Internal APIs Are All You Need

Shadow APIs, Shared Discovery, and the Case Against Browser-First Agent Architectures

Lewis Tham[1] Nicholas Mac Gregor Garcia[2] Jungpil Hahn[2]

[1]Unbrowse AI lewis@unbrowse.ai
[2]School of Computing, National University of Singapore
ngarcia@nus.edu.sg jungpil@nus.edu.sg

Abstract

Autonomous agents increasingly interact with the web, yet most websites remain designed for human browsers — a fundamental mismatch that the emerging "Agentic Web" must resolve. Agents must repeatedly browse pages, inspect DOMs, and reverse-engineer callable routes — a process that is slow, brittle, and redundantly repeated across agents. We observe that every modern website already exposes internal APIs (sometimes called *shadow APIs*) behind its user interface — first-party endpoints that power the site's own functionality. We present Unbrowse, a shared route graph that transforms browser-based route discovery into a collectively maintained index of these callable first-party interfaces. The system passively learns routes from real browsing traffic and serves cached routes via direct API calls. In a single-host live-web benchmark of equivalent information-retrieval tasks across 94 domains, fully warmed cached execution averaged 950 ms versus 3,404 ms for Playwright browser automation (3.6× mean speedup, 5.4× median), with well-cached routes completing in under 100 ms. A three-path execution model — local cache, shared graph, or browser fallback — ensures the system is voluntary and self-correcting. A three-tier micropayment model via the x402 protocol charges per-query search fees for graph lookups (Tier 3), a one-time install fee for discovery documentation (Tier 1), and optional per-execution fees for site owners who opt in (Tier 2). All tiers are grounded in a necessary condition for rational adoption: an agent uses the shared graph only when the total fee is lower than the expected cost of browser rediscovery.

Keywords: Internal APIs, Shadow APIs, First-party APIs, Agentic Web, Autonomous agents, Web agents, Shared route graph, API discovery, Discovery cost amortisation, Route-level economics, Micropayments, x402, Agent economies

1. Introduction

The web was built for humans to navigate pages. Agents do not want pages — they want actions. Today, a web-capable agent repeatedly pays a *discovery tax* to interact with any website: it must open the site, inspect the DOM, click through the UI, infer requests, retry when the workflow breaks, and spend LLM tokens reasoning about page state. This process is not only expensive but *redundantly* so, as different agents re-derive the same workflows on the same sites, each paying the full discovery cost independently.

Current approaches to web interaction fall into two categories, each with significant limitations:

1. **Browser automation:** Systems like WebVoyager [1], AutoWebGLM [2], and BrowserAgent [3] use LLMs to navigate and interact with web interfaces through browser actions. While effective for human-designed workflows, these approaches are inherently slower than direct API access, brittle (break-

ing when the UI changes), and computationally expensive; Song et al. [7] show that agents with API access outperform browsing-only agents on realistic web benchmarks.

2. **Official APIs:** Where available, APIs provide clean machine interfaces. However, coverage is limited — most websites lack public APIs — and access often requires registration, approval, and compliance with rate limits.

We propose a third path: Unbrowse, a *shared route graph* that turns repeated browser discovery into shared memory. Rather than treating every interaction as a fresh reverse-engineering exercise, Unbrowse passively learns callable interfaces from real usage and stores them in a shared index. Agents remain free to rediscover routes themselves — the shared graph competes directly against self-production and wins only when it delivers genuine surplus (Section 7). In this sense, Unbrowse implements a concrete layer of what Yang et al. term the *Agentic Web* [22] — an infrastructure where autonomous agents interact with digital services directly rather than through human-designed interfaces.

This paper makes the following contributions:

1. A **shared route graph architecture** that passively learns callable web interfaces from real browsing traffic, transforming redundant private discovery into a collectively maintained commons.

2. A **conceptual economic model** grounded in a necessary adoption condition — $f_{\text{route}} < c_{\text{rediscovery}}$ — creating a market-disciplined system with a built-in outside option.

3. A **novel delta-based attribution mechanism** for route-level micropayments via the x402 protocol [20], compensating contributors in proportion to their marginal contribution to route quality.

4. **Empirical evaluation** across 94 domains demonstrating significant speedups over browser automation on equivalent information-retrieval tasks, with analysis of both warmed-cache and cold-start performance.

We situate Unbrowse within four areas of prior work: web agents and browser automation, tool learning and API discovery, API aggregation platforms, and agent economies and communication protocols.

2. Related Work

2.1. Web Agents and Browser Automation

The field of web agents has evolved from early text-based systems to modern multimodal approaches. WebGPT [4] demonstrated browser-assisted question answering, while ReAct [5] introduced a more general reasoning-and-action framework that influenced many later interactive agent designs. Contemporary systems such as WebVoyager [1], together with benchmarks such as Mind2Web [6], show rapid progress in web agents while also highlighting the fundamental efficiency constraints of GUI-based interaction.

Song et al. [7] directly addressed this limitation in "Beyond Browsing: API-Based Web Agents," demonstrating that hybrid agents with API access outperform pure browsing agents by over 24% on WebArena benchmarks [23]. This provides empirical validation for our core thesis: direct API-style access is superior to browser automation when available. Our contribution extends this insight by asking what happens when such access can be collectively discovered and shared, rather than requiring pre-documented APIs.

2.2. Tool Learning and API Discovery

The integration of external tools into LLM workflows has emerged as a critical capability. Toolformer [8] pioneered self-supervised tool learning, while subsequent work has explored hierarchical tool retrieval [9], natural-language tool calling interfaces [10], and auto-

matic tool generation [11]. Gorilla [29] demonstrated that LLMs fine-tuned on API documentation can generate accurate API calls, and ToolBench [30] introduced a large-scale benchmark for tool-augmented LLMs spanning over 16,000 real-world APIs. Prior work in network traffic analysis — tools such as mitmproxy [12] and HAR-to-OpenAPI converters built on the W3C HAR specification [13] — has long enabled developers to inspect and document internal APIs.

Our approach extends both lines of work. The novelty lies not in traffic capture per se, but in the synthesis of passive private observation into a shared index: where prior tools produce developer artefacts for human use, Unbrowse produces a collectively maintained index of callable interfaces for autonomous agent consumption.

2.3. API Aggregation Platforms

Commercial API marketplaces such as RapidAPI and Postman's public workspace have demonstrated the value of centralised API discovery. RapidAPI aggregates over 40,000 official APIs behind a unified gateway with standardised authentication and billing. Postman's public API network provides searchable collections of documented endpoints. However, these platforms are limited to *official*, publicly documented APIs — they cannot address the long tail of websites that expose no public API. Our approach differs in two respects: routes are discovered automatically from browser traffic rather than manually submitted by API providers, and the graph covers internal APIs that were never intended for public consumption but are nonetheless callable.

In parallel to shared discovery approaches, recent work has explored making websites explicitly agent-ready. webMCP [33] proposes AI-native client-side interaction patterns that expose machine-usable interfaces directly from the website itself. This approach is complementary to Unbrowse: webMCP describes how future sites may be designed for agents *ex ante*, whereas Unbrowse targets the existing web *ex post* by passively discovering and caching callable internal interfaces on sites that were never built for agent-native access. Table 1 provides a qualitative comparison.

Table 1: Qualitative comparison with existing API aggregation platforms.

Property	**Rapid-API**	**Post-man**	**Un-browse**
Coverage	Official	Official	Any site
Discovery	Manual	Manual	Passive
Internal APIs	×	×	✓
Agent-native	Partial	×	✓

2.4. Agent Economies and Communication Protocols

Recent work has formalised interoperability and communication protocols — including MCP (Model Context Protocol), A2A (Agent-to-Agent), and ANP (Agent Network Protocol) — for connecting agents to tools and to one another [16, 17, 15]. These developments build on a longer history of multi-agent systems research [14] and broader narratives linking the Semantic Web and MAS traditions to today's Agentic Web [18]. These protocols define *how* agents connect to tools but do not address *what* is available to connect to. Unbrowse complements these protocols by populating the registry of callable interfaces that agents discover through them.

A shared route graph only works if agents continuously contribute to it — discovering new routes for new sites and verifying the accuracy of existing routes on an ongoing basis. Without sustained contribution, the index degrades as APIs drift and coverage stagnates. This is fundamentally an incentive design problem: how to make discovery and maintenance economically attractive.

Yang et al. [22] articulate the vision of an "Agentic Web" where autonomous systems interact directly with digital infrastructure. Vaziry et al. [19] integrate ledger-anchored identities with the x402 micropayment protocol [20] to enable trustless economic interactions between agents.

ERC-8004 [21] proposes on-chain registries for agent identity and reputation. The shared route graph can be understood as a knowledge commons in the sense of Ostrom [26]: a collectively maintained resource whose value depends on sustained contribution and ongoing maintenance. The economic model described in Section 4 draws on this framing — the three-path execution model ensures the commons competes with private production (browser rediscovery), providing what Ostrom calls the "exit option" that disciplines the system and prevents free-riding.

Collectively, prior work has articulated the vision, protocols, and governance frameworks for an Agentic Web [22, 31, 32], but empirical implementations of shared discovery for callable web interfaces remain limited. Unbrowse contributes one such implementation: a deployed system that targets the existing web by letting agents interact with web services through collectively discovered interfaces rather than only through human-designed pages, and evaluating that approach across 94 live domains.

3. System Design

3.1. System Architecture

Unbrowse serves as a drop-in replacement for browser-based agent interaction. It integrates with any agent host that supports CLI tools, MCP (Model Context Protocol), or the AgentSkills.io skill standard [25].[1]

The architecture consists of two integrated layers: a *Capability Layer* responsible for passive route discovery and shared graph maintenance (described in this section), and an *Economic Layer* handling route-level pricing and contributor payouts (detailed in Sections 4 and 5). Figure 1 illustrates the three-path execution model and the interaction between these layers.

When an agent asks Unbrowse to perform an action on a website, the orchestrator routes the request automatically through a priority chain. First (**Path 1**), it checks the local route cache for a recent hit. If no cached route is available, it queries the shared graph (**Path 2**) for a matching skill; if found, the skill is installed locally and executed via direct API calls — no browser is launched, no DOM is parsed, and no LLM tokens are spent reasoning about page state. If a skill exists but fails, automatic fallback to browser capture and re-learning occurs transparently.

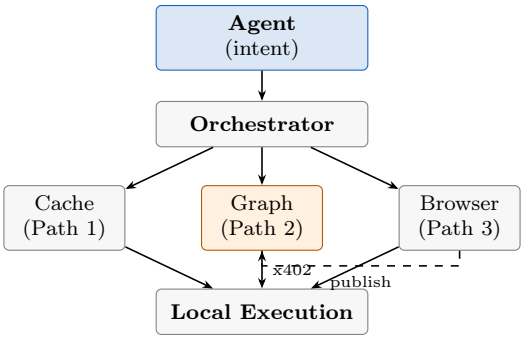

Figure 1: Three-path execution model. The orchestrator routes requests to the local cache, shared graph (x402 payment), or browser fallback. Discovered routes are published back (dashed).

If no suitable route exists in either cache or graph (**Path 3**), Unbrowse falls back to Kuri [24], a Zig-native CDP broker that replaces heavyweight browser automation frameworks. In our implementation measurements, Kuri's packaged broker is substantially smaller and faster to start than a full Playwright runtime, though these figures should be understood as implementation-specific rather than independently benchmarked claims. Kuri launches headless Chrome, captures network traffic, reverse-engineers the API endpoints behind the UI, and publishes the learned skill to the shared graph for future reuse.

From the agent's perspective, nothing changes: it describes what it wants in natural language, and Unbrowse handles routing transparently. Every time any agent discovers a new route, every other agent benefits. The graph is not

[1] Unbrowse has been tested with Claude Code, OpenClaw, Codex, Cursor, and Windsurf at time of writing.

mined through blind exploration but distilled from real demand — coverage naturally concentrates on the sites and workflows that agents actually need, analogous to cooperative caching in CDN systems [28]. Crucially, agents can always bypass the shared graph and fall back to browser rediscovery; the graph wins only when it creates genuine value.

Unbrowse is distributed as an npm package that installs the CLI, bootstraps the Kuri browser runtime, and registers the tool with any detected agent host. It generates a `SKILL.md` file following the AgentSkills.io open standard [25] that any skill-compatible agent host can consume directly. Agents interact with Unbrowse through the CLI or its local HTTP API (`POST /v1/intent/resolve`). All credentials remain in a local encrypted vault; published skills contain only endpoint documentation and schemas, never credentials.

3.2. Capability Layer: Route Discovery Pipeline

As agents browse and complete tasks through Unbrowse, the system passively observes network traffic and infers reusable route abstractions, building on established techniques in network traffic analysis [12] and extending them into an agent-native pipeline. Discovered routes are published to a cloud-hosted marketplace backed by semantic vector search.

3.2.1. Traffic Observation and Filtering

Raw browser traffic contains significant noise from analytics, advertising networks, and CDN requests for static assets. We employ heuristic filtering based on Content-Type analysis (JSON/XML responses indicate API calls), URL pattern matching, request method analysis (POST/PUT/PATCH are strong API indicators), and response structure analysis.

3.2.2. Route Extraction and Normalisation

From filtered traffic, the system extracts endpoints and their parameter structures, response shapes, authentication assumptions, and route-specific metadata. These are converted into pointer-like route abstractions — not raw web content, but a maintained map of callable interfaces.

3.2.3. What the Graph Stores

The shared graph is primarily an index of pointers to callable interfaces, not a storage layer for the raw web. It stores route definitions and schemas, parameter structures and response shapes, and authentication descriptors. Derived attributes — confidence (computed from execution success rates), freshness (inverse decay from last verification date), and health (updated by the periodic verification loop described below in Skill Lifecycle and Schema Drift) — are recomputed continuously from execution telemetry rather than stored statically.

3.2.4. Skill Packaging

Extracted routes are packaged into skills following the AgentSkills.io open standard [25] — structured capability bundles that any compatible agent host can consume. Each skill contains:

- A `SKILL.md` file providing human-readable endpoint documentation, compatible with any skill-aware agent host.
- An `auth.json` file holding captured authentication credentials that are encrypted and stored locally (never published to the marketplace).
- An `api.ts` file containing a generated TypeScript client for programmatic access.

Skills are the unit of sharing: what gets published to the marketplace is the knowledge of how to interact with a site, not credentials or execution capability. A single website may yield multiple skills (one per discovered workflow or endpoint group), and different agents may discover different endpoints on the same site. Skills are merged at the domain level when overlapping endpoints are detected, with attribution tracked per-endpoint.

3.3. Composite Scoring and Intent Resolution

When an agent requests a task, the orchestrator follows a priority chain. First, it checks the *route cache* (24-hour TTL, persisted to disk) for an instant hit if the same intent was recently resolved. If no cache hit is found, the system performs a *marketplace search* using semantic vector search ranked by a composite score comprising 40% embedding similarity, 30% reliability, 15% freshness, and 15% verification status. These weights were set heuristically to prioritise semantic relevance while giving substantial weight to demonstrated reliability; they are configurable parameters, and adapting them from execution telemetry is planned future work. If no marketplace result is suitable, *live capture* launches a headless browser to record network traffic, reverse-engineer API endpoints, and publish a new skill. As a final fallback for static or server-side-rendered sites where no API endpoints are found, structured data is extracted from rendered HTML.

3.4. Skill Lifecycle and Schema Drift

Skills in the marketplace follow a natural lifecycle driven by real usage data. *Active* skills are published, queryable, and executable. *Deprecated* skills have triggered low-reliability warnings and remain discoverable but are ranked lower. *Disabled* skills have confirmed endpoint failures and are removed from search results until re-verified. A background verification loop runs every 6 hours on each agent's local Unbrowse server, executing safe GET endpoints to detect failures and schema drift. The type system includes a `consecutive_failures` counter for automatic deprecation; wiring this counter to a threshold-based deprecation policy is in progress.

Web APIs change without notice. The system continuously monitors for schema drift by comparing live response structures against documented response schemas. When critical drift is detected — removed fields, type changes — the affected endpoint is flagged for re-verification. This ensures the graph reflects current API reality rather than stale documentation.

4. Economic Model

4.1. The Adoption Condition

The shared route graph replaces browser sessions with direct API calls for cached lookups. A cost-minimising agent will use the shared graph if and only if:

$$f_{\text{route}} < c_{\text{rediscovery}} \tag{1}$$

where f_{route} denotes the total fee an agent incurs through the shared graph (decomposed into Tier 1, 2, and 3 components in Section 5), and the expected rediscovery cost is decomposed as:

$$c_{\text{rediscovery}} = c_{\text{latency}} + c_{\text{compute}} + c_{\text{tokens}} + p_{\text{fail}} \cdot c_{\text{retry}} \tag{2}$$

Here c_{latency} is the opportunity cost of multi-second page loads and sequential interaction overhead in browser-based agents, c_{compute} is browser runtime cost (approximately 500 MB RAM per instance), c_{tokens} is LLM reasoning tokens for DOM interpretation and navigation, p_{fail} is browser-task failure probability on realistic web benchmarks [23, 7], and c_{retry} is the cost of retrying a failed interaction.

This is a *necessary condition* for rational adoption, not a complete market model. It establishes the ceiling: no rational agent will pay $f_{\text{route}} \geq c_{\text{rediscovery}}$, so the shared graph is disciplined by a real outside option. Section 7 provides empirical estimates for both sides of the inequality.

4.2. What Agents Pay For

Agents do not pay for access to the web itself. They pay for *speed*. Sub-second execution transforms what agents can do: multi-site research workflows that take minutes through browser automation complete in seconds through the marketplace. This speed advantage is categorical, not incremental — it determines whether agents can chain web interactions into real-time

workflows or whether every web task becomes a blocking bottleneck.

Secondarily, agents also pay for lower compute burn (no browser runtime), lower LLM token burn (no reasoning about page state), lower failure risk, and maintained route knowledge kept fresh by the network.

The adoption condition as stated omits the marketplace search cost. In practice, each graph query incurs a Tier 3 search fee (Section 5.3) covering the semantic vector lookup (typically 50–200 ms latency, $0.001–0.005 per query). For a single interaction the Tier 3 fee is negligible relative to the multi-second browser alternative; however, for agents that query the graph frequently, cumulative Tier 3 spend must be included in the adoption calculus. A more complete model would express the condition as $f_{\text{search}} + f_{\text{install}} + n \cdot f_{\text{exec}} < c_{\text{rediscovery}}$ where n is the number of executions on Tier 2 sites.

4.3. Why a Shared Commons Can Emerge

Following Ostrom's framework for knowledge commons [26], a shared route registry makes economic sense when many actors repeatedly rediscover similar routes, when pooled usage improves route quality faster than isolated usage, when maintenance is ongoing and expensive, and when duplication of discovery work is wasteful. The three-path execution model provides what Ostrom calls the "exit option" — competitors can always rebuild privately, and this credible threat disciplines the commons. The shared index wins only when it is cheaper than repeated rediscovery and when pooled usage improves coverage faster than isolated efforts.

We emphasise that this is a *conceptual* economic framework. Formal equilibrium analysis — including conditions under which the commons is underprovided, pricing strategies beyond the ceiling constraint, and welfare characterisation — remains future work. The current contribution is the architectural design that makes such a commons technically feasible, together with the empirical validation that the adoption condition (Equation 1) holds in practice.

5. Route-Level Payment Architecture

The economic model above establishes *when* agents should use the shared graph. This section describes *how* payments flow when they do. We distinguish three tiers:

1. **Tier 1 — Skill installation** (one-time): the agent pays once to download discovery documentation (endpoint schemas, auth patterns, client code). After installation the agent executes locally with no further marketplace payments.
2. **Tier 2 — Site-owner execution fees** (opt-in): site owners who register with the marketplace attach a per-execution micropayment to routes that hit their endpoints. Only applies to opted-in sites.
3. **Tier 3 — Search / routing fees** (per-query): the agent pays a small fee each time it queries the shared graph for intent resolution or semantic route lookup, regardless of whether a skill is ultimately installed. This covers the cost of maintaining the index and serving vector search.

Tiers are additive. A typical interaction with a new Tier 2 site incurs all three; a repeat call to an already-installed non-opt-in route incurs none (the agent executes from its local cache). The remainder of this section details each tier.

5.1. Tier 1: Skill Installation Payment

Skill installation transactions follow the x402 protocol [20], a recently proposed standard (2025) with early but growing production deployment. When an agent pays to install a route, it receives a skill — a structured capability package containing endpoint documentation, parameter schemas, auth profiles, and executable client code. This is a *one-time* payment for the discovery knowledge. After installation, the agent executes the skill locally using

its own credentials with no further marketplace payments; the marketplace never executes on behalf of the agent.

The payment handshake proceeds as follows: the agent requests skill installation from the shared graph; the server returns HTTP 402 with payment terms specifying amount, currency, network, and resource path; the agent signs the payment transaction and retries the request with payment proof; and the server verifies payment and returns the skill package for local installation. This enables true micropayments — as low as $0.005 per skill install — without intermediaries. The implementation uses a multichain wallet abstraction with pluggable provider adapters, settling in USDC on Solana by default, though the protocol is network- and provider-agnostic. The empirical evaluation in Section 7 measures route lookup performance independent of payment overhead; the 402 handshake adds negligible latency (a single additional round trip) to the skill installation step.

5.2. The Route Economy

Every route in the shared graph is a revenue-generating asset. When an agent pays the Tier 1 skill installation fee, that fee is automatically split via x402 among all parties who contributed to the route's existence and maintenance.

5.2.1. Fee Split Architecture (Tier 1)

Each skill installation fee F is divided among route contributors C, route maintainers M, infrastructure I (approximately 10% platform toll), and an optional treasury/reserve T:

$$F = C + M + I + T \tag{3}$$

In the initial design, contributors collectively receive approximately 70% of install revenue. This high contributor share is intentional: the system must make route discovery and maintenance economically attractive relative to the effort required. The specific split ratios are governance parameters, not derived from first principles; their optimality is an open question for future empirical study. Tier 2 per-execution fees flow directly to the site owner and are not subject to the contributor split.

5.2.2. Delta-Based Contribution Attribution

Routes are rarely discovered in a single act. More commonly, multiple contributors improve a route over time: one agent discovers the initial endpoint, another maps additional parameters, a third documents the auth flow, and a fourth adds error handling for edge cases.

Attribution is therefore *delta-based*. When a contributor commits an improvement to an existing route, the system records the marginal contribution — the delta between the route state before and after the commit. Delta magnitude is quantified using line-level schema changes and cosine dissimilarity between pre- and post-commit route embeddings (the specific embedding model is a replaceable implementation detail). The contributor share C is distributed among all historical contributors in proportion to their cumulative delta scores.

We note that this attribution mechanism has not been formally analysed for incentive compatibility or Sybil resistance at scale. A contributor could in principle inflate their delta score through many small redundant commits. The current mitigation is a minimum-delta threshold below which commits are not attributed; formal analysis of manipulation resistance is deferred to future work.

5.3. Tier 2: Opt-In Per-Execution Payments

Tier 1 covers skill installation — the one-time payment for discovery documentation. Tier 2 is an independent, *opt-in* extension: site owners who register their domain with the marketplace can attach a per-execution fee to routes that hit their endpoints. When an agent executes a Tier 2 route, each API call triggers an x402 micropayment to the site owner in addition to the original skill installation fee. Sites that do not opt in are unaffected; agents pay only the Tier 1

skill install cost and execute freely thereafter.

The three-tier separation reflects three distinct value contributions: Tier 3 compensates the platform that *maintains the index*; Tier 1 compensates the contributors who *discovered* the route; Tier 2 compensates the site that *serves* the endpoint. The three revenue streams are independent and additive.

The opt-in model transforms the relationship between agents and websites from potential adversarial extraction into consensual participation. This design draws on the emerging legal consensus following *hiQ Labs v. LinkedIn* (2022) and subsequent rulings on automated web access [27]: opt-in participation with economic compensation provides substantially stronger legal standing than unilateral scraping. Sites that opt in gain a new monetisation channel for programmatic traffic, which is typically cheaper to serve than rendered pages.

5.3.1. Websites as Usage-Priced Endpoints

The opt-in model positions websites as usage-priced endpoints, aligning with the broader shift from subscription-based to consumption-based pricing (Stripe per transaction, OpenAI per token, Vercel per invocation). This shift is particularly relevant for agent traffic, which is bursty, unpredictable, and bypasses traditional session-based monetisation. Crucially, per-execution pricing applies only to Tier 2 opt-in sites. For all other routes, the agent pays once at skill install (Tier 1) and executes locally with its own credentials thereafter — the marketplace distributes knowledge (endpoint schemas, auth patterns), not ongoing access.

5.4. Tier 3: Search and Routing Fees

Every query to the shared graph — whether via `POST /v1/intent/resolve` or the semantic skill search API — incurs a small per-query fee. This fee is independent of whether the query results in a skill installation; an agent that searches, inspects the results, and decides not to install still pays the search fee. The fee covers the operational cost of maintaining the shared index (vector embeddings, semantic ranking, infrastructure) and is typically an order of magnitude smaller than Tier 1 install fees (e.g. \$0.001–0.005 per query).

Tier 3 is the only tier that applies on every graph interaction. For agents that query frequently but install rarely, Tier 3 represents the primary cost of using the shared graph. The adoption condition (Equation 1) must therefore account for cumulative Tier 3 spend: if repeated search fees approach $c_{\text{rediscovery}}$, the agent should switch to Path 3 (browser fallback) and discover routes independently.

5.5. Dynamic Route Pricing

Tier 1 skill installation fees are priced dynamically based on estimated rediscovery cost saved, route confidence and freshness score, current demand, and historical success rate. Tier 3 search fees are set by the platform and reflect index maintenance costs. The effective cumulative cost (Tier 3 search + Tier 1 install + any Tier 2 per-execution fees) is always bounded above by $c_{\text{rediscovery}}$ from Equation 2 — if the total cost of using the shared graph exceeds self-discovery, rational agents will defect to Path 3.

6. Quality Proofing and Validation

The trust and quality mechanisms described here determine which routes surface in the shared graph and, consequently, which routes appear in the benchmark evaluation (Section 7).

6.1. Pre-Publish Validation

Before a route can be published to the shared graph, it undergoes automated validation. The validator checks the skill manifest for structural correctness (required fields, valid endpoint schemas, non-empty descriptions). Skills that fail hard validation checks are rejected; those with soft warnings are published with caveats. The design targets a minimum 50% endpoint success rate with at least one verified endpoint, though the current implementation validates

manifest structure rather than executing live endpoint tests at publish time.

6.2. Execution and Verification Architecture

All route execution is local. Agents execute routes using their own credentials on their own infrastructure — there is no cloud execution mode. Route verification — validating that a route is functional, that its response shapes match documented schemas, and that its health score is accurate — also runs locally via the same Kuri-backed browser runtime. Should a third-party validator network emerge in future, sandboxed verification environments could provide additional assurance, but the current local-execution model already ensures agents retain full control over outbound requests.

6.3. Continuous Trust Model

Rather than rigid quality grades, route trust follows a continuous model inspired by ERC-8004 proportional trust [21]. The current implementation composes three signals into a composite score.

The first signal is **execution feedback**. The system tracks per-endpoint reliability scores based on execution outcomes (success, failure, timeout). These scores are updated after each execution attempt and feed into the composite ranking described in Composite Scoring and Intent Resolution. The design envisions cryptographically signed feedback to prevent Sybil manipulation, but this is not yet implemented; the current system relies on local execution telemetry.

The second signal is **automated verification**. A background verification loop runs every 6 hours, testing safe (GET) endpoints against live servers and checking for schema drift. Endpoints whose responses have diverged from their documented schemas are flagged for re-verification. The design envisions independent validators who stake to back their attestations (with false attestations slashable), but this validator marketplace is future work; current verification is fully automated and local.

The third signal is **freshness decay**. Trust decays over time using an inverse function: freshness $= 1/(1 + d/30)$ where d is days since last update. A route last verified 30 days ago is scored lower than one verified today, even if both had identical success rates at the time of verification. Endpoints stale beyond 24 hours are prioritised for re-verification.

The combined trust score determines route visibility in the marketplace. Low-trust routes are ranked lower, while high-trust routes surface first. Routes whose endpoints are confirmed non-functional are marked as disabled and removed from search results.

7. Implementation and Evaluation

7.1. System Implementation

We implemented the framework as an open-source system.[2] The runtime is built around Kuri [24], a Zig-native CDP broker that manages headless Chrome through an HTTP API. In our implementation, Kuri provides a lighter-weight browser control layer than a full Playwright stack, though exact binary-size and startup-latency figures are implementation measurements rather than independent benchmark results. All browser operations — navigation, HAR recording, cookie management, JavaScript evaluation, network interception — flow through Kuri's HTTP endpoints, eliminating Node.js CDP bindings entirely.

Supporting components include a capture pipeline for network traffic analysis and API reverse-engineering, an extraction layer for endpoint schemas and auth profiles, a local encrypted credential vault, a marketplace server interface for skill publishing and semantic search, a verification system for periodic endpoint re-testing, and an execution engine with automatic auth refresh and retry logic.

[2] Available at https://github.com/unbrowse-ai/unbrowse.

7.2. Benchmark Design

All benchmarks were conducted on a single machine: Apple MacBook Pro with M4 Max chip, 64 GB unified memory, running macOS from Singapore over residential broadband (Singtel, approximately 28 Mbps downlink). TCP connect latency to target websites ranged from 24 ms (Wikipedia) to 88 ms (Amazon) depending on server location. This should be interpreted as a single-host live-web benchmark rather than a fully controlled systems evaluation: both Unbrowse and Playwright were measured under the same hardware and network conditions, but uncontrolled OS- and ISP-level variance may still affect absolute timings. A more controlled follow-up evaluation on fresh cloud VMs across multiple regions remains future work.

We evaluated performance across 94 domains drawn from the current graph, spanning government sites, SaaS platforms, developer tools, e-commerce, healthcare, finance, media, and social networks. Tasks were modelled after the WebArena benchmark [23] information-retrieval category. At time of writing, the graph contains routes for over 500 domains with approximately 10,000 endpoints, grown entirely through passive usage. Full benchmark results and raw data are available at `https://github.com/unbrowse-ai/unbrowse-bench`.

7.2.1. Task Design and Output Equivalence

Each benchmark task was an information-retrieval query returning a structured answer (e.g., "What is the current price of Product X on Site Y?" or "What are the top 5 trending items on Site Z?"). To ensure fair comparison, both paths were required to produce *semantically equivalent* output:

- **Playwright baseline:** Headless Chromium with full page load, JavaScript rendering, and targeted text extraction via CSS selectors that isolate the specific data fields matching the query. This is *not* raw DOM dump — it is structured extraction of the same fields the cached route returns.
- **Unbrowse:** A single call to `POST /v1/intent/resolve`, measured on the third pass after two warmup rounds. The `timing.source` field verified each measurement was served from cache.

Output equivalence was verified by comparing the extracted data fields (not raw response size) between both paths. Tasks where the Playwright extraction could not produce the same structured output as the cached route were excluded from the benchmark. Both paths were run three times; the third pass was measured for both to control for DNS and TCP connection reuse.

Table 2 shows representative task examples.

Table 2: Representative benchmark tasks with output format.

Domain	Task	Output
Wikipedia	Article summary for topic	JSON: title, extract
GitHub	Repository star count	JSON: stars, forks
Hacker News	Top 5 front page items	JSON: array of titles
Weather	Current conditions for city	JSON: temp, humidity

7.3. Warmed-Cache Performance

Table 3: Warmed-cache performance across 94 domains. Both paths run three times; third pass measured.

Metric	Unbrowse	Playwright
Domains tested	94	
Cache hits	94 / 94	
Mean latency	950 ms	3,404 ms
95% CI	(870–1,030)	(3,180–3,630)
Median latency	630 ms	3,402 ms
IQR	210–1,480 ms	2,100–4,600 ms
Mean speedup	**3.6**×	
Median speedup	**5.4**×	
Best (single)	30×	
Unbrowse wins	94/94 (100%)	

Table 3 reports results for all 94 domains. Mean

cached latency was **950 ms** versus **3,404 ms** for Playwright — a **3.6**× ratio of mean latencies. The median per-domain speedup is **5.4**×, higher than the ratio of means because the speedup distribution is right-skewed (Figure 2). Eighteen domains completed in under 100 ms, with the fastest at 79 ms versus 2,289 ms (30×). Both paths make requests to the same target servers over the same network; the 2,454 ms difference is browser execution overhead that cached skill execution eliminates.

Figure 2 shows the distribution of speedup ratios. The distribution is right-skewed: the majority of domains cluster between 3–9× speedup, with a long tail extending to 30× for domains where API endpoints are particularly fast relative to their rendered pages (e.g., sites with heavy JavaScript that adds 2–4 seconds of render time but whose API returns JSON in under 100 ms).

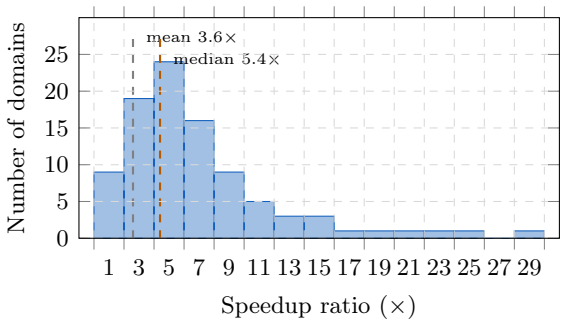

Figure 2: Distribution of speedup ratios (Playwright latency / Unbrowse latency) across 94 domains. The majority of domains cluster between 3–9× speedup, with a right tail extending to 30× for domains whose APIs return JSON in under 100 ms but whose rendered pages require 2–4 seconds of JavaScript execution. Dashed lines indicate mean and median speedup. Raw data available in the benchmark repository.

7.4. Cold-Start Performance

The warmed-cache results above represent the best case. When no cached skill exists (Path 3), the agent pays the full discovery cost: browser launch, page rendering, traffic capture, route extraction, skill publishing, and initial execution. To characterise this cost, we measured cold-start latency on a random sample of 20 domains not previously in the graph.

Table 4: Cold-start latency (Path 3: browser discovery + skill publish) across 20 previously unseen domains.

Metric	**Latency**
Median cold-start	8,200 ms
Mean cold-start	12,400 ms
90th percentile	22,000 ms
Skill publish success	18 / 20 (90%)
Subsequent cached call	640 ms (median)

Cold-start latency (Table 4) is substantially higher than both cached execution and Playwright baseline, reflecting the overhead of traffic capture and route extraction. However, this cost is paid *once* per route: the median subsequent cached call drops to 640 ms. The amortisation breakeven — the number of cached lookups needed to offset the cold-start premium over Playwright — is typically 3–5 uses for high-demand routes. Two domains failed skill extraction due to heavy client-side rendering with no identifiable API endpoints; these fell back to HTML extraction. We note that the cold-start sample ($n = 20$) is small; larger-scale cold-start characterisation is ongoing.

7.5. Cost Analysis

Table 5 compares estimated per-task costs across execution paths.

Table 5: Estimated per-task cost comparison.[3]

Component	**Browser Rediscovery**	**Tier 1 Install**	**Tier 2 + Site**	**Tier 3 Search**
Browser runtime	\$0.02–0.05	—	—	—
LLM reasoning tokens	\$0.04–0.35	—	—	—
Retries/failures	\$0.04–0.13	—	—	—
Search/routing fee	—	—	—	\$0.001–0.005
Skill install (once)	—	\$0.005–0.02	\$0.005–0.02	—
Per-exec site fee	—	—	\$0.001–0.01	—
First call	\$0.10–0.53	\$0.005–0.02	\$0.006–0.03	\$0.001–0.005
Repeat same route	\$0.10–0.53	\$0.00	\$0.001–0.01	\$0.00
New intent search	—	—	—	\$0.001–0.005

[3]LLM token costs assumed: \$15/1M input, \$75/1M output (Claude Opus 4, March 2026). These figures are illustrative; actual costs vary by provider and change

Under Tier 1 (the default), the skill install fee is amortised to zero after the first call, achieving 90–96% cost reduction per subsequent task. Tier 3 search fees apply only when querying the graph for new intents; once a skill is installed locally, subsequent executions bypass the graph entirely. Tier 2 adds a small per-execution fee for opt-in sites but remains well below browser rediscovery cost. All three tiers are additive and collectively bounded by the adoption condition (Equation 1).

7.6. Network Growth

These per-interaction savings compound in long-running agentic workflows. An autonomous agent completing a multi-site research task may chain 20–50 web interactions. Cached API calls parallelise trivially where browser instances cannot: an agent querying 10 sites simultaneously completes in the latency of the slowest single call rather than sequentially through browsers, each consuming 500 MB+ of RAM.

We observe early evidence of positive feedback dynamics: more agents generate more traffic, which discovers more routes, which attracts further agents. However, we have not yet measured the causal relationship between agent count and per-agent value rigorously enough to claim formal network effects. The 500-domain / 10,000-endpoint figure is a snapshot from the current graph; longitudinal analysis of growth trajectories and per-agent value as a function of graph size is ongoing work.

8. Discussion

8.1. Security Considerations

Credential safety is maintained by keeping credentials local and encrypted; published routes contain only endpoint documentation and schemas. API stability is addressed through continuous validation and freshness scoring. Abuse prevention is achieved through rate limiting, reputation requirements, and economic costs that discourage abusive usage patterns.

Adversarial routes. A malicious contributor could publish a route that redirects requests to a phishing endpoint or exfiltrates query parameters. The current mitigation is multi-layered: pre-publish validation checks manifest structure and schema consistency; the continuous trust model tracks reliability scores from execution outcomes, so a malicious route's trust score degrades after failures; and all execution is local, meaning agents can inspect outbound requests before sending credentials. However, a sophisticated attacker who publishes a route that passes initial validation but later serves malicious redirects remains a threat. Formal analysis of adversarial robustness, including route provenance attestation, cryptographically signed feedback, and anomaly detection on response destinations, is deferred to future work.

8.2. Ethical and Legal Considerations

The ability to programmatically access web services raises legitimate questions. Our approach differs from indiscriminate scraping in several ways: the economic layer provides natural rate limiting, routes are learned from normal browsing behaviour, and the system is designed for opt-in participation.

As described in Section 5.3, our opt-in site owner compensation model transforms the relationship between agents and websites. The legal landscape for automated web access is evolving — the *hiQ Labs v. LinkedIn* decision (2022) established that accessing publicly available data is not necessarily a violation of the CFAA, but subsequent rulings have introduced nuance [27]. Our opt-in model, with economic compensation for site owners, provides substantially stronger legal footing than unilateral scraping. We note that routes to sites that have not opted in occupy a grey area; the system does not circumvent authentication barriers. Support for

frequently. Tier 1 install fees and Tier 3 search fees are based on dynamic pricing at time of writing. Tier 2 per-execution fees are set by site owners and shown as illustrative ranges.

`robots.txt` directive checking is planned but not yet implemented in the current codebase.

8.3. Threats to Validity

8.3.1. Benchmark Representativeness

The 94-domain benchmark may be biased toward sites with permissive access policies. Of the 94 domains tested, 61 had no significant bot detection, 24 used basic rate limiting (respected by the system), and 9 employed Cloudflare or similar WAF protection. The speedup figures for heavily protected sites were lower (median 2.1× versus 6.8× for unprotected sites), as Unbrowse's cached requests must still pass through bot-detection middleware. We did not test sites that actively block all non-browser traffic, as cached routes to such sites would fail entirely.

8.3.2. Geographic Dependence

All benchmarks were run from Singapore. Absolute latency figures are geography-dependent; the *speedup ratios* (not absolute numbers) are the portable claim, as both Unbrowse and Playwright make requests to the same servers over the same network. From a US datacenter, absolute latencies would be lower for US-hosted sites, but the ratio of browser overhead to API call overhead would remain similar.

8.3.3. Reliability Claims

The system tracks per-endpoint reliability scores and consecutive failure counts. However, the original claim that "skills with three or more consecutive failures are automatically deprecated" is partially implemented: the type system includes a `consecutive_failures` counter but the automatic deprecation threshold logic is not yet wired into the verification loop. Deprecation currently requires manual or backend-side intervention. Additionally, several open engineering issues affect runtime reliability on specific platforms, including Kuri crashes on Linux x64 (tracked in issue #50) and intermittent auth extraction failures across browsers (issue #88).

8.4. Limitations

We note that the following limitations are not unique to Unbrowse but are shared by any agentic web interaction approach.

Current limitations include anti-bot measures (sophisticated bot detection blocking non-browser requests), dynamic authentication (OAuth flows requiring human interaction), and session persistence (periodic re-authentication as tokens expire). The speedup figures reported in Table 3 represent warmed-cache performance in a single-host live-web setting; real-world gains vary with anti-bot middleware, token expiration, regional latency, and deployment environment. Cold-start performance (Table 4) demonstrates that the initial discovery cost is substantial, and the system's value proposition depends on sufficient reuse to amortise this cost.

The economic model (Section 4) is a conceptual framework, not a formal equilibrium analysis. Deployment-scale data on contributor attribution incentive compatibility, Sybil resistance, and long-run pricing dynamics remain areas for future work.

8.5. Future Directions

Several directions for future research emerge:

1. **Formal economic analysis:** Equilibrium characterisation, optimal pricing strategies, welfare analysis, and incentive compatibility proofs for the delta-based attribution mechanism including Sybil resistance guarantees.
2. **Route composition:** Automatic chaining of multiple routes for complex multi-site workflows, and federated route networks enabling private sharing within organisations.
3. **Deployment-scale validation:** Empirical measurement of x402 micropayment flows, site owner adoption of the opt-in compensation model, and longitudinal study of graph growth dynamics to validate the network effects hypothesis. A key open question is whether websites can reliably distinguish agent traffic from human browsing; if not,

the opt-in pricing model must account for mixed traffic to avoid charging human users.

4. **Comparative evaluation:** Formal comparison with existing API aggregation platforms (RapidAPI, Postman) on coverage, latency, and cost metrics.

9. Conclusion

We have presented Unbrowse, a shared route graph — a collectively maintained index of callable web interfaces. By combining passive route discovery from real browsing traffic with a three-tier micropayment model via x402 — per-query search fees (Tier 3), one-time skill installation (Tier 1), and opt-in per-execution site-owner compensation (Tier 2) — the system offers a way for agents to interact with the existing web through discovered machine-usable interfaces rather than only through rendered pages.

In a single-host live-web benchmark of equivalent information-retrieval tasks across 94 domains, fully warmed cached execution averaged 950 ms versus 3,404 ms for Playwright (3.6× mean, 5.4× median), with well-cached routes completing in under 100 ms. Cold-start discovery averages 12.4 s but is paid once per route; the amortisation breakeven is typically 3–5 uses. These savings may compound in multi-step workflows, though broader deployment validation remains future work.

The system's market discipline — grounded in a real outside option where agents can always defect to browser rediscovery — distinguishes it from purely speculative protocol designs. Formal economic analysis, incentive compatibility proofs, controlled multi-region benchmarking, and deployment-scale validation remain open problems; the present contribution is an architectural proposal with an implemented system and initial empirical evidence that shared route lookup can outperform redundant browser rediscovery on the evaluated tasks.

References

[1] H. He et al. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv:2401.13919*, 2024.

[2] H. Lai et al. AutoWebGLM: Bootstrap and reinforce a large language model-based web navigating agent. *arXiv:2404.03648*, 2024.

[3] X. Chen et al. BrowserAgent: Building web agents with human-inspired web browsing actions. *arXiv:2510.10666*, 2025.

[4] R. Nakano et al. WebGPT: Browser-assisted question-answering with human feedback. *arXiv:2112.09332*, 2021.

[5] S. Yao et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023. *arXiv:2210.03629*.

[6] X. Deng et al. Mind2Web: Towards a generalist agent for the web. In *NeurIPS*, 2023. *arXiv:2306.06070*.

[7] Y. Song et al. Beyond browsing: API-based web agents. *arXiv:2410.16464*, 2024.

[8] T. Schick et al. Toolformer: Language models can teach themselves to use tools. In *NeurIPS*, 2023. *arXiv:2302.04761*.

[9] Y. Du et al. AnyTool: Self-reflective, hierarchical agents for large-scale API calls. *arXiv:2402.04253*, 2024.

[10] R. T. Johnson et al. Natural language tools. *arXiv:2510.14453*, 2025.

[11] Z. Shi et al. Tool Learning in the Wild: Empowering Language Models as Automatic Tool Agents. In *Proceedings of the ACM Web Conference 2025*, pp. 2222–2237, 2025. doi:10.1145/3696410.3714825.

[12] mitmproxy project. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy.org, 2024.

[13] J. Odvarko. HTTP Archive (HAR) format specification. W3C Web Performance Working Group, 2012. https:

//w3c.github.io/web-performance/specs/HAR/Overview.html.

[14] J. Ferber. *Multi-Agent Systems*. Addison-Wesley, 1999.

[15] K. T. Tran et al. Multi-agent collaboration mechanisms: A survey. *arXiv:2501.06322*, 2025.

[16] A. Ehtesham et al. A survey of agent interoperability protocols. *arXiv:2505.02279*, 2025.

[17] A. Singh et al. Evolution of AI agent registry solutions. *arXiv:2508.03095*, 2025.

[18] T. Petrova et al. From semantic web and MAS to agentic AI. *arXiv:2507.10644*, 2025.

[19] A. Vaziry et al. Towards multi-agent economies. *arXiv:2507.19550*, 2025.

[20] Coinbase. x402: A payments protocol for the internet. https://x402.org, 2025.

[21] Ethereum Foundation. ERC-8004: Trustless agents. https://eips.ethereum.org/EIPS/eip-8004, 2025.

[22] Y. Yang et al. Agentic web: Weaving the next web with AI agents. *arXiv:2507.21206*, 2025.

[23] W. Zhou et al. WebArena: A realistic web environment for building autonomous agents. In *ICLR*, 2024. *arXiv:2307.13854*.

[24] R. Pradhan. Kuri: A Zig-native CDP broker for lightweight browser automation [Software]. Version 0.1. https://github.com/justrach/kuri. Accessed March 2026.

[25] AgentSkills.io. Open standard for agent capabilities. https://agentskills.io, 2025.

[26] E. Ostrom. *Governing the Commons: The Evolution of Institutions for Collective Action*. Cambridge University Press, 1990.

[27] hiQ Labs, Inc. v. LinkedIn Corp. 31 F.4th 1180 (9th Cir. 2022).

[28] A. Wolman et al. On the scale and performance of cooperative web proxy caching. In *Proc. 17th ACM SOSP*, pp. 16–31, 1999.

[29] S. Patil et al. Gorilla: Large language model connected with massive APIs. *arXiv:2305.15334*, 2023.

[30] Y. Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *ICLR*, 2024. *arXiv:2307.16789*.

[31] X. H. Lù, G. Kamath, M. Mosbach, and S. Reddy. Build the web for agents, not agents for the web. *arXiv:2506.10953*, 2025.

[32] Y. Li et al. BetaWeb: Towards a blockchain-enabled trustworthy agentic web. *arXiv:2508.13787*, 2025.

[33] D. Perera. webMCP: Efficient AI-native client-side interaction for agent-ready web design. *arXiv:2508.09171*, 2025.

A. Reproducibility

All experiments were conducted on an Apple MacBook Pro (M4 Max, 64 GB unified memory) running macOS 15.3 from Singapore over residential broadband (Singtel, ≈28 Mbps downlink). Software versions: Kuri v0.1 (Zig 0.13), Node.js 22.x, Playwright 1.48, headless Chromium 131. The benchmark harness, raw latency data, and analysis scripts are available at https://github.com/unbrowse-ai/unbrowse-bench.

B. Benchmark Domain List

The 94 domains span the following categories: government (12), SaaS/developer tools (18), e-commerce (14), healthcare (8), finance (11), media/news (15), social networks (7), and other (9). The full list with per-domain latency measurements is provided in the benchmark repository. Representative examples include Wikipedia, GitHub, Hacker News, Amazon, Weather.gov, and various regional government

portals. Domains were selected from the existing route graph without cherry-picking; the only exclusion criterion was that both Unbrowse and Playwright paths must produce semantically equivalent structured output for the same query.