INDEPENDENT PRODUCT  2025–2026  AI BROWSER · AGENT TOOLING

# A browser-native agent with 3–10× token savings

Most browser agents send the page to the model. Tappi indexes the page, references the element, and sends only what the model needs.



**Written by Azeruddin Sheikh**

INDEPENDENT PRODUCT · 2025–2026

LINKEDIN ↗

## Context

Browser agents are usually built by giving a model a screenshot, a DOM dump, or a set of coordinates. All three are expensive. A single page can be 100KB of HTML; a model that reads the whole page on every step pays for navigation, repeated markup, and content it will never use.

Tappi Browser started from a different assumption: the browser already knows the page structure. The agent should ask the browser for a *reference* to the element it needs, not for the page.

## The problem

The agent loop had four constraints:

**Token cost.** Reading the DOM or a screenshot on every step makes long sessions expensive and slow.

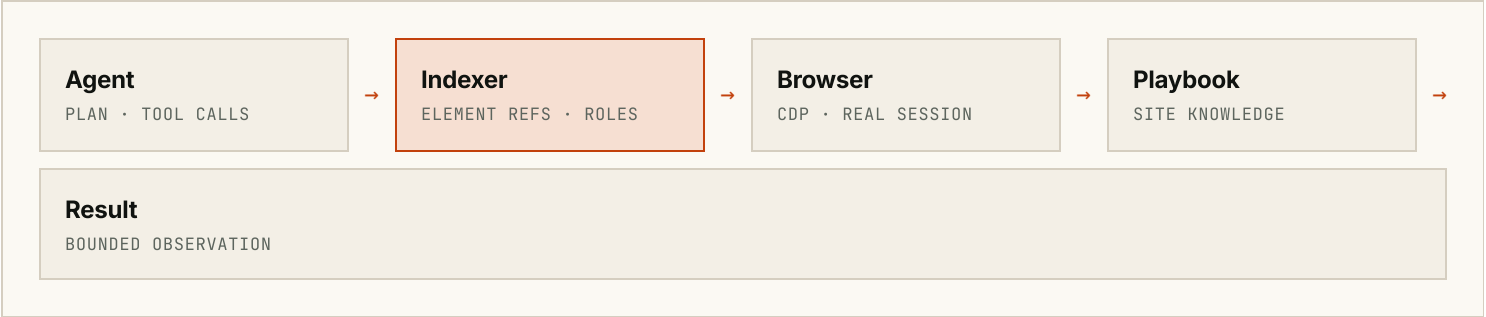
**Brittleness.** Coordinates break when the layout shifts. Selectors break when the page changes. Screenshots force the model to do geometry.

**Detection.** Headless automation leaves fingerprints. A browser agent that is visibly automated cannot work in the sessions where the user is already signed in.

**Learning.** Most agents start from zero on every site. A domain that is used every day should get cheaper and more reliable over time.

## Architecture

Tappi runs as a Chromium-based browser, so the agent and the page share the same session. A content-side indexer assigns stable references to interactive elements. The agent calls a small, typed tool surface — click, type, read, select — with a reference instead of a selector or a coordinate.




SIMPLIFIED. THE PRODUCTION SYSTEM CARRIES MORE ADAPTERS AND FALLBACKS THAN SHOWN.

The key decisions:

**Reference, don't describe.** The model receives `[ref=e42] button "Submit"` rather than a DOM subtree or a coordinate. The browser resolves the reference at action time, so the model is not asked to do layout math.

**Bound the observation.** A tool result returns the element the agent asked for, plus a small amount of surrounding state. It does not return the page.

**Use the real browser.** The agent drives the user's signed-in Chromium session through CDP, not a separate headless profile. That avoids `navigator.webdriver` and keeps cookies, SSO, and extensions intact.

**Learn per domain.** Playbooks record the steps that worked on a site, so repeated workflows start from a known path instead of a fresh exploration.

**Keep a harness-agnostic core.** The same browser-control surface is exposed to more than one agent harness, so the product is not coupled to one model vendor.

## What I did

- Built the browser-native agent and the element-indexing layer that gives the model a bounded, reference-based view of the page.
- Built the CDP control layer: navigation, tabs, frames, shadow roots, typing, selection, uploads, dialogs, and screenshots.
- Built the domain-playbook system and the persistent learning loop.
- Built the harness adapters and the packaging for a desktop product.
- Measured token usage against conventional browser-tool baselines and kept the comparison reproducible.

## Results

| MEASURE | RESULT |
| --- | --- |
| Token usage vs. conventional browser tools | **3–10× fewer** |
| Automation footprint | **No** `navigator.webdriver` |
| Codebase | **~85K LOC** |
| Session model | **User's real signed-in browser** |
| Learning | **Per-domain playbooks persist** |

The token range depends on the page and the task. A dense page with a small target element gives the largest saving; a page that is mostly one form gives less. The claim is a range, not a single number, because the baseline is a range.

| 3–10× | ~85K | 0 | Real |
| --- | --- | --- | --- |
| FEWER TOKENS | LOC | WEBDRIVER FLAGS | SIGNED-IN SESSION |

## What it proves

“The agent stopped and nobody knows why” is usually a design choice, not a model limitation. If the agent receives an unbounded page and an untyped action surface, failures are inevitable and hard to name. If it receives a small, referenced view and a typed tool surface, failures become specific and fixable.

The open-core SDK and architecture notes are in tappi-goes-bonsai. The commercial browser stays private.

---

AT A GLANCE

HEADLINE RESULT

3–10×

fewer tokens than browser-tool baselines

LABEL

Independent product

PERIOD

2025–2026

CONTEXT

AI browser · agent tooling

STACK

Electron + Chromium | CDP | Element indexing | Domain playbooks | Claude Code integration

---

**Download PDF →**

**Discuss a similar problem →**

| | |
|---|---|
| 3–10× TOKEN REDUCTION | ~85K LINES OF CODE |
| 0 | Persistent |

NAVIGATOR.WEBDRIVER FLAGS

DOMAIN LEARNING

NEXT CASE STUDY

Pikki: a live, paid Chrome extension

All work →