

Hand the Agent the Clicker

Designing developer tools for humans and
agents at the same time



WHOAMI

Hi, I'm Giovanni

Giovanni Laquidara

Developer Advocate at Amazon

<https://giolaq.dev/>

github.com/giolaq

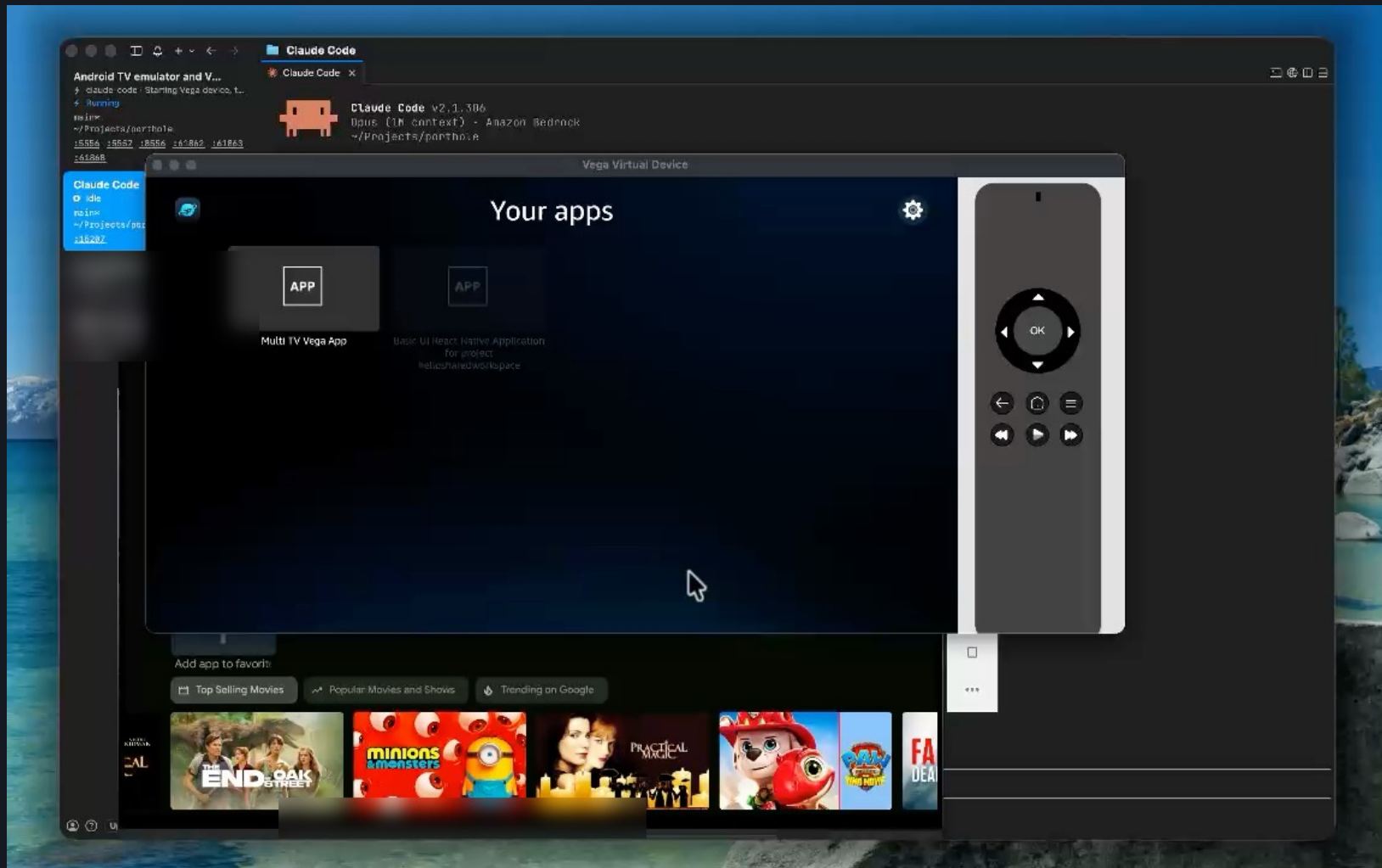


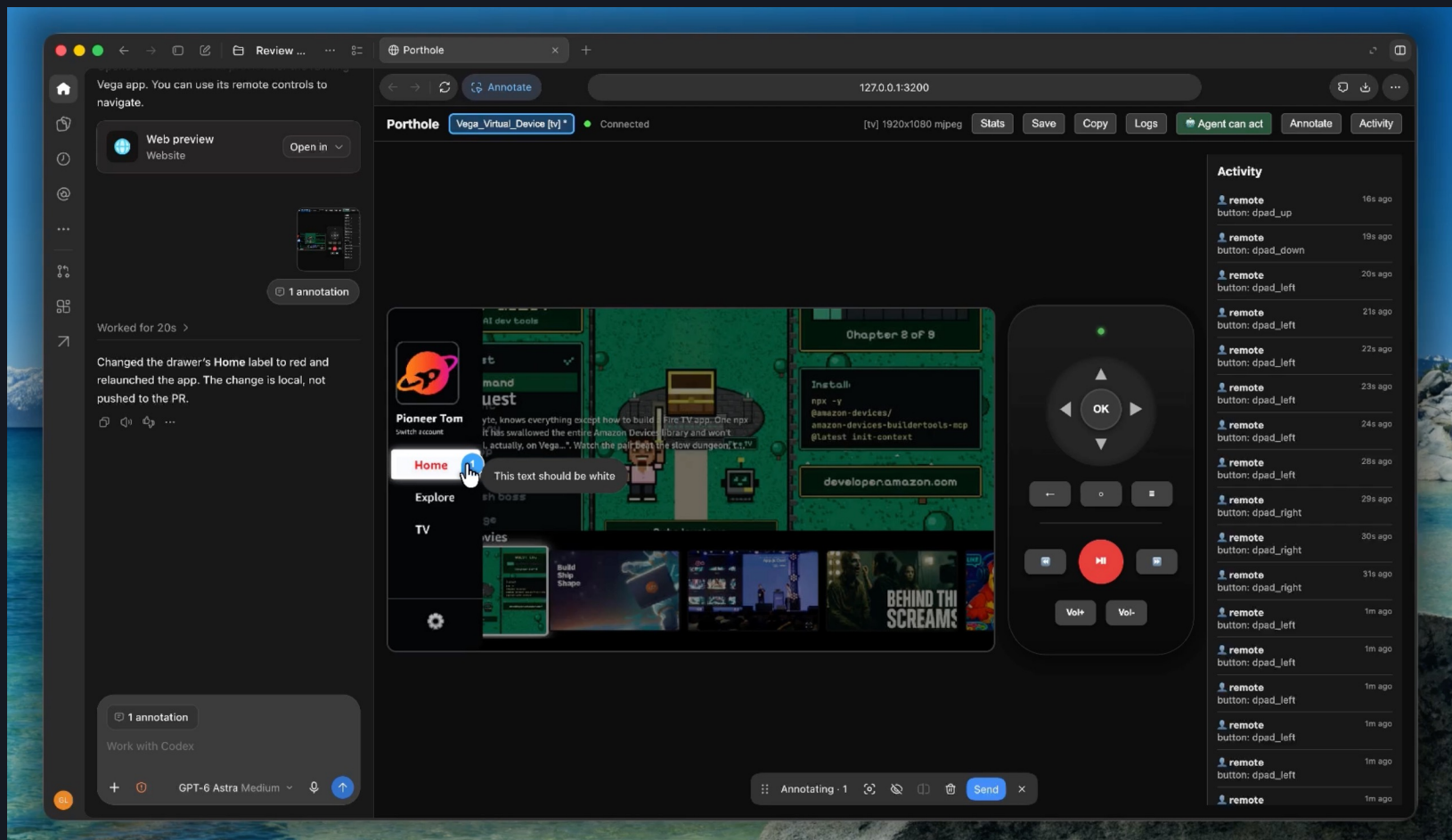
I Work ON

Fire OS



Vega OS



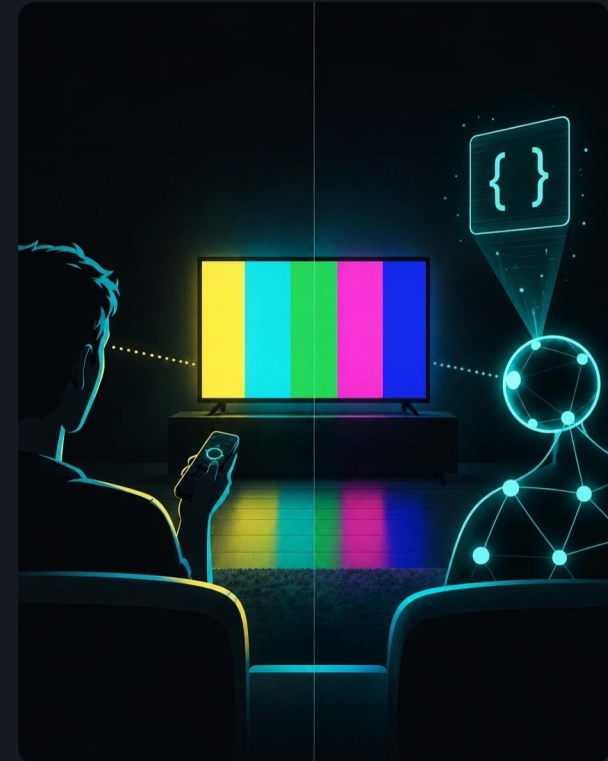


THE PROBLEM

Humans and agents using same tools

task: *“Move focus to Interstellar and open it.”*

Human	Agent
Sees the screen	Sees everything (exposed)
Presses a button	Calls a tool
Slow	Fast
Controls	Guardrails

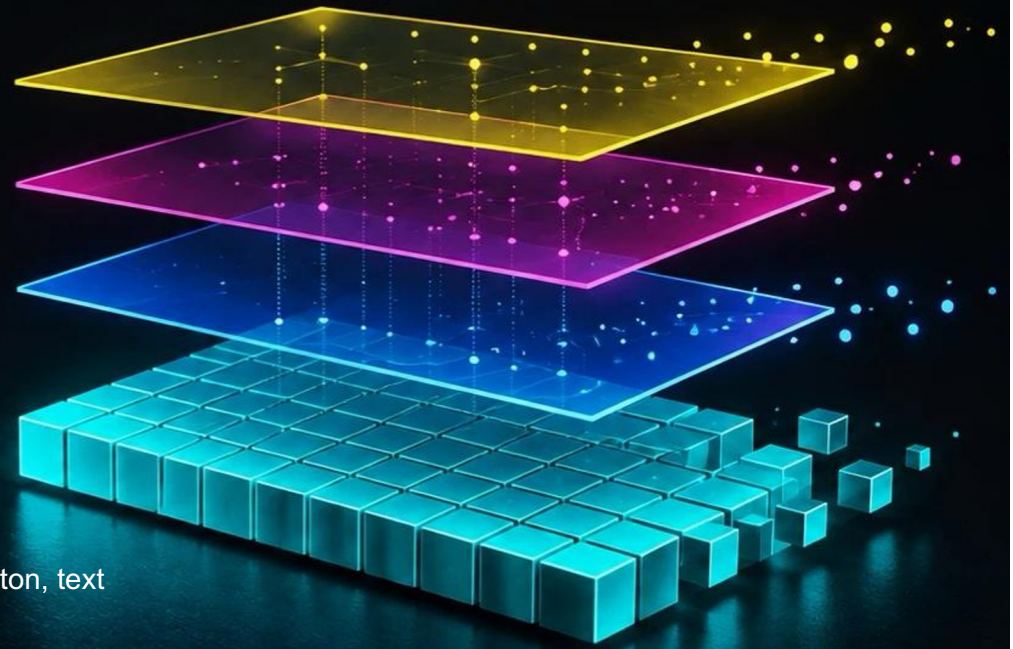


THE ARCHITECTURE

Three surfaces

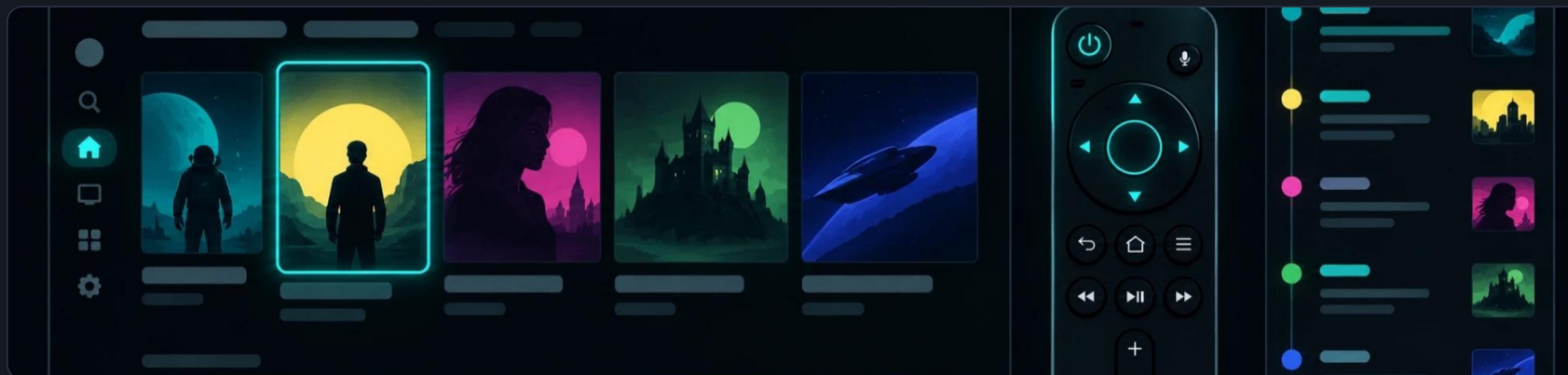
One set of primitives

sendInput: touch, gesture (swipe or long press), key, remote button, text
launch, stopApp, clearApp, installApk, pushFilestartRecording,
stopRecording.....



THE CASE STUDY

Porthole



Browser

For both: live stream, D-pad remote,
logcat

CLI

```
porthole remote dpad_down -q
```

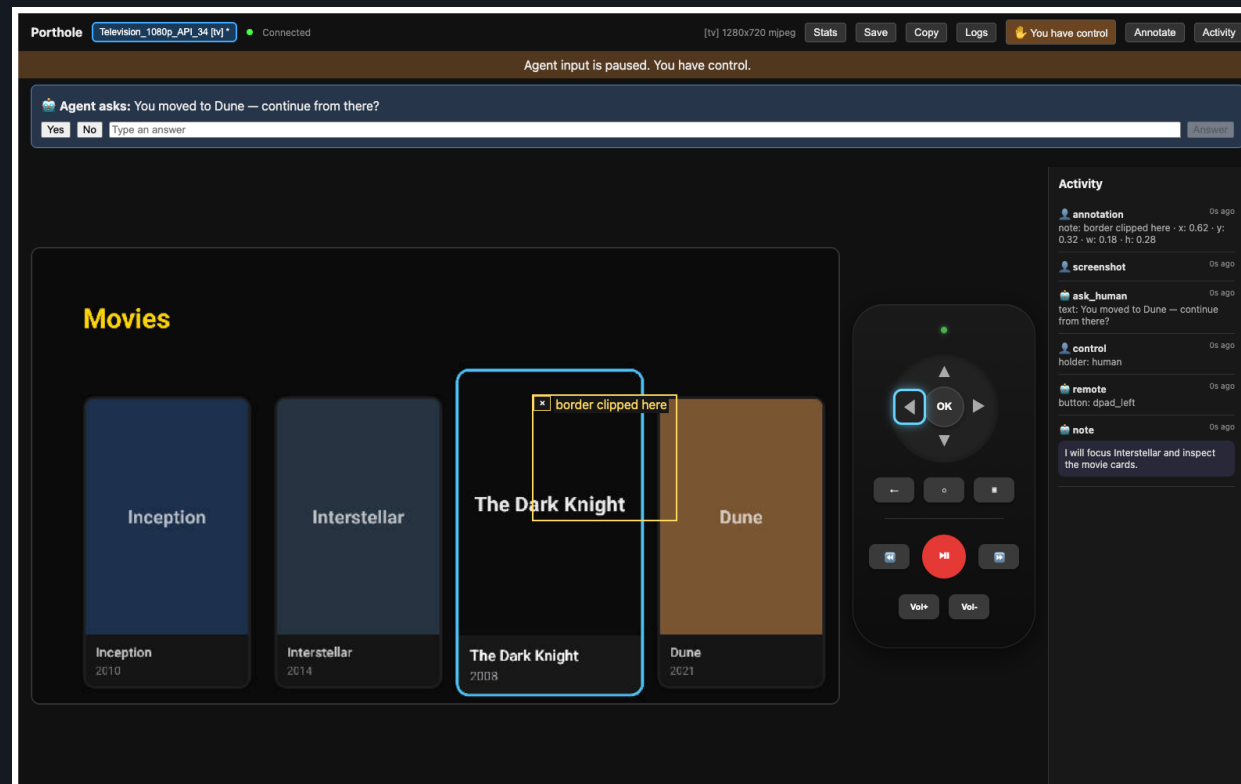
MCP server

```
claude mcp add porthole -- npx  
portholejs mcp
```

```
$ npx portholejs start Android_TV_1080p
```

THE CASE STUDY

Browser



THE CASE STUDY

CLI

Group	Commands	● ● ● porthole ... -q · one JSON object per call
Set up	start <avd> list · doctor	\$ porthole remote dpad_right -q { "ok": true, "button": "dpad_right" } ACT
Act	remote <button> tap <x> <y> · text "<s>" launch <package>	\$ porthole focused -q { "ok": true, "node": { "class": "android.widget.Button", "text": "Details", "focused": true, "resourceId": "...tvlauncher:id/tray_see_more" } } VERIFY
Verify	focused · dump-ui wait-for <text> screenshot · assert-screen	\$ porthole wait-for "Favorite Apps" -q { "error": "Timed out waiting for text: Favorite Apps" } # exit code 1 FAIL
TV	focus-on <text> --select	
Share	activity · control ask "<question>"	\$ porthole activity -q { "actor": { "kind": "cli", "action": "remote", "outcome": "ok", "durationMs": 1087 } { "actor": { "kind": "cli", "action": "wait_for", "outcome": "error", "error": { "code": "TIMEOUT", ... } } EVIDENCE

THE CASE STUDY

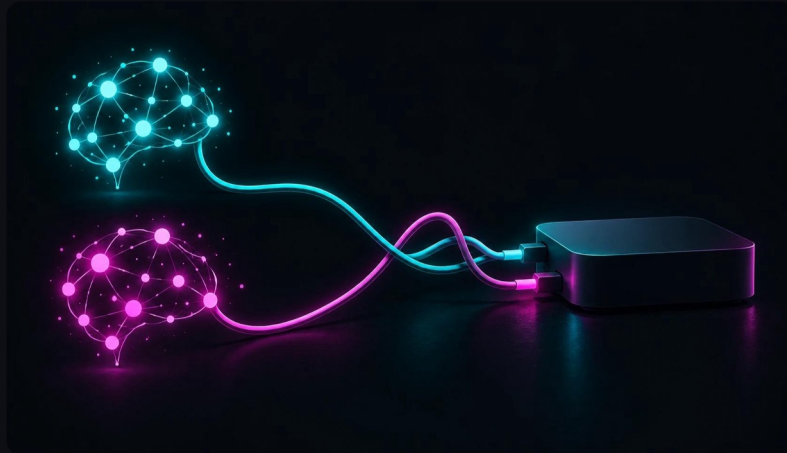
MCP

Group	Tools
Lifecycle (4)	<code>list_devices, boot_device, wait_for_boot, attach_device</code>
Input (7)	<code>tap, swipe, long_press, scroll, key, remote, type_text</code>
Screen (4)	<code>screenshot, assert_screen, start_recording, stop_recording</code>
UI (5)	<code>dump_ui, get_focused, focus_on, find_element, wait_for</code>
Apps (4)	<code>open_url, stop_app, clear_app, install_apk</code>
Logs (2)	<code>read_logcat, get_crashes</code>
Collaboration (5)	<code>get_recent_activity, get_control_state, ask_human, get_annotations, say</code>

`mcp/server.ts`

THE FIRST VERSION

The first architecture



Brain 1 • Browser + CLI

Browser and CLI → Session → Engine

Brain 2 • the MCP server

Separate state and a separate engine

One device

Connected to by both engines

This creates some issues...

THE FIRST VERSION

Issues



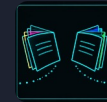
Invisible agent

Nothing in the browser showed what the agent did.



Conflicts

MCP could push a second scrapy server to an emulator the browser was already streaming.

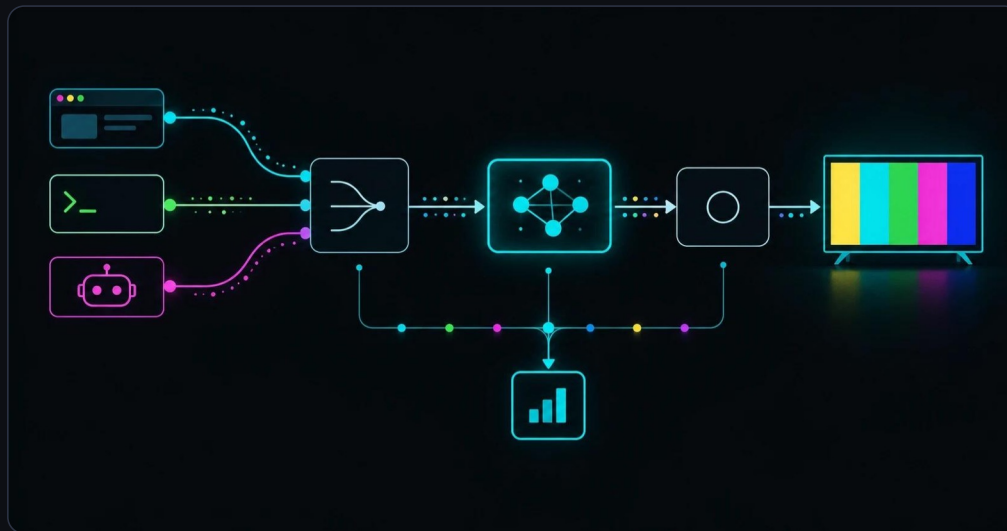


Two rulebooks

Validation, error handling, and multi-device support existed in two places and drifted.

THE FIX

One session for all the surfaces



surfaces → routes → actions → session → device

every step → ActivityBus → browser feed

Surfaces translate

Input/output

Actions decide

Validation, policy, device behavior

The session owns the device

one engine per device connection

UNDER THE HOOD • ENGINE/TYPES.TS

Swappable Engines

Above the plug: routes, actions, session, MCP server, browser



```
interface Engine {  
  start() · stop() · sendInput(event) · screenshot() · onVideoChunk(cb)  
}
```

scrcpy engine • Android

Pushes a small Java server onto the emulator over adb and streams H.264 video back.

Vega engine • Fire TV

Sends remote keys through QEMU and Fire TV's own input tool.

The MCP server joins a session

01

Agent calls
attach_device



02

Look up a running
session in the state file



03

None found: start one
in the background



04

Call it over HTTP,
tagged as the agent

```
async attach(serial, port) {  
  try {  
    this.current = await discoverSession({ device: serial, port });  
  } catch {  
    await startDetachedSession({ devices: [serial], port, quiet: true });  
    this.current = await discoverSession({ device: serial, port });  
  }  
}
```

THE FIX

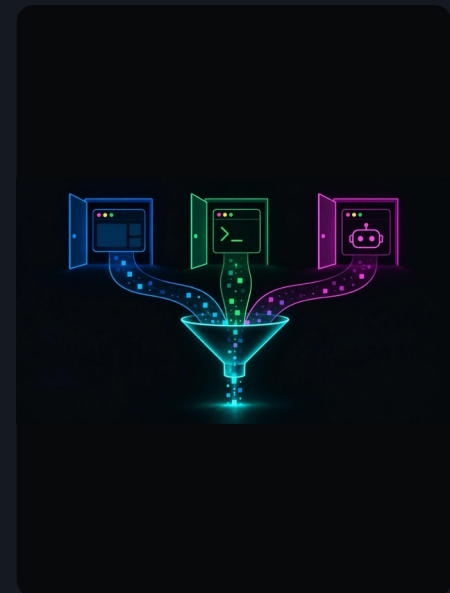
Every surface calls the same function

```
// CLI - cli/input.ts
await sendSessionInput({ kind: "remote", button }, sessionControlOpts(opts));

// MCP - mcp/tools/input.ts
await sendSessionInput({ kind: "remote", button }, link.options());

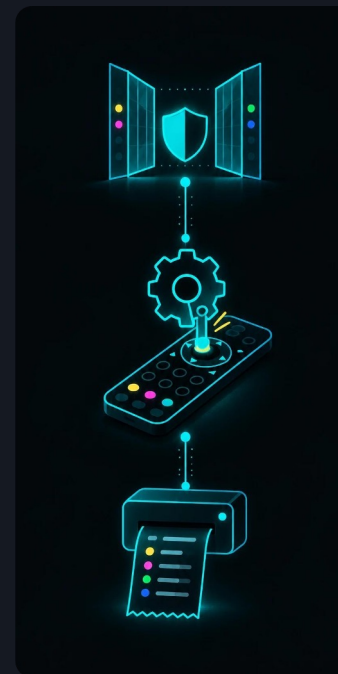
// Browser - server/ws.ts (via the same action)
void sendInput({ ...ctx, actor: { kind: "human", id: "browser" } }, input);
```

```
// server/routes/input.ts
const event = parseInputEvent(await readJson(ctx.req));
await sendInput(actionContext(ctx), event);
```



Wrap every action the same way

```
export async function runAction(ctx, name, params, fn) {  
  const started = Date.now();  
  try {  
    assertInputAllowedByControl(ctx, name, params); // 1  
    const result = await fn(); // 2  
    ctx.session.activityBus?.publish({ // 3  
      deviceId: ctx.deviceId, actor: ctx.actor,  
      action: name, params, outcome: "ok",  
      durationMs: Date.now() - started,  
    });  
    return result;  
  } catch (error) {  
    const converted = asPortholeError(error);  
    ctx.session.activityBus?.publish({ /* ... */ });  
    throw converted;  
  }  
}
```



1 • Policy

Is this actor allowed right now?

2 • Operation

Press, screenshot, launch.

3 • Evidence

Actor, action, outcome and duration, for failures as well.

WHERE TO DRAW THE LINE

Action tiers

Tier	Example	Who composes it
Primitives	remote, get_focused, screenshot, dump_ui	The agent
operations	focus_on, wait_for, assert_screen	The tool
Workflows	“Open my app, go to Settings, check nothing crashed”	Agent + skill (skills/porthole/SKILL.md)

workflow > operations > primitives

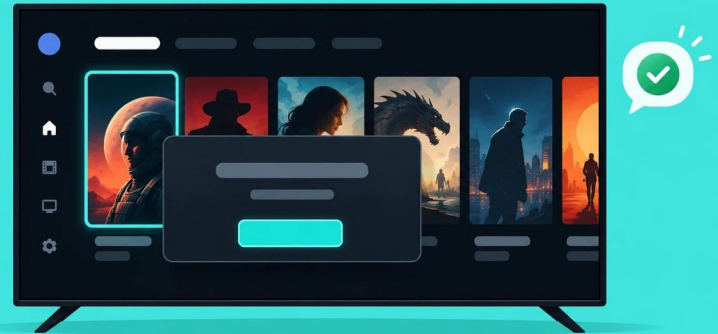
EVIDENCE

Check for evidence

WHAT THE AGENT SEES

```
→ remote({ button: "dpad_right" })  
← "Remote: dpad_right"
```

Reported as success



A follow-up read shows the screen changed correctly

EVIDENCE

Reads > writes

```
attach_device
get_control_state  ← read
say "Focusing Interstellar"
focus_on "Interstellar"
  ↳ dump_ui · remote · dump_ui · remote · dump_ui
get_focused        ← read
screenshot         ← read
```

2×

six reads for three writes in this trace

Most agent calls are reads, so design the read tools first.

EVIDENCE

Observable evidence



Structured output

`get_focused` returns `structuredContent` with an `outputSchema`; the CLI's `-q` prints one JSON object.



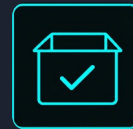
Typed, retryable errors

```
{ "code": "PAUSED_BY_HUMAN",  
  "retryable": true }
```



Mark reads as reads

```
annotations: { readOnlyHint: true }
```



Expected nulls aren't errors

`get_focused` on a phone returns null, by design.

SHARING THE DEVICE

Rules for sharing one device

Primitive	Human (browser)	Agent (MCP / CLI)
Actor	human/browser	agent/mcp, cli, ci
Control	“Take control” button	gets PAUSED_BY_HUMAN (423, retryable)
Narrate	sees intent in activity feed	say “Focusing Interstellar”
Ask	answers in the browser	ask_human(“Which profile?”)



```
// actions/run.ts
if (!isMutatingAction(name, params) || ctx.actor.kind === "human") return;
if (ctx.session.getControlState?.(ctx.deviceId)?.holder === "human")
  throw new PortholeError("PAUSED_BY_HUMAN", "...Wait, or call ask_human.", 423, true);
```

LIVE DEMO

Hand off the remote

01 · Watch

Agent checks control, narrates, verifies

02 · Take over

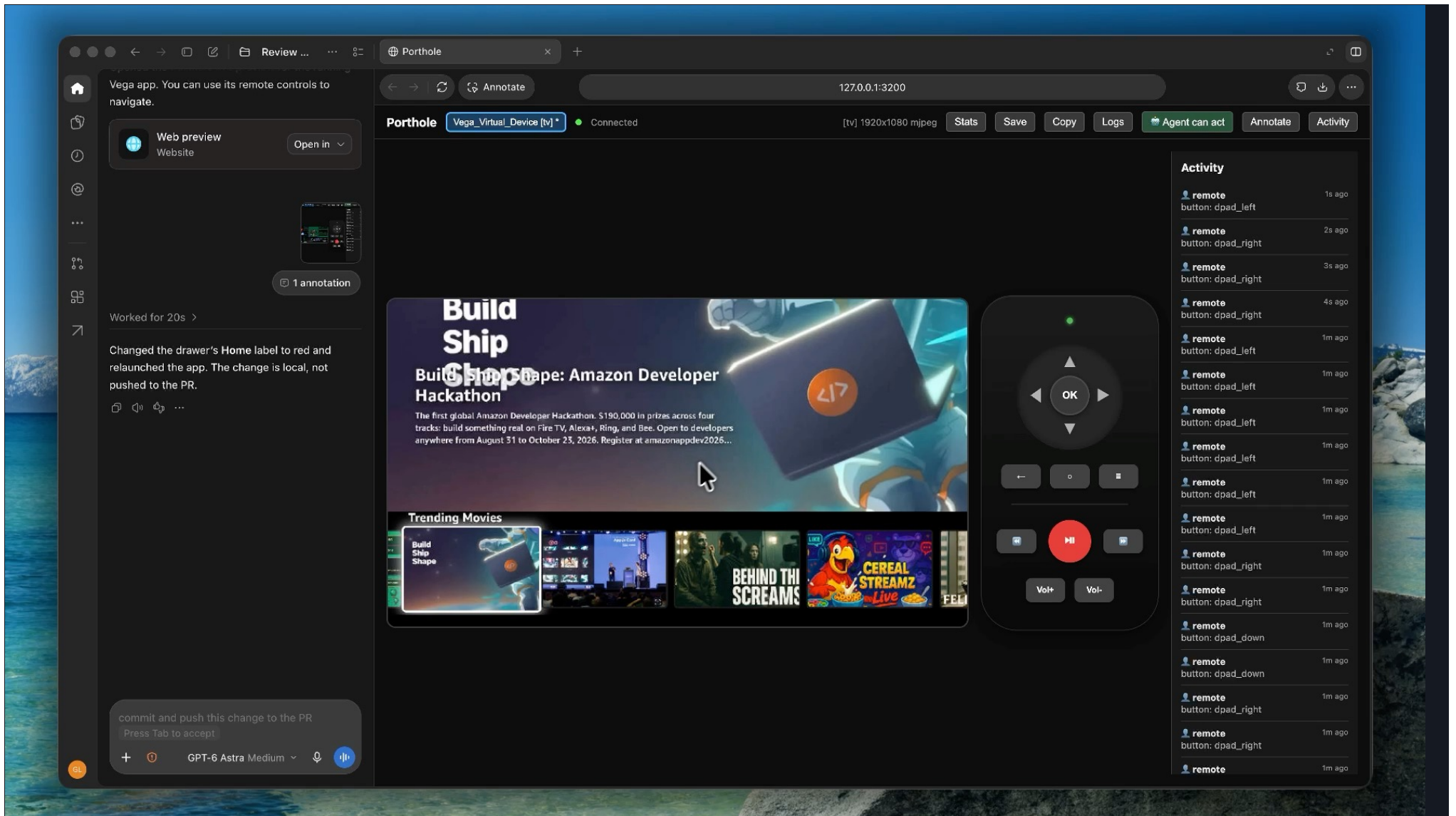
I grab the remote; feed says "human"

03 · Paused

Agent gets PAUSED_BY_HUMAN, asks me

04 · Hand back

Agent acts, then checks again



TAKEAWAYS

Patterns you can reuse



One owner of state. Every surface is a client.



Logic below the surfaces. Surfaces only translate.



Record evidence where actions run. Who, what, outcome



Add workflows for complex combined interactions. Otherwise, ship primitives



Design reads first. Structured, typed, retryable.



Humans can pause agents. Block writes, keep reads

TAKEAWAYS

Audit a tool you maintain

Question	Porthole's answer	Yours
What's the smallest thing it does?	<code>remote(button)</code>	
How do you check it worked?	<code>get_focused</code> , <code>screenshot</code>	
Who can see what happened?	Activity bus → browser feed + API	
What if a human steps in?	<code>PAUSED_BY_HUMAN</code> , <code>ask_human</code>	

Start with one workflow/action, the verification, the failure and a human hand-off

Thank you

`npx portholejs · github.com/giolaq/porthole`

Giovanni Laquidara · github.com/giolaq

