · N calls** to turn it off. The switch is shared by every VS Code window.

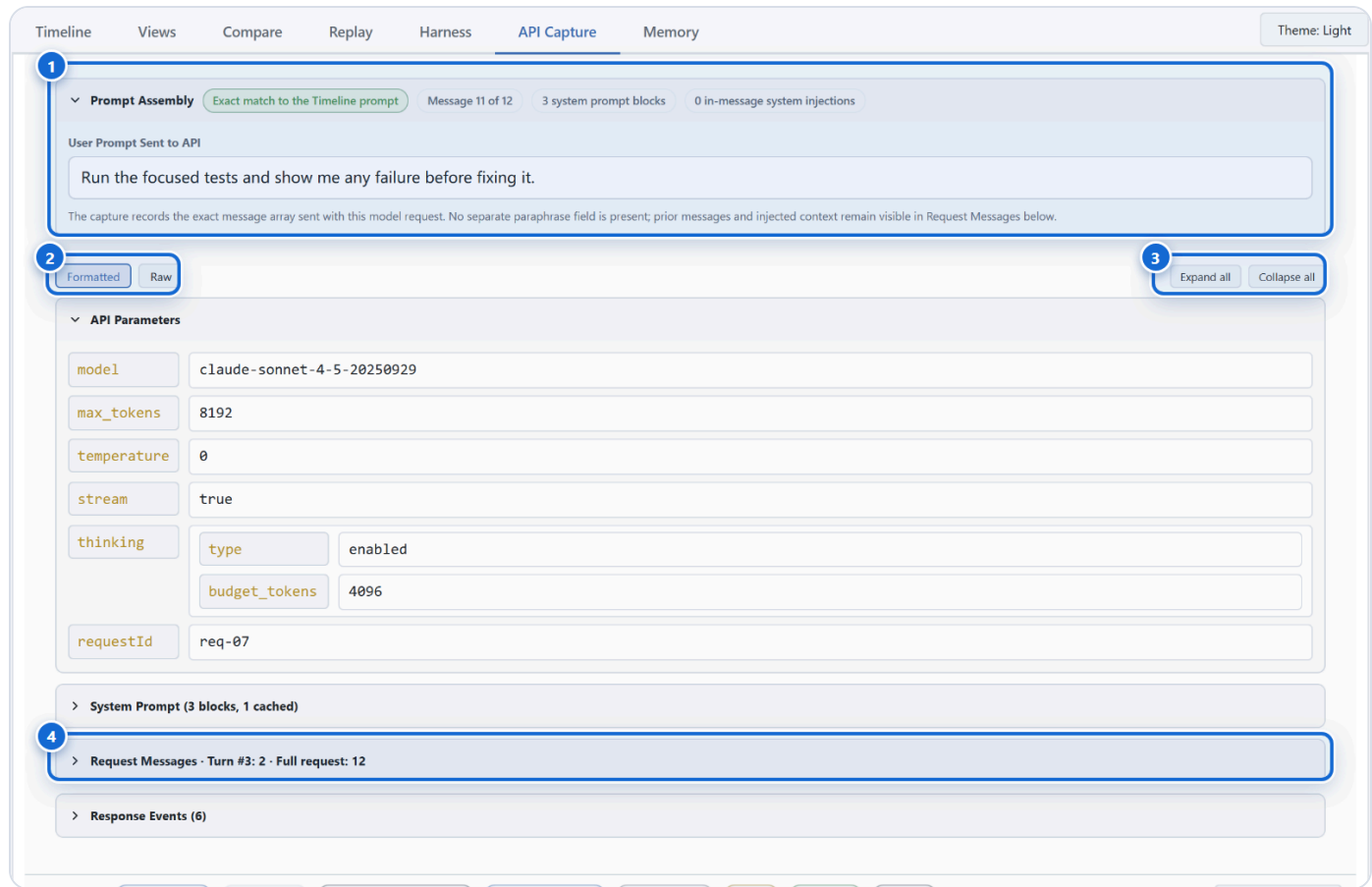
When capture turns on, new terminals receive the managed `ANTHROPIC_BASE_URL` automatically. A Claude Code process that was already running keeps its old environment, so restart that process or reload its VS Code window before expecting its calls to appear. A custom `ANTHROPIC_BASE_URL` is never overwritten.

How to verify capture:

- The status-bar counter increases while Claude Code works.
- The session shows captured-call badges and an **API Capture** tab.
- Existing recordings remain available after capture is turned off.

Captured files are stored under `~/ .claude/recorder/captures/` on the machine where the extension host runs. Codex API Capture is not available in this proxy setup; Codex Timeline, Views, Compare, Replay, Harness, Memory, and Dashboard still work.

Read one API call



1. **Prompt Assembly** identifies the user prompt matched to the Timeline and its location in the request.
2. Switch between a readable **Formatted** view and exact **Raw** JSON.
3. Expand or collapse all top-level sections. Individual blocks and key/value rows can also open independently.
4. **Request Messages** shows both the number belonging to the current Turn and the size of the complete cumulative request.

API calls are grouped chronologically by Turn; a Turn with many calls uses ten calls per page. Each call displays:

- Turn # and Call # in turn .
- Session-wide API Call # / total .
- Model, latency, TTFT, cache blocks, and HTTP status.
- API Parameters, System Prompt, Request Messages, and Response Events.
- A reverse **Turn** → **Timeline** jump to the exact response.

Timeline intentionally previews only the message slice for that Turn. API Capture is the authoritative place for the complete cumulative request. Large arrays are paged, long values use bounded scroll regions, and eligible values have their own **Read full screen** action.

System Prompt and Tool Schemas

Inside API Capture:

- **System Prompt** groups unique prompt versions, shows which calls used each version, and compares two versions with an added/removed-line diff. **Reset** returns to no comparison.
- **Tool Schemas** groups the complete tool definitions by category, including each description and raw input schema.

These two views require Claude API Capture data.

5. Compare or Replay sessions

Compare is designed for controlled comparisons. Session choices show provider/company logo, model, effort, and project tags so differences in the result are visible before playback.

The screenshot displays the 'Compare' view of the application. At the top, there are tabs for 'Timeline', 'Views', 'Compare' (selected), 'Replay', 'Harness', 'API Capture', and 'Memory'. A 'Theme: Light' button is in the top right. Below the tabs, the first session is 'Harden checkout retry behavior' by Anthropic · Claude Code, with tags 'Model · Opus 4.1', 'Effort · High', and 'checkout-service'. The second session is 'Harden checkout retry behavior' by OpenAI · Codex · Model · GPT-5.3 Codex · Effort · High. Below the sessions, a 'Compared variables' section shows 'Provider: Claude Code -> Codex', 'Model: Opus 4.1 -> GPT-5.3 Codex', 'Effort: same', and 'Project: same'. A progress bar is shown with a play button and a '1x' speed control. Below the progress bar, there are tabs for 'Align by' with 'Progress' (selected) and 'Raw step'. A note states: 'Both runs scaled to 0-100% so they start and finish together, even at different lengths.' Below this, a 'Paused' button is visible. The main content area shows a comparison of two sessions. The first session is 'Claude Code: user: Continue from the compaction summary. Guard recovery with the in'. The second session is 'Codex: user: Continue from the compaction summary. Guard recovery with the in'. Below this, a table compares metrics for 'Claude Code' and 'Codex'.

	Claude Code	Codex
Steps	22	18
Tool calls	9	8
Failures	1	▲0
Output tokens	1,960	▲1,690
Wall-clock	10.5h	▲8.8h

At the bottom, there are two tabs: 'Claude Code' and 'Codex'.

1. Choose the second session and inspect Provider, Model, Effort, and Project changes or matches.
2. Align by overall progress or raw step and play both trajectories with one control.
3. Compare steps, tools, failures, output tokens, wall-clock time, and the two phase-aware timelines.

Replay presents the same trajectory as a chat-like sequence rather than a dense analysis page.

1 Session replay
● paused

2

The retry helper owns the loop, while checkout owns the stable key. I found one unsafe bra...

Implement a bounded retry policy. Keep one idempotency key for every attempt and...

I will make the retry limit explicit and preserve the original key.

✓ Edit retry.ts

✓ Edit retry.test.ts

The implementation now keeps one key across attempts and stops after the configured retry...

Run the focused tests and show me any failure before fixing it.

✗ PowerShell npm test -- retry.test.ts

The new test exposed an off-by-one condition: the loop counts the initial call as a retry....

✓ Edit retry.ts

✓ PowerShell npm test -- retry.test.ts

The focused suite passes: 8 tests, including the timeout and stable-key regression cases.

Investigate the duplicate charge report. Do not change code yet—explain which st...

✓ Read state-machine.ts

✓ Grep submitting|recoverPendingPayment

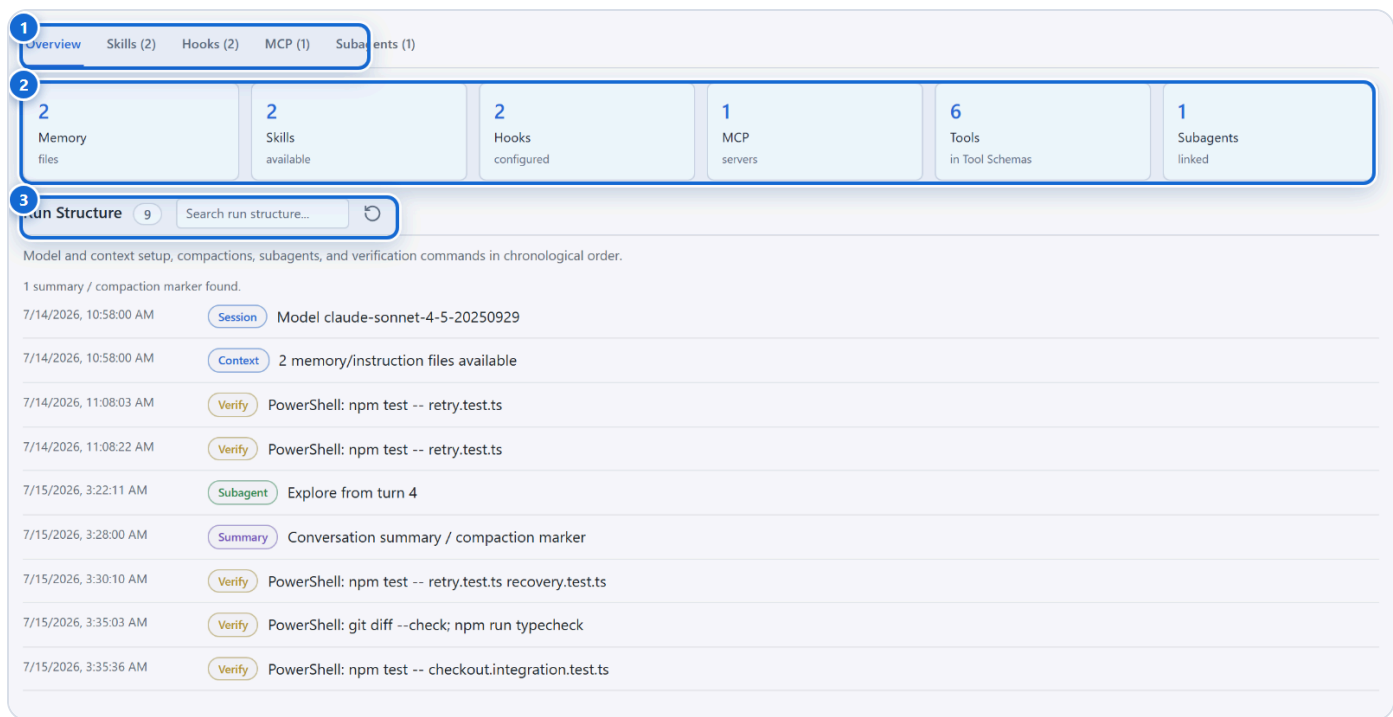
3

1x 20/35

1. The replay header shows provider and playback state.
2. Prompts, assistant messages, tools, successes, and failures appear in order.
3. Scrub, change speed, restart, pause, or resume without moving the page to another section.

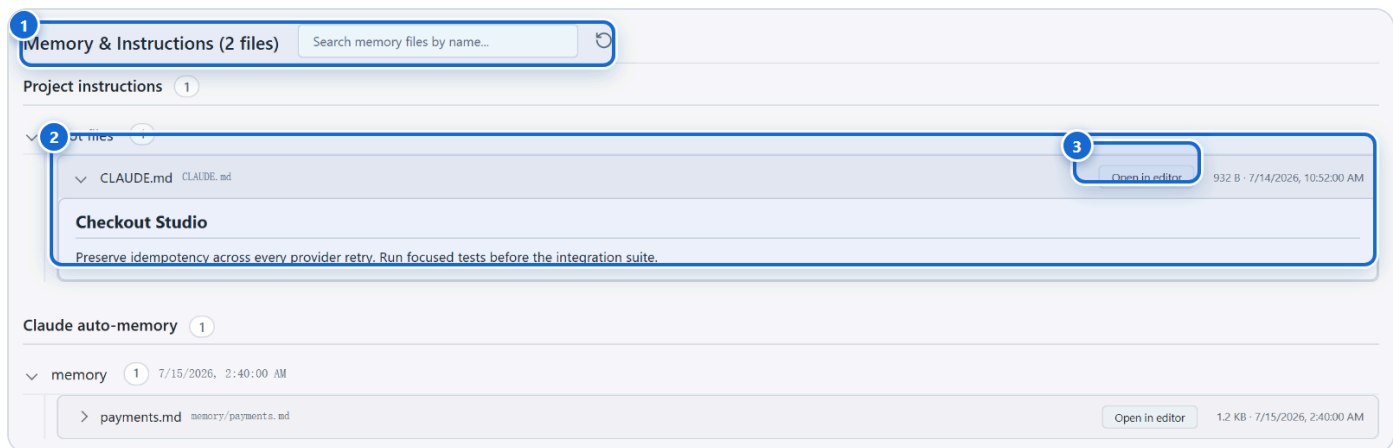
6. Inspect Harness and Memory

The session **Harness** tab answers which supporting configuration shaped this run.



1. Open Overview, Skills, Hooks, MCP, or Subagents without leaving the session.
2. Summary cards show the available Memory, Skills, Hooks, MCP servers, captured Tools, and linked Subagents.
3. Run Structure places model/context setup, compactions, subagents, and verification commands in chronological order. Each section has its own compact search and reset control; long lists are paged.

The dedicated **Memory** tab shows the instruction and memory files available to this session.



1. Search files by name and reset expanded groups from the compact toolbar.
2. Browse project instructions, Claude auto-memory, Codex memory when enabled, and relevant Markdown read during the session.
3. **Open in editor** jumps to the real file so it can be reviewed or changed.

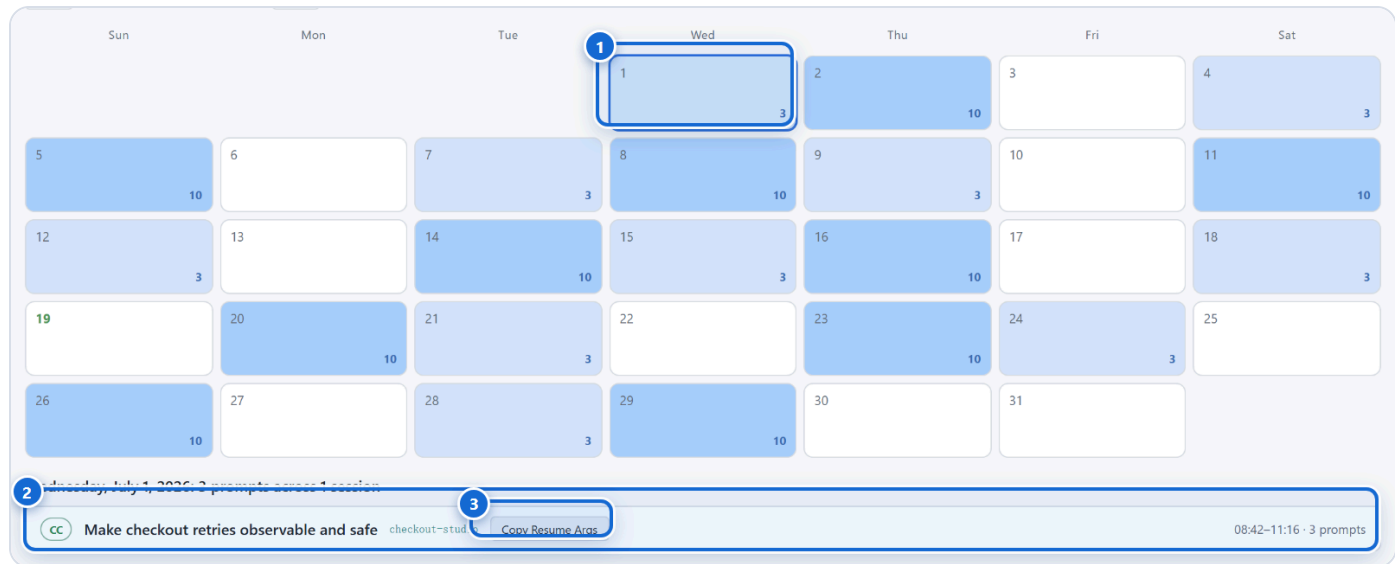
Use **Trajectory: Open Agent Setup** for the cross-session library of Memory, Skills, Hooks, and MCP. It is the convenient place to inspect the files and integrations that apply beyond one session.

7. Dashboard, calendar, and resume

Run **Trajectory: Open Usage Dashboard** from the sidebar calendar icon or Command Palette. It summarizes Sessions, active days, streaks, Turns, prompts, tool calls, output tokens, estimated Claude cost, Runtime Rank, achievements, response times, model/effort

usage, hourly habits, and frequent words.

Tables sort only on meaningful metric columns; a chevron appears after a sortable heading is selected. Runtime Rank is based on recorded work time, with progressively larger level requirements.



1. Select an active date to list every session edited that day.
2. Open the session directly from the calendar result.
3. **Copy Resume Args** copies only the argument needed to resume (`--resume <id>` for Claude Code or `resume <id>` for Codex); the long command is not printed beside the button.

8. Export and manage long sessions

The session header provides:

- **Export Unified JSON:** writes the merged session and linked capture data to `~/.claude/recorder/exports/`.
- **Export Markdown:** writes a readable transcript of prompts, process summaries, and final responses.
- **Update available:** when a live session changes while you are reading, the panel waits for you to request the update instead of forcing a reload and moving your scroll position.

Timeline, API calls, Request Messages, System Prompt versions, Memory, Skills, Hooks, MCP, and Subagents use pagination or bounded readers where needed. Previous, page number, and Next stay together so navigation does not require moving across the full page width.

9. Quick troubleshooting

A session is missing

Use Refresh Trajectories, switch between time and path grouping, then search for a phrase from the prompt or response. Remember that Remote SSH sessions live on the remote extension host.

Capture says On but no new calls appear

Restart the Claude Code process that was already running when capture was enabled. Confirm the status-bar count increases. If the status bar says **Custom endpoint**, clear the custom `ANTHROPIC_BASE_URL` before using the managed capture proxy.

A Turn or API call is very large

Open only the level you need. Turn bodies, inline API details, Memory, and Harness are loaded on demand. Use section collapse, pagination, bounded readers, and Read full screen rather than expanding every nested value.

Where are exported files?

- Windows: `%USERPROFILE%\.claude\recorder\exports\`
- macOS/Linux: `~/.claude/recorder/exports/`
- Remote SSH: the same path on the remote machine, not the local laptop.