WebAgents: Unbundling Agent Creation, Hosting, and Execution for an Agent-Native Web

Dario Fenoglio*†, Martin Gjoreski*, Marc Langheinrich*, Pradyumna Chari†, and Ramesh Raskar†
*Università della Svizzera italiana (USI), Switzerland
†Massachusetts Institute of Technology (MIT), United States
Corresponding author: dario.fenoglio@usi.ch

***Abstract*—Large language models are moving from isolated assistants toward autonomous agents that can plan, use tools, and coordinate with others, motivating an emerging Agentic Web of independently developed agents and services. Realizing this vision requires agent behavior to move across creators, hosts, and runtimes without coupling publication to a particular execution environment or transferring authority over user resources. We introduce WebAgents, which unbundles these roles through a signed `agent.pkg` that carries portable workflows, skills, schemas, and resource requirements. An OpenClaw-based agent-native browser/runtime verifies the package, obtains authorization, and binds it at run time to user-controlled models, tools, permissions, and budgets, while hosts distribute the public program and may retain private services server-side. We demonstrate this contract with three unrelated workflows executed unchanged under local and API-backed models, including literature review with live services and a website-published travel agent using semantic tools and user-selected personal resources. In a controlled suite, WebAgents explicitly blocks all 716 injected contract violations while accepting all 295 matched benign controls, with median automated pre-execution processing below 0.5 s for all three packages. We conclude by outlining a path toward NANDA-based discovery and interoperable package ecosystems while retaining execution authority in the user's runtime.**

***Index Terms*—AI agents, Agentic Web, agent interoperability, portable execution, runtime security.**

I. Introduction

Large Language Models (LLMs) are increasingly moving beyond passive content generation toward autonomous agents that can plan, invoke tools, maintain state, and coordinate with others. These capabilities are already being explored across domains ranging from scientific discovery to enterprise automation, suggesting a shift from isolated AI applications toward ecosystems of interacting agents [1]–[3]. As these systems mature, the Web can become a substrate not only for humans accessing pages and services, but also for agents discovering capabilities, delegating tasks, and acting across organizational boundaries. We refer to this emerging setting as the *Agentic Web*: an open environment where independently developed agents, tools, and services can be discovered, composed, and executed [4]. Realizing this vision, however, requires more than increasingly capable models. It requires infrastructure that allows agent behavior to move across creators, hosts, and runtimes without coupling publication to a particular execution environment or transferring authority over user resources.

Consider a researcher sharing a literature-review workflow. Hosting it as a service makes the publisher responsible for orchestration and inference, while distributing a script leaves each recipient to configure models, credentials, and tools. Neither approach, by itself, provides a common boundary between the published workflow and the recipient's authority.

We argue that coupling agent creation, hosting, and runtime limits an open agent ecosystem. We therefore separate three roles: (i) a creator produces a portable agent program; (ii) a host makes that program reachable and may retain private services server-side; and (iii) a user-controlled runtime verifies and executes it using authorized resources. This separation lets creators distribute expertise without operating every invocation, while users retain control over the models, credentials, and budgets used on their behalf. With a common execution contract, creation tools, hosting providers, and runtimes can evolve independently.

However, separating these roles is not sufficient. A downloaded agent may request access to private data, invoke remote tools, or consume a substantial inference budget. The package must therefore describe behavior without granting itself the authority to perform it; the receiving runtime decides which requests to accept and enforces those decisions throughout execution. In our framing, an *agent-native browser* executes a publisher-defined program rather than reconstructing behavior from human-facing pages. The goal is a web designed for agents, not only agents designed to navigate the existing web [5], [6]. Website automation can remain one of its tools, but is not its defining function.

To explore this separation, we introduce WebAgents, a portable execution framework built around a signed agent package and an OpenClaw-based browser/runtime. WebAgents realizes the three roles around a shared `agent.pkg` artifact: creators produce and sign portable packages, a host distributes them independently of execution, and the OpenClaw agent-native browser/runtime binds them at run time to the user's authorized models, tools, permissions, and budgets. This connects signed behavior, user authorization, and concrete model/tool bindings in an auditable execution contract. We demonstrate the design with three workflows executed unchanged under local and API-backed model bindings, and use controlled experiments to test whether the contract rejects violations and

This work was supported by EU Horizon Europe through SmartCHANGE (Grant 101080965) and by the Swiss National Science Foundation through XAI-PAC (Grant PZ00P2_216405).

measure its automated pre-execution cost. We then identify requirements for portable execution and outline a path toward discovery and richer execution profiles.

With this work, we hope to contribute toward a more open and interoperable Agentic Web in which reusable agent behavior can move across providers and runtimes while execution authority remains with the user.

II. Related work

a) Agent interoperability and discovery: The Web of Agents identifies messaging, interaction, state management, and discovery as foundations for cross-ecosystem collaboration [1]; Agent-OSI develops communication and settlement interfaces for a decentralized agent ecosystem [7]. MCP standardizes access to tools and context, including consent requirements [8], while A2A supports task exchange between independently implemented agents [9]. Work on agent infrastructure motivates identity and discovery beyond fixed service endpoints [10]; NANDA separates a lean index from verifiable AgentFacts and endpoint metadata [11]. These mechanisms let agents find and invoke capabilities. Our focus is the additional step of transferring an agent program for execution under the recipient's authority.

b) Portable agent behavior: Agent Skills packages reusable instructions, scripts, and resources that agents can load across compatible environments [12], while Paper2Agent exposes research methods and workflows as reusable MCP tools [13]. Open Agent Specification defines a framework-independent representation of agents, workflows, data flow, and tool interfaces, with adapters that execute the same specification across multiple agent frameworks [14]. WebAgents moves portability to a different boundary: portable execution across independently operated creators, hosts, and runtimes. A signed program is distributed independently of execution, while the receiving runtime selects and authorizes the concrete models, tools, permissions, and budgets used on its behalf. Thus, portability does not require adopting the publisher's framework, infrastructure, or execution authority.

c) Package identity and provenance: JAR combines manifests, content digests, and signatures [15], while in-toto verifies signed evidence of software production steps [16]. The Agent Packaging Standard also specifies agent manifests, distribution, and optional provenance verification [17]. These approaches establish artifact identity and provenance. WebAgents extends this boundary into execution: the verified package is bound to the user's approved execution configuration, and the same binding must still hold when execution begins.

d) Runtime control and enforcement: D'Agents combined authenticated programs with resource policies and restricted execution [18]. For LLM agents, AIOS separates applications from managed models, tools, and access control [19]. AgentSpec enforces event–condition–action rules at runtime [20], and CaMeL uses explicit control and data flow with capability policies against prompt injection [21]. These systems show the importance of mediating execution when external capabilities are involved. WebAgents brings publication, authorization, and runtime enforcement into one verifiable lifecycle, connecting this control to a portable, independently published agent program. We evaluate this lifecycle through unchanged-package executions, contract-violation tests, and pre-execution timing.

III. Unbundling through a portable execution contract

An open Agentic Web needs publication and execution to scale independently. Reusing an agent should not require the creator to operate its inference or the user to adopt a publisher-specific resource environment. We therefore define unbundling as *independent substitutability*: creators, hosting providers, and compatible runtimes can change without rewriting the public agent program. A provider can offer several roles; the shared contract keeps the others replaceable.

a) Creation, hosting, and runtime: The creator specifies the workflow, skills, and requirements in a signed `agent.pkg`. The host makes the package reachable and may separately expose private services. The browser/runtime inspects the package and supplies the user's chosen model, approved tools, and execution limits (Figure 1). This separates the cost of making an agent available from the cost of running it: inference is provisioned by the runtime, while private data and business logic can remain behind service interfaces. Users can reuse a configured environment across independently authored agents, and creators can improve workflows without operating each user's execution stack. As the ecosystem grows, shared interfaces would let creation tools, hosting services, and runtimes improve independently rather than requiring one platform to provide all three.

b) Requirements are not permissions: A literature-review package may require structured model outputs, sufficient context capacity, and scholarly search. These requirements describe what execution needs; the runtime determines which concrete resources satisfy them and obtains permission to use them. The resulting contract binds the signed program to those approved resources and limits for a particular run. This is the connection between portability and control: the program can be shared across environments, while credentials and authority remain with each receiving runtime. Compatibility is explicit, and different model bindings can produce different answers.

IV. The WebAgents architecture

The prototype implements the separation in Figure 1 through a data-only package, a hosting layer, and a user-controlled OpenClaw runtime containing the browser control plane and mediated execution engine. The host distributes the package and may expose private services, while the runtime verifies, authorizes, binds, and executes it. Hosting a package does not grant access to runtime credentials or tools.

A. A signed program, not a capability grant

An `agent.pkg` is a deterministic archive containing a manifest, workflow graph, markdown skills, input/output schemas, tool declarations, and policies. The manifest identifies the package version and execution profile, and declares the model

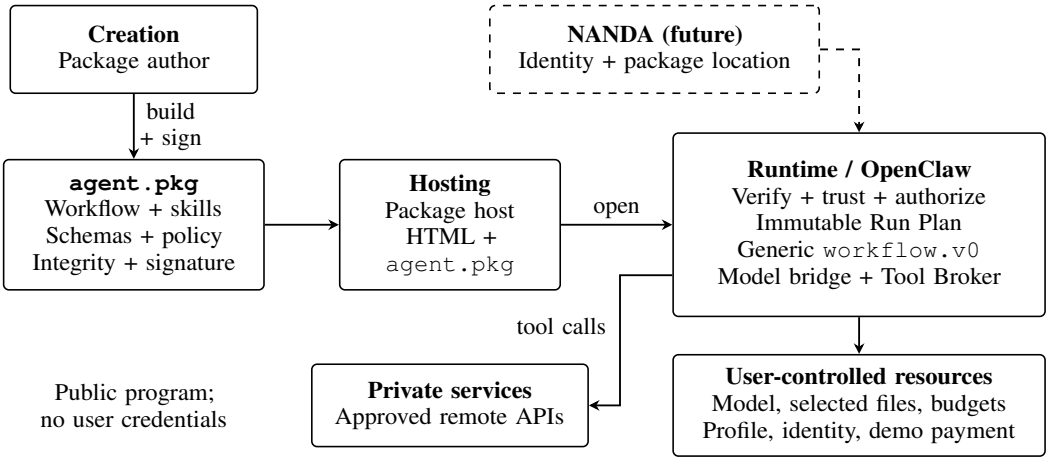


Fig. 1. Separation of publication and execution in WebAgents. The package transports behavior and requirements; it does not transport user authority. Solid paths include experimental website-advertised discovery. The dashed discovery path is a proposed NANDA integration: its output would enter the same verification and authorization process as a direct package URL.

capabilities and resources needed to run the workflow. The current format contains data interpreted by `workflow.v0`; it does not load package-supplied Python, shell scripts, or native modules.

Package verification uses SHA-256 content hashes and an Ed25519 publisher signature over the canonical integrity statement. The browser also records a hash of the exact archive bytes. Parsing is bounded and rejects ambiguous or unsafe structures, including duplicate archive names, path traversal, duplicate authoritative JSON keys, and remote schema loading. Inspection does not execute the workflow. This matters because authorization must be based on a stable description of what will run, not on behavior already triggered while opening it.

A valid signature binds content to a signing key; it does not establish that the signer is trusted or that the workflow is safe. In the prototype, the runtime independently pins accepted signer keys under an explicit scope. The package is inspectable by design, so secrets do not belong inside it. Provider credentials stay with the runtime, while proprietary host-side logic remains accessible only through approved service interfaces.

B. From inspection to an immutable Run Plan

The browser opens an explicit HTTPS package URL or an approved local artifact. When a user instead provides a website URL, the browser can also discover an `agent.pkg` published by that site. HTTPS acquisition is size-bounded and rejects redirects and private-network targets. It checks the package structure, integrity, signature, and supported runtime before evaluating publisher trust. The user then grants or denies permissions and chooses a compatible, routable model. Tool selection has a further boundary: a declared tool, an available runtime implementation, and an approved binding are distinct objects. A general permission to perform scholarly search does not authorize arbitrary network requests.

These decisions produce a *Run Plan*: a hash-bound description of the exact package snapshot, signer and trust state, authorization, selected model, approved tool bindings, and personal-resource grants. A separate *Run Invocation* binds task input to that plan. Immediately before execution, the runner re-verifies the snapshot and current authority. If the package, trust decisions, or bindings no longer match the plan, execution is refused rather than silently updated. This closes the gap between the artifact the user inspected and the artifact actually executed.

The plan is preparation, not an independent permission token. Execution still passes through the trusted runner and its checks. A distinct *Run Record* captures completion status, usage, artifacts, and audit information without mutating the plan. Separating these objects makes it possible to compare the authorized configuration with what a particular run consumed and produced. Appendix A summarizes the complete package-to-result lifecycle.

C. Generic execution with runtime-controlled resources

The `workflow.v0` interpreter executes an acyclic graph using six node types: `llm`, `tool`, `map`, `join`, `validate`, and `output`. Model and tool nodes use schema-checked interfaces; map nodes create bounded workers; join nodes combine results; validation nodes check declared structural and cross-record constraints. Skills and orchestration are package data, with no dispatch on package names. Workers receive copied state views, and validated writes are committed after successful execution waves. Failed waves do not partially commit shared workflow state.

Model calls pass through a model bridge to exactly the model recorded in the Run Plan. The package cannot access provider credentials or silently switch to a fallback. Abstract requirements are checked against runtime capability information, while returned structured data is independently validated.

Tool calls pass through the runtime Tool Broker. Each binding fixes an adapter and its contract; the package cannot choose arbitrary request URLs, headers, filesystem paths, or shell commands for execution. For the Literature Review package, the runtime provides adapters for Crossref search, a remotely

hosted Citation Resolver, and user-selected PDF snapshots. The resolver returns scholarly metadata and BibTeX. For PDF access, the runtime exposes opaque file identifiers rather than local paths. WebAir adds four operator-configured HTTP bindings for its airline services. These are example runtime bindings, not built-in workflow operators: other packages can use different adapters without changing the interpreter.

Packages can also request user-controlled resources by type, such as a travel profile or passenger identity. The runtime exposes eligible resources, the user selects and authorizes one, and that choice is bound to the Run Plan. The package therefore expresses what it needs without selecting or carrying the user's private resource itself.

The runtime enforces token and call budgets, worker concurrency, and deadlines outside the prompts. It reserves resources before dispatch and accounts for actual usage afterward. Cancellation propagates to model and tool providers where supported. These controls bound execution, but neither cancellation nor failed state commits imply rollback of an external service's completed side effects.

Execution emits a locally hash-linked audit trail with provenance and usage artifacts, recording run configuration and actions rather than certifying its answers. Enforcement assumes a trusted OpenClaw runtime and approved adapters. A malicious instruction can still misuse an approved capability, as agent-risk and prompt-injection evaluations illustrate [22], [23].

V. Evaluation

Our evaluation separates three questions. The live case studies test whether unchanged packages execute under different model bindings. RQ1 tests whether the contract blocks violations without rejecting matched benign executions. RQ2 measures the local cost of its automated pre-execution checks. Additional implementation lessons appear in Appendix C. The WebAgents implementation, evaluated packages, and experimental artifacts are publicly available.[1]

A. Three packages, one interpreter

Table I summarizes *Meeting Agenda*, *Literature Review*, and *WebAir* executed through `workflow.v0` under both a local Ollama model and an OpenAI API model. For each workflow, the package archive and task input are identical across the two runs, so changing the model binding does not require repackaging the agent. *Meeting Agenda* plans a meeting, creates three parallel role-based perspectives, and synthesizes a structured agenda. Both runs produced agendas whose items sum to the requested 25 minutes. It uses no external tools, providing a case distinct from literature review without an application-specific runner.

Literature Review adds live services to the same runtime. Its package defines search, screening, citation resolution, evidence extraction, synthesis, writing, and verification. Both runs address the same task on prompt-injection defenses with the same approved tool bindings. Each run made three Crossref searches and three Citation Resolver calls and produced a related-work draft with bibliographic, evidence, unresolved-item, and verification artifacts. All 18 model calls in each run used the model selected in its Run Plan.

WebAir adds a website-native case: a fictional airline publishes a signed `agent.pkg` alongside its human-facing webpages. After discovering and verifying the package, the agent can interact with WebAir through its agent-native tools for flight search, fare rules, booking quotes, and simulated purchase, rather than reconstructing these actions from the human-facing site. The matched search runs each read the selected travel profile and make four live service calls: one flight search and three fare-rule requests.

All six runs completed with valid audit chains. Together, they show that the same packaged workflows can execute under local and API-backed model bindings without changing the package itself. This demonstrates portability across model-provider bindings within the same OpenClaw runtime.

B. RQ1: Contract enforcement

Successful workflows do not establish that their execution boundaries are enforced. We therefore inject violations of the contract's declared guarantees across eight families (Table II). Each template specifies the expected refusal stage and an objective forbidden boundary before the run. Examples include a tampered package reaching a prepared plan, an unplanned model's answer being committed, or a call being charged below its honest usage.

a) Setup and metrics: Across Meeting Agenda and Literature Review, spanning tool-free and tool-enabled workflows, the suite contains 716 distinct violations from 61 templates and 295 matched benign controls that follow the same execution paths without the violated condition, preventing trivial reject-all behavior. Each case exercises the Agent Browser, control plane, and runtime, with deterministic stand-ins for external dependencies. We count a violation as detected only when the forbidden boundary is not reached and the contract explicitly refuses execution, while the controls measure whether benign executions remain accepted. To identify which mechanisms provide this protection, we repeat the suite under eight ablations, each disabling one enforcement component. Full threat-model, outcome, simulator, and ablation details are given in Appendix B.

b) Results: The full system explicitly blocked all 716 violations before their forbidden boundary, at the lifecycle stage specified in advance, and accepted all 295 controls. All 9,099 executions across the nine profiles were recorded without harness errors. Removing each component exposed violations in its corresponding family. For example, detection fell to 48.7% without package verification, 30.9% without plan revalidation, 30.0% without runtime mediation, and 17.9% without output validation. Every lost block became a silent violation, not an incidental failure.

The checks also overlap. Removing plan revalidation reduced authorization detection to 52.9% because 16 authorization cases concern consent binding or expiry before the run

[1] https://github.com/dariofenoglio98/WebAgents

TABLE I

ACCEPTANCE RUNS THROUGH THE SAME GENERIC INTERPRETER. WITHIN EACH WORKFLOW, LOCAL AND API RUNS USE IDENTICAL PACKAGE BYTES, TASK INPUTS, PERMISSIONS, RUNTIME LIMITS, AND TOOL BINDINGS. TIMES ARE SINGLE-RUN WORKFLOW DURATIONS.

	Meeting Agenda		**Literature Review**		**WebAir**	
	Local	**API**	**Local**	**API**	**Local**	**API**
Model	Qwen3-30B	GPT-5.6 Luna	Qwen3-30B	GPT-5.6 Luna	Qwen3-30B	GPT-5.6 Luna
Model calls	5	5	18	18	3	3
External tool calls	0	0	6	6	4	4
Personal-resource reads	0	0	0	0	1	1
Peak parallel workers	3	3	3	3	3	3
Reported tokens	11,346	5,198	100,352	68,398	11,577	7,716
Workflow duration (s)	154.3	29.4	1,489.8	417.6	81.7	18.6
Audit events	92	102	572	678	241	247
Model cost (USD)	N/A	0.0147	N/A	0.2005	N/A	0.0094

TABLE II

CONTRACT ENFORCEMENT: PERCENTAGE OF INJECTED VIOLATIONS EXPLICITLY BLOCKED BEFORE THEIR FORBIDDEN BOUNDARY. ABLATIONS ARE DEFINED IN THE TEXT; BOLD RATES MARK FAMILY-LEVEL DROPS. ALL 295 BENIGN CONTROLS ARE ACCEPTED IN EVERY PROFILE; INCIDENTAL FAILURES AND HARNESS ERRORS ARE ZERO. THE OVERALL ROW IS WEIGHTED BY THIS CATALOG'S CASE COUNTS.

				Removed enforcement component							
Family	Viol.	Controls	Full	Verif.	Trust	Author.	Compat.	Plan reval.	Runtime med.	Output val.	Resource
Package verification	191	134	100	**48.7**	100	100	100	100	100	100	100
Publisher trust	14	12	100	100	**14.3**	100	100	100	100	100	100
Authorization	34	12	100	100	100	**47.1**	100	**52.9**	100	100	100
Requirements / compatibility	85	17	100	100	100	100	**76.5**	100	100	100	100
Plan binding	55	21	100	100	100	100	100	**30.9**	100	100	100
Runtime mediation	80	15	100	100	100	100	100	100	**30.0**	100	100
Output / state integrity	196	59	100	100	100	100	100	100	100	**17.9**	100
Resource governance	61	25	100	100	100	100	100	100	100	100	**18.0**
Overall	**716**	**295**	**100**	86.3	98.3	97.5	97.2	92.5	92.2	77.5	93.0

deadline. Other layers continued to block subsets of the ablated families, including retained structural checks and checks at later boundaries. The matrix therefore identifies which cases depend on each component, rather than showing that one check can enforce the whole contract.

C. RQ2: Pre-execution overhead

We measure the automated contract processing from signed package bytes already in memory to the interpreter's first-node dispatch boundary, before any node executes. To approximate the cost of local execution in the user-controlled runtime, we evaluate it on a local workstation: an Apple M3 Max (14 cores, 36 GB), with Python 3.13.5. Each of three runs includes 50 warmup iterations per package followed by 1,000 timed observations per package, giving 3,000 observations per package. Package order is randomized within each iteration. Appendix B-C gives the protocol and artifact sizes.

Median processing took 59.0 ms for Meeting Agenda, 449.9 ms for Literature Review, and 488.7 ms for WebAir (Table III), corresponding to less than 0.2% of end-to-end workflow duration for Meeting Agenda and Literature Review, and 0.6–2.6% for WebAir. The coefficient of variation of run medians was at most 0.41%. Repeated verification and payload validation at the control plane, runner, and interpreter account for approximately 90%, 96%, and 96% of the measured time. The higher processing times for Literature Review and WebAir relative to Meeting Agenda are consistent with their larger archives, workflow graphs, and sets of declared tools. A separate component microbenchmark identifies schema and cross-reference validation as the largest contributor to package-open time; Ed25519 verification takes at most 0.39 ms.

TABLE III

AUTOMATED PRE-EXECUTION PROCESSING (MS): POOLED MEDIAN (95TH PERCENTILE), 3,000 OBSERVATIONS PER PACKAGE.

Operation	Meeting Agenda	Literature Review	WebAir
Package verification	19.3 (20.4)	152.7 (160.1)	161.7 (168.4)
Authorization / binding	1.7 (1.8)	2.7 (2.9)	4.0 (4.2)
Run Plan preparation	2.1 (2.3)	3.9 (4.2)	6.5 (6.9)
Pre-execution revalidation	35.8 (38.1)	289.4 (301.1)	316.1 (327.3)
Total	**59.0 (62.5)**	**449.9 (465.0)**	**488.7 (504.7)**

These measurements isolate the contract's local processing, not end-to-end startup latency. They exclude package download, human authorization time, model and tool calls, loopback HTTP transport, and runner process startup. Together, the

experiments show rejection of the injected violations and sub-second automated preparation on this machine.

VI. Toward an open Agentic Web

The current prototype opens packages from explicit HTTPS URLs, experimental website-advertised links, or approved local artifacts. NANDA integration would let users discover candidate packages by identity and capability [11], then enter the same verification and authorization pipeline. This extends how programs are found while preserving the execution contract.

a) Discovery without delegated execution authority: This integration needs an explicit mapping from agent identity and AgentFacts to a package URL, version, and content digest. AgentFacts describes an agent; `agent.pkg` contains the program to execute. Neither a resolver result nor a signed capability assertion grants execution authority. The browser would still verify the downloaded artifact, evaluate trusted issuers, and obtain local consent. A certificate authority could bind publisher identities to signing keys, while evaluators attest to tested package properties [24]. Runtimes could accept these credentials under their own issuer policies, checking scope and validity rather than manually approving every key. Freshness, revocation, and conflicting registry information need defined failure behavior. Federated discovery could then connect public and organizational registries without making one index the universal runtime authority.

b) An ecosystem needs conformance, not just more packages: Open collaborative agent ecosystems require interoperability across messaging, interaction, state management, and discovery to avoid fragmentation across frameworks [1], [7]. WebAgents extends this interoperability to the execution boundary: independently published agent programs should move across creators, hosts, and runtimes while execution authority remains with the receiving runtime. Achieving this requires versioned schemas, stable execution-profile semantics, and conformance tests. Evaluation should identify the package digest, runtime configuration, and tested workload rather than attaching an unconditional quality label to an agent. Future work should execute unchanged packages across independently implemented runtimes and multiple model/tool configurations, measuring completion, resource use, and authorization violations. This would test substitutability across runtimes, extending the current model-binding and contract evaluations.

c) Broader execution and service integration: Conversational and persistent-service profiles require explicit state lifetimes, renewed consent, cancellation, and budgets. Executable modules need an additional isolation boundary beyond the current data-only format. MCP tools, A2A delegation, and legacy-web automation could enter through runtime adapters [8], [9]. A changed binding must be evaluated and authorized again, never substituted within an existing Run Plan. Broader data-flow controls inspired by CaMeL [21] and confirmations for consequential actions could further constrain the use of approved capabilities. Scoped persistent memory, external attestations, and real payment integration with monetary spending controls are additional extensions.

VII. Conclusion

We introduced WebAgents as an implementation of agent creation, hosting, and runtime separation. Its signed package carries the public program, while the OpenClaw-based runtime supplies and controls the authority to execute it. Three documented workflows show that the same packaged behavior can execute with both local and API-backed models, including an origin-published website agent using live services and runtime-held personal resources. In the controlled suite, WebAgents blocks all 716 injected violations and accepts all 295 benign controls, while the three-package benchmark measures median pre-execution costs of 59.0, 449.9, and 488.7 ms. Together, these results support the feasibility of the execution contract and its enforcement within the evaluated scope. The next challenge is to turn that working boundary into a shared contract across creators, hosts, runtimes, and discovery systems without treating portability as permission. More broadly, WebAgents points toward an open and interoperable Agentic Web in which reusable agent behavior can move across providers and runtimes while execution authority remains with the user.

AI Disclosure

ChatGPT and Claude were used to assist with language editing, improving manuscript clarity, and coding support. All technical content, experiments, analyses, and references were verified by the authors.

References

[1] R. Sharma, M. de Vos, P. Chari, R. Raskar, and A.-M. Kermarrec, "Position: Collaborative agentic AI needs interoperability across ecosystems," in *International Conference on Machine Learning (ICML), Position Paper Track*, 2026.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations*, 2023.

[3] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang *et al.*, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," in *Conference on Language Modeling (COLM)*, 2024.

[4] Y. Yang, M. Ma, Y. Huang, H. Chai, C. Gong, H. Geng, Y. Zhou, Y. Wen, M. Fang, M. Chen, S. Gu, M. Jin, C. Spanos, Y. Yang, P. Abbeel, D. Song, W. Zhang, and J. Wang, "Agentic Web: Weaving the Next Web with AI Agents," Jul. 2025.

[5] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, H. Sun, and Y. Su, "Mind2Web: Towards a Generalist Agent for the Web," in *Advances in Neural Information Processing Systems 36, Datasets and Benchmarks Track*, 2023.

[6] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "WebArena: A Realistic Web Environment for Building Autonomous Agents," in *International Conference on Learning Representations*, 2024.

[7] W. Xu, T. Wang, Y. Xia, S. Zhang, and S. C. Liew, "Agent-OSI: An Interoperability Architecture for Communication and Settlement in the Decentralized Internet of Agents," Jul. 2026.

[8] Model Context Protocol, *Model Context Protocol Specification*, 2026, revision 2026-07-28. [Online]. Available: https://modelcontextprotocol.io/specification/2026-07-28

[9] A2A Project, *Agent2Agent (A2A) Protocol Specification*, n.d., version 1.0.0. Accessed 26 September 2026. [Online]. Available: https://a2a-protocol.org/v1.0.0/specification/

[10] R. Raskar, P. Chari, M. Lambe, R. Lincourt, R. Bala, A. Joshi, J. J. Grogan, A. Singh, A. Chopra *et al.*, "Upgrade or switch: Do we need a next-gen trusted architecture for the internet of AI agents?" in *IEEE 7th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*, 2025, pp. 399–408.

[11] R. Raskar, P. Chari, J. Zinky, M. Lambe, J. J. Grogan, S. Wang, R. Ranjan, R. Singhal, S. Gupta *et al.*, "Beyond DNS: Unlocking the internet of AI agents via the NANDA index and verified AgentFacts," *arXiv preprint arXiv:2507.14263*, 2025.
[12] Agent Skills, *Agent Skills Specification*, n.d., accessed 26 September 2026. [Online]. Available: https://agentskills.io/specification
[13] J. Miao, J. R. Davis, Y. Zhang, J. K. Pritchard, and J. Zou, "Reimagining research papers as interactive and reliable AI agents," *Nature*, pp. 1–9, Sep. 2026.
[14] S. Amini, Y. Benajiba, C. Bernardis, P. Cayet, H. Chafi, A. Fathan, L. Faucon, D. Hilloulin, S. Hong, I. Kossyk, T. M. S. Le, R. Patra, S. Ravi, J. Schweizer, J. Singh, S. Singh, W. Sun, K. Talamadupula, and J. Xu, "Open Agent Specification (Agent Spec): A Unified Representation for AI Agents," Nov. 2025.
[15] Oracle, *JAR File Specification*, n.d., Java SE 25. Accessed 26 September 2026. [Online]. Available: https://docs.oracle.com/en/java/javase/25/docs/specs/jar/jar.html
[16] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos, "In-toto: Providing farm-to-table guarantees for bits and bytes," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1393–1410.
[17] Agent Packaging Standard Working Group, *Agent Packaging Standard (APS): Specification v0.1*, accessed 2 October 2026. [Online]. Available: https://agentpackaging.org/specs/APS-v0.1/
[18] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'Agents: Security in a Multiple-Language, Mobile-Agent System," in *Mobile Agents and Security*, G. Vigna, Ed. Berlin, Heidelberg: Springer, 1998, pp. 154–187.
[19] K. Mei, X. Zhu, W. Xu, M. Jin, W. Hua, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang, "AIOS: LLM Agent Operating System," in *Second Conference on Language Modeling*, Aug. 2025.
[20] H. Wang, C. M. Poskitt, and J. Sun, "AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents," in *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, ser. ICSE '26. New York, NY, USA: Association for Computing Machinery, 2026, pp. 2938–2950.
[21] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr, "Defeating prompt injections by design," in *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026.
[22] Y. Ruan, H. Dong, A. Wang, S. Pitis, Y. Zhou, J. Ba, Y. Dubois, C. J. Maddison, and T. Hashimoto, "Identifying the Risks of LM Agents with an LM-Emulated Sandbox," in *The Twelfth International Conference on Learning Representations*, 2024.
[23] E. Debenedetti, J. Zhang, M. Balunovic, L. Beurer-Kellner, M. Fischer, and F. Tramèr, "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents," in *The Thirty-Eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, Nov. 2024.
[24] World Wide Web Consortium, *Verifiable Credentials Data Model v2.0*, 2025, W3C Recommendation, 15 May 2025. [Online]. Available: https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/

APPENDIX A
PACKAGE-TO-RESULT LIFECYCLE

Algorithm 1 summarizes the contract in Section IV, including website-advertised acquisition. In the current prototype, a website can advertise an associated `agent.pkg` through an experimental same-origin `rel="agent-package"` link. NANDA discovery and issuer credentials remain future extensions. The lifecycle keeps artifact identity, resource authorization, and execution records distinct.

Algorithm 1 From package acquisition to an auditable result

Require: Package or website location, task input, and runtime policy; optionally a future NANDA identity.
Ensure: Preparation error or Run Record with status and available artifacts; existing plans unchanged.
1: **Discover.** Read an experimental same-origin website package link; *future:* resolve identity through NANDA to AgentFacts and a package location/version/digest. Treat either result as metadata, not authority.
2: **Acquire.** Snapshot bounded package bytes from HTTPS or an approved local artifact without executing the workflow.
3: **Verify.** Check container, hashes, signature, schemas, and execution profile. Reject invalid packages.
4: **Evaluate trust.** Require a scoped signer pin; future issuer credentials can inform trust without replacing verification.
5: **Authorize and bind.** Obtain permissions, tool approvals, and personal-resource selections. Choose a compatible, routable model; refuse unavailable required capabilities.
6: **Prepare.** Freeze package, signer/trust, receipt, model, tool bindings, and resource identifiers/digests in a Run Plan. Bind input in a Run Invocation.
7: **Revalidate.** Lease the snapshot; recheck identity, authority, bindings, resource grants, model availability, and limits. Refuse stale plans.
8: **Execute.** Run the signed graph with copied state and approved bridges, enforcing permissions, schemas, budgets, concurrency, and deadlines.
9: **Validate and commit.** Validate outputs and declared relationships; commit writes only after successful waves. Record progress and usage.
10: **Handle interruption.** Stop affected work and propagate cancellation where supported, without rolling back completed external side effects.
11: **Finalize.** Emit Run Record, audit, usage, provenance, and available artifacts. Publish successful outputs after checks; preserve the Run Plan.

APPENDIX B
CONTROLLED EVALUATION PROTOCOL

A. Violation construction and outcome rules

RQ1 (experiment 1.0.0, threat model 1.0, commit `23eac90`) uses Meeting Agenda 0.2.2 and Literature Review 0.2.5, not WebAir. Of 61 templates, 54 are enumerated and seven parametric. Seed `20261002` varies content edits, added members, context sizes, token-budget fractions, and schemas. Nine schema-mutation operators invalidate calibrated benign answers; duplicate parameters are rejected. Thirty-nine templates apply to both packages and 22 only to Literature Review, yielding 253 and 463 violations.

The eight families cover package tampering; untrusted/misscoped signers; missing, denied, replayed, or expiring consent; incompatible requirements or tool contracts; post-plan changes; unapproved capability use; malformed outputs or broken relations; and resource overruns or invalid usage. Each template fixes its assumption, refusal component/stage/codes, forbidden boundary, matched control, and ablation before the run. Controls restore the missing condition, use valid perturbations, or set limits at measured need; deduplication leaves 295 controls.

A boundary crossing is a silent violation even if an error follows. Otherwise, detection requires an explicit production refusal; unexplained failure is incidental. Refusal codes are checked against production source. Simulated-runtime refusals receive no credit; failed injections and harness errors are separate. No violation was detected only after its forbidden boundary.

B. Execution and ablation scope

The compiled plugin and `agentpkg` run from source with simulators for models, services, DNS, and PDF parsing. Workers refuse and audit real network, process, and canary-file access. The suite is offline and credential-free, with in-process control plane and runner; the runner uses the real bridge protocol. It tests contract logic, not OS isolation, live services, compromised runtimes, release rollback, side channels, availability, or independent-runtime interoperability.

Ablations use isolated source copies, exact-match Python/TypeScript patches, and control-plane policy switches. Workers verify their profile; no production option exposes ablations. Verification retains parsing/schema prerequisites, runtime mediation retains static workflow checks, and output validation retains JSON parsing/lookups. Unremoved sites also block four root-level schema mutations and one tool-timeout variant. These gaps limit ablation coverage: remaining detection does not imply protection after all checks of that type are removed.

All 716 refusals matched the predeclared stage; component attribution matched 710 and refusal codes 704. Six runtime-version cases are labeled package validation; six trust-withdrawal cases reach control-plane preflight before the publisher check. Boundary outcomes are unchanged. Development exposed under-charging of malformed usage reports, fixed before this run. Expectations were corrected and three templates split before freezing the catalog, without changing outcome rules. Smoke, development, and pre-fix runs are excluded.

Rates describe this finite catalog: variants share templates and the family mix determines the pooled rate. Controls rule out reject-all behavior here, not false rejections on arbitrary benign workloads.

C. Pre-execution timing protocol

RQ2 (benchmark 1.1.0, commit `9d78ca5`) uses the artifacts in Table IV. Only the new three-package series is pooled. A long-lived control plane uses `time.perf_counter_ns` with garbage collection enabled. Per run, 50 warmups precede five batches of 200 iterations per package, with package order randomized using seed `20261001`. Three runs yield 9,000 observations and 450 warmups, with none of the timed observations dropped. Code, configuration, fixtures, package hashes, and machine match across runs.

Benchmark fixtures describe 11 tool bindings and six personal resources. For WebAir, all seven bindings are approved but only the travel profile is granted; identity, payment, and purchase permissions remain denied. Grant/denial propagation is checked at the runner. Literature Review uses 0.2.4 in Table I and 0.2.5 in RQ1/RQ2.

Eight timestamps delimit package verification, authorization, model/tool compatibility, plan preparation, runtime revalidation/lease, runner re-verification, and workflow start. Table III groups stages 2–3 as authorization/binding and 5–7 as pre-execution revalidation. Groups and totals are directly timed; medians need not sum. Preconditions are checked outside timers. Guards reject model/tool calls, network access, and process creation. Human decisions are pre-recorded; file reads are untimed.

TABLE IV
SIGNED ARTIFACTS IN RQ2.

Property	Meeting Agenda	Literature Review	WebAir
Version	0.2.2	0.2.5	0.3.0
Archive bytes	21,580	224,302	167,193
Payload files	17	66	58
Graph nodes	7	25	34
Declared tools (remote / local)	0	3 (2 / 1)	7 (4 / 3)
Permissions (personal-resource)	2 (0)	5 (0)	7 (3)

Pooled p95 uses linear interpolation. Run-median coefficients of variation are 0.14%, 0.31%, and 0.41% for the three totals and at most 0.94% per stage. Runs began only when one-minute load was at most 3.0; none was discarded and no inference ran concurrently. Component measurements attribute most package-open time to schema/cross-reference validation; Ed25519 takes at most 0.39 ms. Absolute timings also depend on runtime configuration and machine/session conditions.

APPENDIX C
IMPLEMENTATION LESSONS

a) Resource assumptions are part of the contract: Earlier package versions passed deterministic fixtures but failed on the local model backend. Conservative token reservations exceeded declared budgets, and local reasoning exceeded deadlines chosen around hosted inference. Search breadth also exceeded output limits. The corrections changed bounded workload, timing, budgets, and skill instructions while preserving runtime checks. A transferable graph is therefore insufficient: a package also needs resource requirements that a runtime can assess. This makes incompatibility visible instead of hiding it behind a model substitution or relaxed limits.

b) Output contracts must cover composition and failure: An earlier review run reused local evidence identifiers across workers, causing the join to discard three genuine records. The current runs use paper-scoped evidence identifiers, avoiding this collision. An earlier query returned insufficient grounded evidence; validation rejected unsupported prose instead of producing a valid no-results artifact. These cases show why output contracts must cover cross-worker relationships and empty-result behavior, not only individual schemas. Such checks cannot establish review completeness or whether claims follow from the evidence.