07 NOVEMBER 5, 2026 3 PP

Copper Cloud and the tenth-of-a-second browser step

Felipe Arce · Collin Rijock · Tim Child

How we built Copper Cloud, one Rust server that syncs browsers, pairs devices and relays agent calls to a phone, and how handing whole goals to a small action model brought a browser decision down to about 0.13 seconds.

COPPER COPPER CLOUD RUST MCP AGENTS LATENCY BROAD-DAYLIGHT OPEN-SOURCE



At 19:47 UTC on October 31 we asked one more thing of Copper: let agents drive the browser on an iPhone. Copper is Collin Rijock and Felipe Arce's WebKit browser and also an MCP server, so Claude Code or phi can work in the browser you are already signed into. It is open source under Broad Daylight, Exowatt's open-source initiative, and the browser and its server live in the public copper-browser organization on GitHub. On a Mac that server listens on loopback, and a phone can't offer that: no app on an iPhone accepts inbound connections, and iOS suspends an app seconds after it leaves the screen. At 21:14 the server half shipped as copper-cloud 0.13.0, and at 23:15 the iPhone half merged. In the first full run, an agent on a laptop gave Copper on the phone a goal, and Jev searched Wikipedia and stopped on the right article in 3.0 seconds.

That took three and a half hours because the server already existed. Felipe wrote Copper Cloud in October: one Rust binary and one Postgres database, sixteen

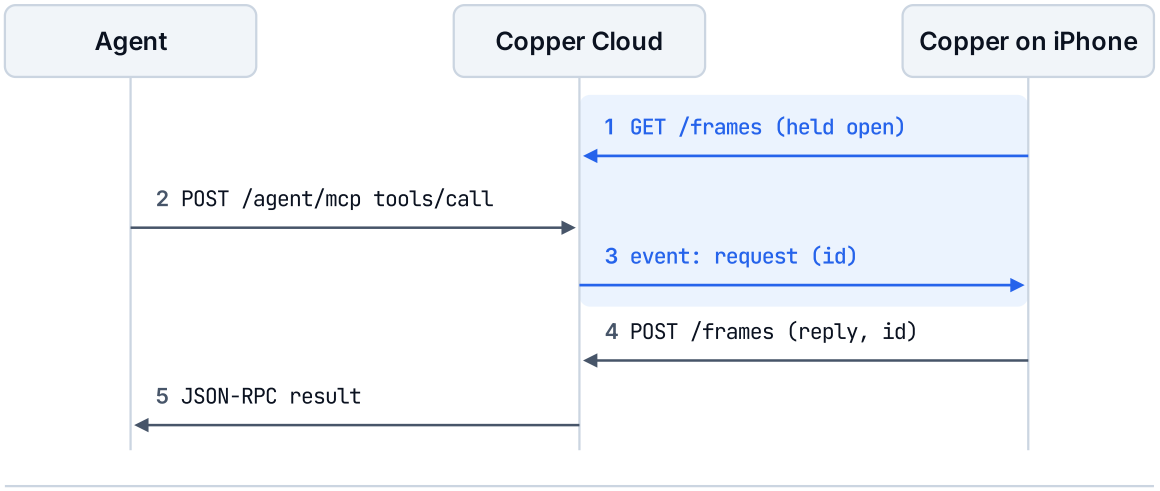
tagged versions between October 1 and October 31, and about 98% of its 52,700 lines of server code. It syncs browsers, pairs devices and serves the org’s model keys, and now it relays agent calls to phones. Speed was the other problem. Handing a whole goal to Jev, a small action model from TypeSafe, cut an agent’s time to its first useful browser action from 12.6 to 3.3 seconds.

A server with very little state

Browser state is small, and one device owns most of it at a time, so we chose the dullest sync model that would work. Spaces, settings and bookmarks are whole documents with a version number. A write names the version it was based on, and the server applies it with a single `UPDATE … WHERE version = $base`. If another writer got there first, the client gets a 409 with the current copy and merges. Each user gets one in-process server-sent-event stream with a 15-second keepalive and a `resync` event for clients that fall behind. The server seals each stored blob with AES-256-GCM under a per-user key, with the user id and domain as associated data, so rows can’t move between users.

Linking produced our most expensive mistake. Most instances run on a raw IP with a self-signed certificate, so the link code carries the certificate’s SHA-256 fingerprint and Copper pins it. Early on, an upgrade replaced the VM, minted a new certificate, and broke every linked Copper at once. Upgrades now swap the binary in place and keep the certificate. In 0.9.0 we added hybrid TLS: clients that ask for the domain get a Let’s Encrypt certificate, and the rest keep the old self-signed one, which the server never regenerates.

A mailbox for the phone



Stream drops before step 4: the next stream gets the same request id again.
The phone answers an id it has already seen with its stored reply.

Figure 1. The device agent relay in copper-cloud 0.13. The phone dials out and holds one event stream (blue); the cloud matches each agent call to it by request id. Nothing listens on the phone, and nothing about a call is stored.

There are two ways to reach a phone. Push notifications can wake an app, but delivery is best-effort and the app still has to fetch the work. We chose a mailbox. The phone registers a link with its tool manifest when the person turns agents on, so an agent can list tools even while the phone is closed. While Copper is on screen, the phone holds one event stream open over ordinary outbound HTTPS. An agent calls a standard MCP endpoint with a 256-bit token minted on the phone, and the server stores only its hash. The server hands the call down the stream as a `request` event with a 200-second deadline, waits on a one-shot channel keyed by request id, and returns the phone's reply. If the phone is off screen, the agent gets an error at once.

What broke

A review of 0.13.0 found a hang. If a stream died before the phone replied, the request sat there for the full 200 seconds. Phones drop streams often, on a network change or a moment in the background, and we had tied each call to one stream. In 0.13.1, shipped 87 minutes later, we made the request id the unit of delivery. A call stays pending when its stream ends, and each new stream gets the pending requests again right after its `hello` event. The phone answers any id it has already handled with its stored reply, which makes re-delivery safe. The regression test drops a stream mid-call and gets the answer in under 10 seconds of a 60-second deadline.

Where the time goes

On September 23 we ported browser-use's jev-ultrafast loop to Swift so it runs inside the open tab. Each step reads the visible controls into a numbered table. One request to Jev returns the operation plus a speculative target row for each operation, and Copper acts only on a row it listed itself. The next morning we measured why agents still felt slow. Copper's server answered `tools/list` in 1 ms. The frontier model used the rest: each turn took 3.5 to 5 seconds even for a trivial choice, and the agent spent three turns looking around before its first useful call, 12.6 seconds in all. The fix makes the first call `jev_run` with the whole goal, which takes one turn and 3.3 seconds, so the large model pays one turn per goal.

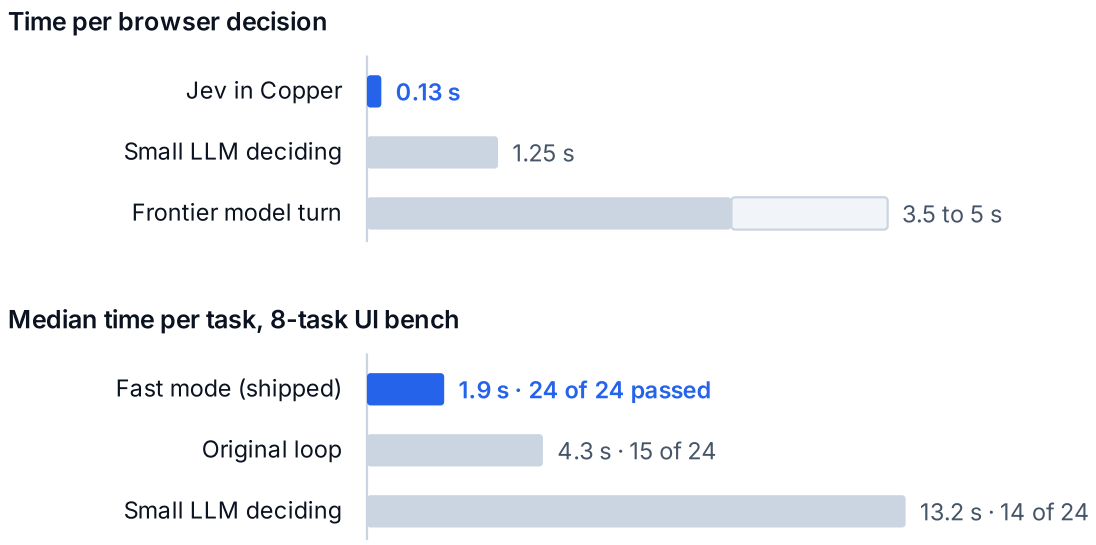


Figure 2. Decision and task latency; the two panels use different scales (0 to 5 s and 0 to 14 s). The frontier turn is from our dispatch measurement on September 24; the rest are from Collin's UI bench on October 16, with Haiku as the small LLM and 24 runs per arm.

Collin, who built the Copper for iOS app, then measured the loop itself. Jev decided in 110 to 130 ms, but the original loop spent 120 to 320 ms per click on a cosmetic pointer glide and 0.7 to 1.1 seconds per typed field on a second model. His Fast mode drops the glide, lets Jev type values the goal quotes, fills whole forms in one decision, and waits on DOM changes in place of a fixed sleep. It ships as the default and passes 24 of 24 tasks at a 1.9-second median, against 15 of 24 at 4.3 seconds for the loop we ported.

Impact and what is next

An agent can now take a goal and finish it in a browser in about two seconds, on a Mac or on a phone in someone’s hand, behind one server we can rebuild from a tarball. The relay’s hub lives in memory in one process, so a second daemon behind a load balancer would need it moved into Postgres or sticky routing. The phone must stay on screen. Jev gives up too early on long pages: in the Wikipedia bench, 19 of 23 failures were Jev answering BLOCKED after a scroll or two, so the next thing we would try is a guard that refuses BLOCKED while scrolling is still possible.

Copper is built on Search by Office Commun.