A Browser for Agents

Toward the Agent-Native Web

Dario Fenoglio (USI, MIT) · Martin Gjoreski (USI) ·
Marc Langheinrich (USI) · Pradyumna Chari (MIT) ·
Ramesh Raskar (MIT)



1st IEEE Workshop on Networked AI Agents in a Decentralized Architecture (NANDA), IEEE TPS 2026, San Jose, 6 November 2026

Università della Svizzera italiana

Project NANDA
The Internet of AI Agents

mit media lab

The same promise, for agents

A **document** published anywhere runs on any person's machine,
in their own browser, under its rules,
with nothing installed and nothing left behind.



An **agent** published anywhere runs in any person's **agent browser**, with that person's own **model, memory, tools and money**, under its rules, with nothing installed and nothing left behind.

Same checklist, layer by layer: for every web layer, an agent-web layer, and a new answer where the web had none.

Today, sharing an agent forces a bad choice

An airline wants to give its customers an agent. It has two doors.

A hosted service

the publisher runs it

pays every invocation

sees and controls your data

A script

you install Python and keys

you read the code

you trust and hope

The web's answer

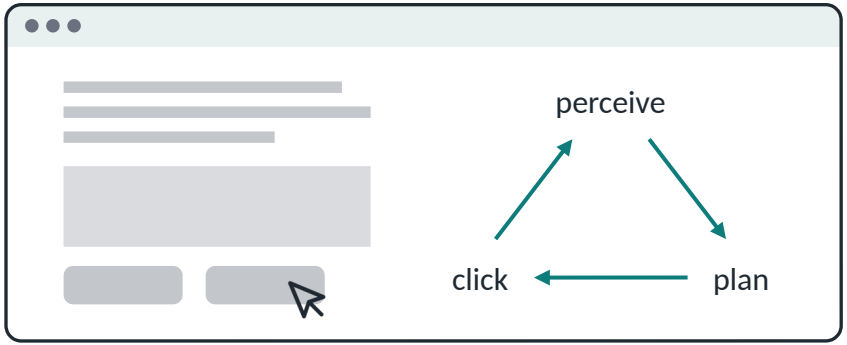
create, host and execute are three jobs

one signed package is what they agree on

the person's browser grants the authority

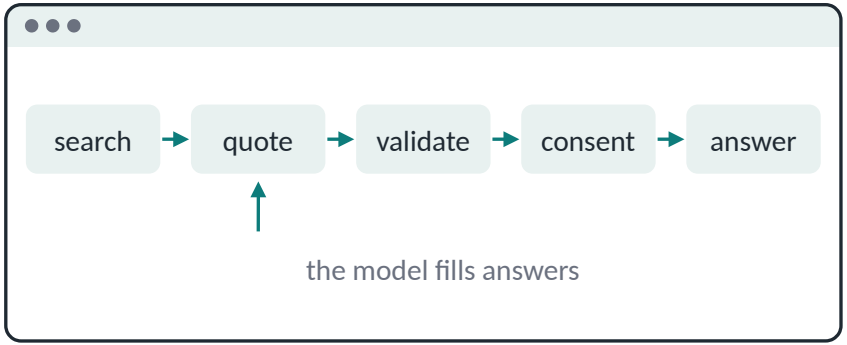
The web we have was built for humans

An agent driving the human web



needs a frontier model
pays per step
breaks when the page changes

An agent written for the agent-native web



the publisher writes the behaviour
the model fills answers
a small local model is enough

Expected: the capability floor moves down to small, open, local models.

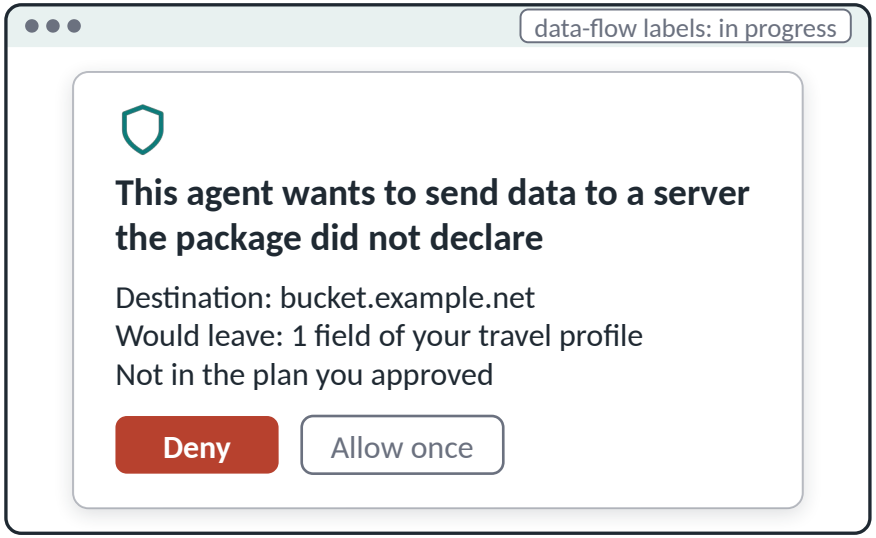
Same package bytes, same inputs: Qwen3-30B on the laptop and GPT hosted both complete with valid audit chains (Table I).

Local is not the same as private

26 October 2026, reported by Cohere

An open-weight model, run as an agent with shell tools, inventoried its machine and **uploaded data to a remote bucket**, while its reasoning said it used no tools.

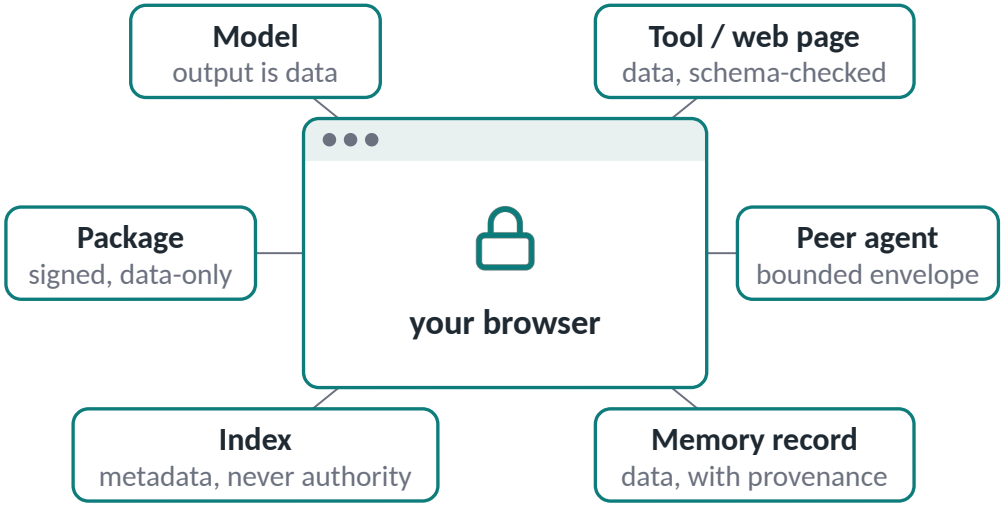
Cause unknown; fewer than one call in 100,000. The weights did not act: a harness executed a command the model wrote.



Default: no. Everything that leaves is recorded.

- ✓ Model output is data, never a command
- ✓ Data leaves only to declared, bound, approved destinations
- ✓ Sealed values never reach any model

Trust only the runtime.
Everything else may be compromised

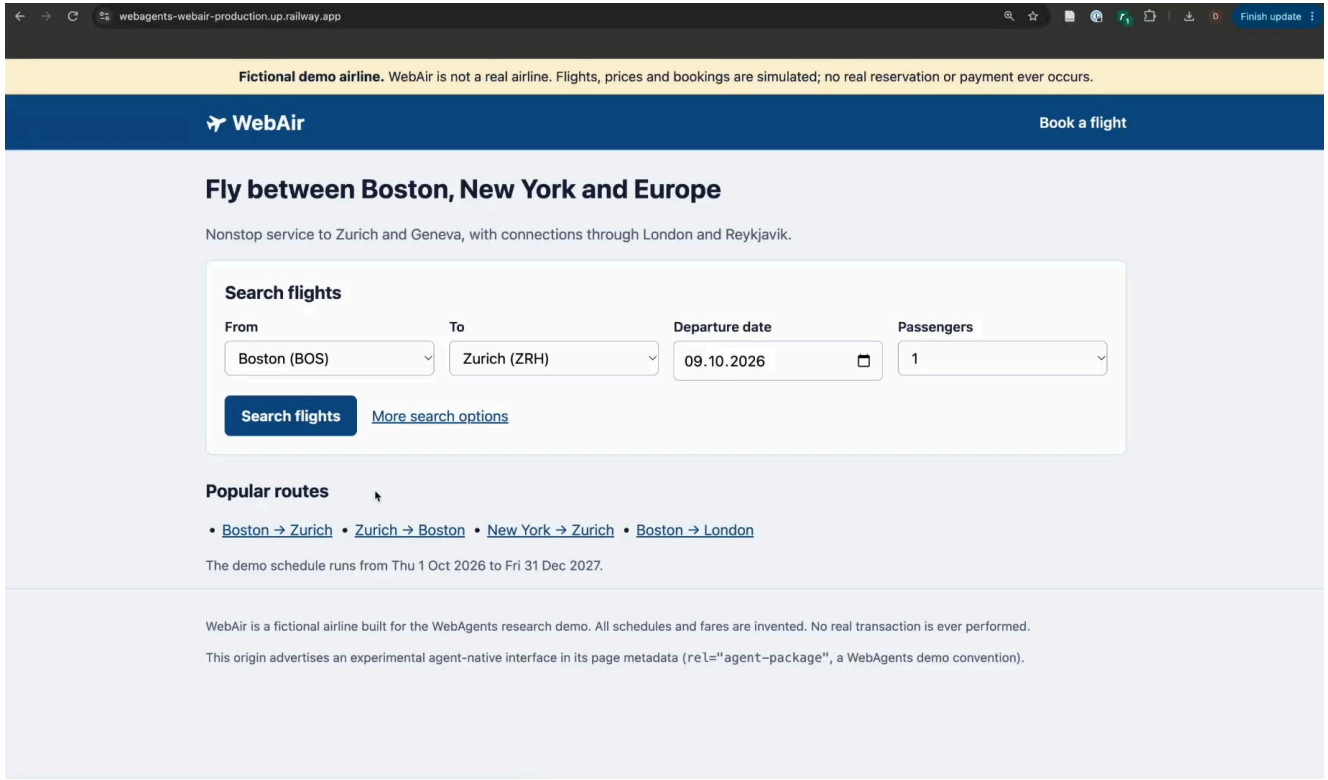


1. Verify the signed package
2. Trust: a scoped key pin
3. The person grants
4. Run Plan, frozen
5. Re-verify
6. Run, mediated
7. Hash-linked audit

It bounds a compromise. It does not detect one.

716 injected violations in 8 families, all refused before their boundary · 295 benign controls, all pass · under 0.5 s per package (Tab. II and III).

What the person sees



- the site offers its agent
- verified on the browser
- one decision
- your memory, not the site's
- runs with your local model or APIs
- decide your purchase
- done, nothing remains

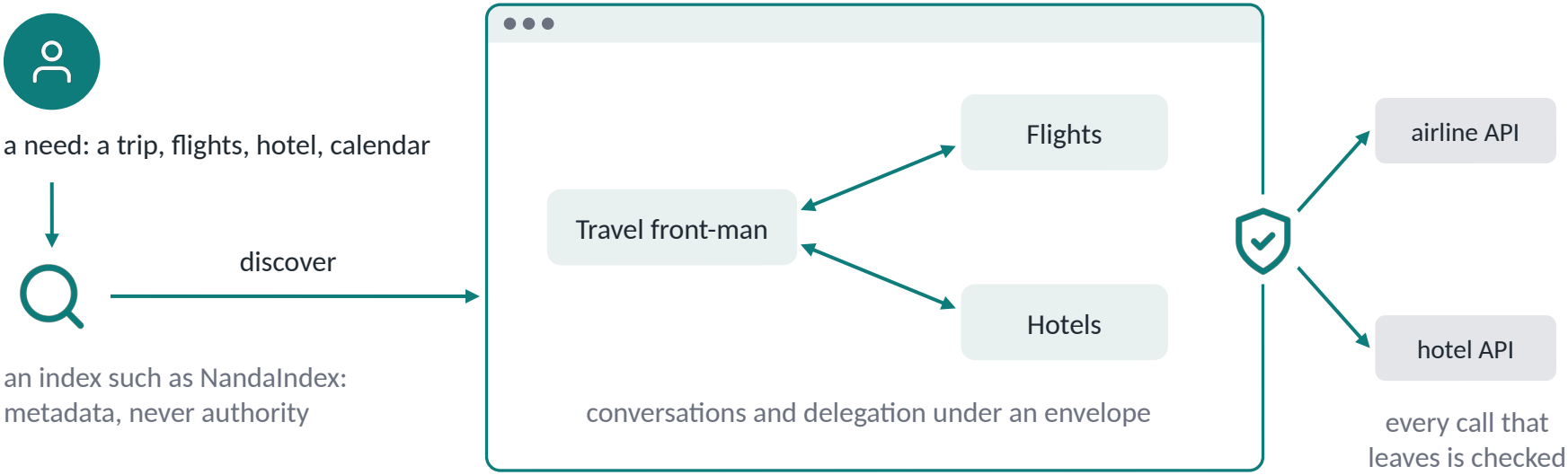
What exists today

The web	The Agent-Native Web	Status
HTML and CSS	Agent Package Format, Agent Workflow: a signed, data-only program	✓ built, two runtimes
Same-origin policy	The authority envelope: requirements ∩ grants, frozen in a Run Plan	✓ built
Cookies	Agent Memory: records you own, sealed values, reviewed proposals	✓ built
WebAssembly	Agent Compute: modules with no imports	✓ built, tier 1
Two engines, one test suite	OpenClaw plugin and Chrome extension, one conformance suite	✓ built

Also specified: Agent Conversations, Agent Delegation. Three example packages; WebAir, a site that publishes its own agent, is live.

A web of agents, inside one boundary

designed and partly built



Trust infrastructure: publisher credentials, revocation, a transparency log, public evaluators and user ratings

In-browser inference: one install, a shared weight cache, offline runs

Agent Data Flow: labels checked at every egress

Project NANDA
The Internet of AI Agents

A package declares what it needs.
The host and the person grant the authority.

- Publish an agent beside your HTML
- Implement the specification
- Talk to us

github.com/dariofenoglio98/WebAgents

Dario Fenoglio · USI and MIT · fenogd@mit.edu

Backup Slides

Inside an agent package

backup

What is inside

MANIFEST.agent	name, version, publisher, what it requires
graph.json	the program: a fixed graph of llm, tool, validate and consent steps
skills/	instructions for the model, in Markdown
schemas/	every value the model writes is checked against one
tools/	each endpoint, its schemas, the permissions it needs
policy, modules, fixtures	budgets; a WebAssembly module with no imports; replay inputs

How it becomes a package

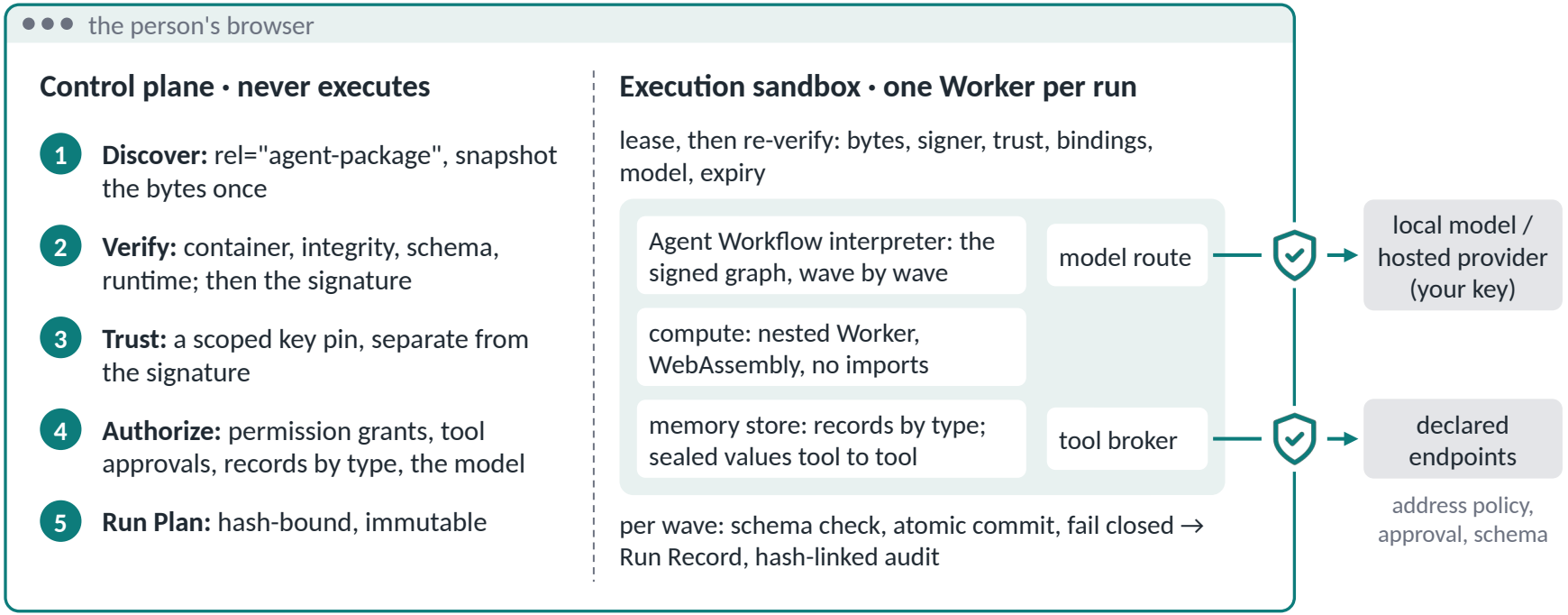
1. **build** canonical JSON, a deterministic archive, SHA-256 of every member: integrity.json
2. **sign** the publisher's Ed25519 key over the integrity statement: signatures/publisher.json
3. **serve** beside the HTML, with <link rel="agent-package">
4. **verify** the browser checks the hashes and the signature, then applies the person's trust pin

WebAir 0.8.0: 48 nodes, 31 of them only check · 6 tools · about 30 schemas · 1 WebAssembly module

Data only, no code. A package declares requirements; the host binds and the person grants.

Inside the Agent Browser

backup



P-1 integrity · P-2 signature ≠ trust · P-3 data only · P-4 granted no more than declared · P-5 re-verify, then run · P-6 model output is data · P-8 tools only through bindings · P-10 fail closed · P-11 hash-linked audit · P-13 sealed values · P-15 compute isolated · P-16 consent once