ASHITA ORBIS

The Saving Was per Successful Call

On 24 August 2026 a Claude Code session drove Browser Use's command line tool (release 0.13.8, on a local headless Chrome 151) and a Playwright MCP server that answers with accessibility tree snapshots through the same eight public web tasks. By the driver's tally the Browser Use tool needed about 21 calls against about 47 and printed 26 to 340 bytes per successful result (858 for the form task), though its median time per operation ran roughly nine to ten times the snapshot server's, an upper bound; over the arm as run, its replies came to 23,011 characters against 35,218, and the 28,654 characters of commands the driver wrote to run its programs took its total past the snapshot arm's.

OCTOBER 23, 2026 · LAB-NOTES · ASHITAORBIS.COM/POSTS/BROWSER-USE-ROUND-TRIPS

LAB-NOTES · BROWSER AUTOMATION · AGENT LOOPS · CONTEXT BUDGET · TOOL CALLS · BENCHMARK METHOD

On 24 August 2026 a Claude Code session in this workspace drove two browser tools through eight frozen tasks on public websites, whose answers, all but task 6's, the trial's record says were fixed with curl beforehand. One was the command line tool of Browser Use, which takes a short Python program piped into it and runs it against Chrome over the DevTools protocol. The release tested was 0.13.8, on version 0.1.9 of its harness, with a local headless Chrome 151. The other was a Playwright server for the Model Context Protocol (MCP) that answers with snapshots of the page's accessibility tree. Each tool figure below measures those releases on that day, and the trial's report records one driver and one pass.

The Browser Use tool passed 7 of 8 tasks on its local headless browser, and the snapshot server passed 8 of 8 by the trial's report, though no program graded that arm's replies. The one failure was task 6, a public post on X, where the local browser received an HTTP 403 and the same tool's cloud browser was served the page. The total of 7 therefore counts against the local mode alone. The vendor's autonomous cloud agent also ran three of the eight tasks, and its run recorded each session as successful.

The trade per call

By the driver's own tally on that single pass, the Browser Use tool needed about 21 calls against about 47 for the snapshot server. Each successful result's final program printed 26 to 340 bytes (858 for the form task), because the filtering ran inside the driver's program and that program printed its filtered result rather than the page. The snapshot server's replies ran, by the trial's report, to hundreds or thousands of tokens per task, so with the units differing the comparison supports at most an order of magnitude per successful result.

Per call, the timing favoured the snapshot server. Through the Browser Use tool, opening a catalogue page and extracting its prices took about 450 milliseconds, of which an empty invocation of the tool alone took about 130, against a median of 44 to 52 milliseconds through the snapshot server's HTTP API. That is roughly nine to ten times slower per operation, an upper bound, since the snapshot timing omits the MCP wrapper hop of the server's real path. The trial's report weighs this against the agent loop, where it assumes each call also costs a model turn, which it takes to be far slower than either tool. On that reading the number of round trips matters more end to end than the speed of each call, and since no model turn was timed, it remains a reading.

The arm as run

By the report's token estimates, the saving holds per successful call, and on this run the failed attempts, the debugging probes and the programs themselves spent it. Over the arm as run, every attempt and probe included, the Browser Use replies came to 23,011 characters against 35,218 for the snapshot arm. The commands the driver wrote to pipe in those Python programs added 28,654 characters more, which put the arm's total at 1.40 times the snapshot arm's. When the program has to be found by trial, every rejected program and its reply land in the context, as they did on the arXiv search chain and the infinite scroll.

The snapshot arm ran second, after the driver had met the sites' quirks (the arXiv search toggle and the scroll page's loader), so an order effect in its favour is not ruled out. The method and results page gives each task's row, both timed sets, the session's own call counts and the defects that decided a task, as observations at release 0.13.8.

The record also shows where that text sat: the snapshot arm's arguments came to 1,814 characters against 35,218 of replies, while the Browser Use arm's commands, which carried its programs, came to 28,654 characters, more than the 23,011 characters of replies they drew. On this run, a tool that answers through a program the agent writes moved the larger share of its text from the tool's replies into the agent's own output, which a comparison of reply sizes alone does not count.