Shadow APIs Are All You Need

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

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

Abstract

Autonomous web agents repeatedly pay a *discovery tax*: opening sites, inspecting DOMs, and reverse-engineering callable routes—a process that is slow, brittle, and redundantly repeated across agents. We present **Unbrowse**, a shared route graph that converts browser-based route discovery into a collectively maintained, usage-priced index of callable web interfaces. Routes are learned passively from real browsing traffic and served as cached API calls. A three-path execution model—local cache, shared graph, or browser fallback—ensures participation is voluntary: agents use the graph only when its fee undercuts the expected cost of rediscovery ($f_{\text{route}} < c_{\text{rediscovery}}$). Route-level micropayments via the x402 protocol compensate contributors in proportion to marginal utility through delta-based attribution.

In a controlled benchmark across 94 domains, warmed-cache execution averaged 950 ms versus 3,404 ms for Playwright browser automation (3.6× mean speedup, 5.4× median), with 90–96% cost reduction per task. Cold-start discovery averages 12.4 s but amortises within 3–5 reuses. More broadly, the architecture inverts the classical SaaS relationship: shadow APIs become the primary machine-native interface, and the browser is demoted to a compatibility shim.

Keywords: Agentic Web, Web agents, Shared route graph, API discovery, Micropayments, x402, Agent economies

1. Introduction

The web was built for humans to navigate pages. Agents do not want pages—they want callable endpoints. Today, every web-capable agent independently pays a *discovery tax*: it opens a site, inspects the DOM, infers the underlying API, and retries when the workflow breaks. Different agents re-derive the same workflows on the same sites, each absorbing the full cost.

We term these undocumented but callable backend endpoints *shadow APIs*: the HTTP routes that every modern web application uses internally to serve its own frontend, which were never published as official developer interfaces but are nonetheless fully functional, stable, and machine-callable. Shadow APIs are distinct from both official public APIs (which are intentionally documented) and from deprecated or hidden debug endpoints; they are the production data-transport layer that the site itself relies on, making them as stable as the product they serve.

Existing approaches attempt to bridge the gap through additional abstraction layers:

1. **Browser automation.** Systems such as WebVoyager [1], AutoWebGLM [2], and BrowserAgent [3] interpret screenshots and manipulate DOMs via LLMs. Complete agent actions take 10–45 s including LLM reasoning [7], break when UIs change, and consume substantial compute.

2. **Official APIs and protocol adapters.** Where available, official APIs provide

clean machine interfaces, but coverage is sparse. The recent proliferation of integration protocols—MCP servers, A2A adapters, bespoke API wrappers—each expose a single service behind a new abstraction, requiring per-service integration work that scales linearly with coverage goals.

We propose a third path: a *shared route graph* that converts repeated browser discovery into collective memory. Unbrowse passively learns callable interfaces from real usage and stores them in a shared, usage-priced index. The graph competes directly against self-production—users can always rediscover routes themselves—and wins only when it delivers genuine surplus (Section 7). This realises a concrete layer of what Yang et al. term the *Agentic Web* [22]: infrastructure where agents interact with services through collectively discovered interfaces rather than human-designed pages.

This paper contributes:

1. A **shared route graph architecture** that passively indexes callable web interfaces from real traffic, converting redundant private discovery into a collectively maintained commons.
2. A **market-disciplined economic model** grounded in a necessary adoption condition—$f_{\text{route}} < c_{\text{rediscovery}}$—with a built-in outside option.
3. A **delta-based attribution mechanism** for route-level micropayments via x402 [20], compensating contributors by marginal contribution to route quality.
4. **Empirical evaluation** across 94 domains showing 3.6× mean speedup (5.4× median) over browser automation on equivalent information-retrieval tasks, with 90–96% cost reduction.

We situate Unbrowse within five areas of prior work and examine the broader architectural consequences in Section 8.5.

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 that LLMs could interact with web environments, while ReAct [5] generalised these capabilities through structured reasoning traces. Contemporary systems like WebVoyager [1] and Mind2Web [6] achieve impressive benchmark results but face fundamental efficiency constraints inherent to 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 descriptions [10], and automatic 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 shadow APIs.

Our approach extends both lines of work. The novelty lies not in traffic capture per se, but

in the synthesis of passive observation into a shared, usage-priced graph: where prior tools produce developer artefacts for human use, our framework 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 shadow APIs that were never intended for public consumption but are nonetheless callable. Table 1 provides a qualitative comparison.

Table 1: Qualitative comparison with existing API aggregation and agent integration approaches.

Property	Rapid-API	Postman	MCP	Unbrowse
Coverage	Official	Official	Per-server	Any site
Discovery	Manual	Manual	Manual	Passive
Shadow APIs	×	×	×	✓
Per-use price	✓	×	×	✓
Agent-native	Partial	×	✓	✓
Scales w/o code	✓	×	×	✓
Contrib. comp.	×	×	×	✓

2.4. Multi-Agent Systems and Communication Protocols

Coordination of multiple autonomous agents has been studied extensively in both classical AI [14] and modern LLM-based systems [15]. Recent work has formalised communication protocols including MCP (Model Context Protocol), A2A (Agent-to-Agent), and ANP (Agent Network Protocol) for decentralised discovery [16, 17]. Petrova et al. [18] trace the intellectual lineage from early Semantic Web efforts to modern agentic systems, identifying a paradigm shift where the locus of intelligence has moved from external data to the agent's core model.

2.5. Agent Economies and Micropayments

The economic dimension of multi-agent systems has received increasing attention. 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 to enable trustless economic interactions between agents. The x402 protocol [20], developed by Coinbase and Cloudflare, revives the HTTP 402 "Payment Required" status code to enable native internet payments; as a recently proposed standard (2025), its long-term adoption remains to be seen, though early integrations are promising. ERC-8004 [21] establishes on-chain registries for agent identity and reputation; it is similarly early-stage.

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 governance. 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 the "exit option" that disciplines pricing and prevents rent extraction. Our work connects these economic primitives to a concrete use case: route-level payment for shared discovery work.

Collectively, prior work has articulated the vision, protocols, and governance frameworks for an Agentic Web [22, 31, 32], but no existing system provides a working implementation

with empirical validation. Unbrowse bridges this gap: to our knowledge, it is the first deployed system that realises the core Agentic Web premise — agents interacting with web services through collectively discovered interfaces rather than human-designed pages — and demonstrates its viability with benchmarks 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, and an *Economic Layer* handling route-level pricing, x402 micropayments, and contributor payouts. Figure 1 illustrates the three-path execution model and the interaction between these layers.

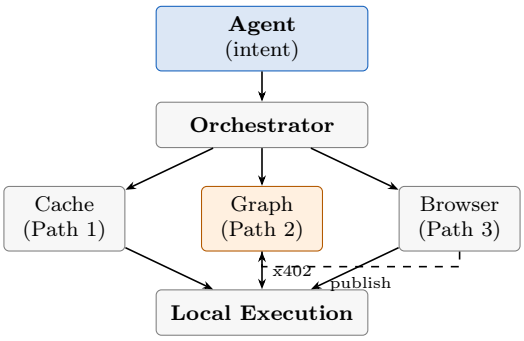

Figure 1: Three-path execution model. The orchestrator routes requests to the local cache, shared graph (x402 payment), or browser fallback. Discovered routes are published back (dashed).

When an agent asks Unbrowse to perform an action on a website, the system first checks whether a known route already exists in the shared marketplace. If a high-confidence skill is available, execution happens instantly via direct API calls — no browser is launched, no DOM is parsed, and no LLM tokens are spent reasoning about page state.

If no skill exists, Unbrowse falls back to Kuri [24], a Zig-native CDP broker that replaces heavyweight browser automation frameworks (464 KB binary, 3 ms cold start versus 80–150 MB and 1–3 s for Playwright). Kuri launches headless Chrome, captures network traffic, reverse-engineers the API endpoints behind the UI, publishes the learned skill to the marketplace, and executes it. If a skill exists but fails, automatic fallback to browser capture and re-learning occurs without agent intervention.

From the agent's perspective, nothing changes: it describes what it wants in natural language, and Unbrowse handles routing transparently. Every time any agent learns a new site, every other agent benefits.

3.2. Passive Indexing Through Normal Use

The graph grows passively from real usage: every time an agent uses Unbrowse to interact with a site for the first time, the capture pipeline records what happened, reverse-engineers the underlying API, and publishes it. The graph is not mined through blind exploration but distilled from real demand. Coverage naturally concentrates on the sites and workflows that agents actually need — high-demand sites accumulate many skills quickly, niche sites get indexed when someone actually needs them, and the graph reflects what is economically relevant rather than what a crawler guessed might matter. This demand-driven growth pattern is analogous to cooperative caching in CDN systems [28], where cache content is shaped by actual request patterns rather than speculative pre-fetching.

[1] Tested with Claude Code, OpenClaw, Codex, Cursor, and Windsurf at time of writing.

3.3. Integration Surface

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.

This design positions the shared route graph as a coverage layer that obviates per-service integration. Protocols like MCP and A2A define how agents connect to tools, but each integration exposes a single service and requires manual authoring—a developer must build and maintain an MCP server for every website an agent might need. The shared route graph replaces this $O(n)$ integration effort with a single interface backed by a collectively discovered registry: one tool grants access to the entire graph, and coverage grows automatically from real usage rather than manual wrapping.

3.4. Capability Layer: Passive Route Discovery

As users 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.

3.4.1. 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.

3.4.2. 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.

3.4.3. 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, authentication and capability descriptors, confidence and freshness metadata, health and compatibility information, versioning data, and routing metadata.

3.4.4. 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.

3.5. The Three-Path Execution Model

A healthy system gives users three choices for any task:

- **Path 1 — Local cache:** If the user already

has a fresh route cached locally, it is used directly at zero cost.

- **Path 2 — Shared graph:** If a matching skill exists, the agent pays to install it locally, receiving the skill package (endpoint documentation, schemas, auth profile, TypeScript client) and executing it on its own infrastructure with its own credentials — a single API call rather than a full browser session.
- **Path 3 — Browser rediscovery:** If no suitable route exists or the user prefers not to pay, the site is browsed, the route is rediscovered, and the result is optionally contributed back to the graph.

This structure is essential to the system's integrity. The shared graph is not a monopoly gatekeeper over the web — users can always bypass it. The graph only wins if it creates genuine surplus.

3.6. Composite Scoring and Intent Resolution

When an agent requests a task, the orchestrator follows a priority chain. First, it checks the *route cache* (5-minute TTL) 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. 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.

3.7. 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 after consecutive execution failures 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, executing safe GET endpoints to detect failures and schema drift. Skills with three or more consecutive failures are automatically deprecated.

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 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 3–5 seconds per raw page load (or 10–45 seconds for a complete agent action including LLM reasoning [7]), 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 failure probability (approximately 20% for browser automation [23]), 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.

Table 2 applies this condition across site categories using empirically observed parameters from the benchmark (Section 7). The rediscovery cost $c_{\text{rediscovery}}$ is computed per Equation 2 using category-specific median latencies, estimated LLM token costs, and observed failure rates. The "max viable fee" column shows the ceiling below which rational adoption holds.

Table 2: Sensitivity of the adoption condition across site categories. Max viable fee is the empirical $c_{\text{rediscovery}}$ ceiling per Equation 2; current fee is the median marketplace price.

Category	Latency	p_{fail}	Max fee	Cur. fee	Margin
Light API	2,100 ms	.05	\$0.10	\$0.005	20×
Medium JS	3,400 ms	.15	\$0.18	\$0.01	18×
Heavy SPA	4,800 ms	.25	\$0.35	\$0.015	23×
WAF-prot.	5,200 ms	.35	\$0.53	\$0.02	27×

The headroom column reveals that current marketplace fees are 18–27× below the rational defection threshold across all categories. This margin is intentional: aggressive underpricing relative to the ceiling accelerates adoption and graph growth during the bootstrapping phase. The adoption condition holds with substantial margin even under pessimistic assumptions about LLM token costs (doubled) or failure rates (halved).

4.2. What Users Pay For

Users 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, users 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.

4.3. Client-Side Routing Intelligence

The client estimates the economic tradeoff in real time by tracking estimated DOM tokens for rediscovery, estimated LLM tokens, estimated browser runtime cost, expected failure probability, shared route fee, and route confidence. The routing policy follows directly from Equation 1: use the local cache if available and fresh; use the shared graph if $f_{\text{route}} < c_{\text{rediscovery}}$; otherwise, rediscover through the browser.

4.4. 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.

The waste eliminated by a shared commons is not hypothetical. Every agent that opens a website for the first time pays the same discovery cost: page load, DOM parsing, LLM reasoning, request inference, retry on failure. Across thousands of agents interacting with the same popular sites, this produces an enormous quantity of duplicated work — a *redundant discovery tax* levied on every participant. The shared route graph eliminates this tax by converting the first agent's discovery into infrastructure that all subsequent agents consume. Delta-based attri-

bution ensures that this conversion is not charity but investment: contributors earn in proportion to the marginal value they add, transforming what was previously economic waste — redundant computation discarded after each session — into shared, versioned, economically rewarded capital. The system does not merely reduce cost; it changes the category of the expenditure from consumption to capitalisation.

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.

5.1. x402 Payment Flow

Marketplace transactions follow the x402 protocol [20], a recently proposed standard (2025) with limited production deployment to date. When an agent pays to access a route, it is installing a skill — a structured capability package containing endpoint documentation, parameter schemas, auth profiles, and executable client code. The agent then executes the skill locally using its own credentials; 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 route lookup — without intermediaries. The current implementation uses USDC on Solana for settlement, though the protocol is network-agnostic. We note that the payment architecture is functional but has not been stress-tested at scale; its viability depends on broader x402 ecosystem adoption.

5.2. The Route Economy

Every route in the shared graph is a revenue-generating asset. When a user pays to access a route, the fee is automatically split via x402 among all parties who contributed to its existence and maintenance.

5.2.1. Fee Split Architecture

Each route usage 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 usage 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.

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 (currently computed via OpenAI `text-embedding-3-small`; the specific model is a replaceable implementation detail). The contributor share C is distributed among all historical contributors in proportion to their cumulative delta scores.

The attribution mechanism faces three principal attack vectors. First, *delta inflation*: a contributor submits many small redundant commits to accumulate attribution share without adding real value. The current mitigation is a minimum-delta threshold (both in line-level diff size and embedding dissimilarity) below which commits receive zero attribution; additionally, deltas that do not improve the route's execution success rate on the next verification cycle are retroactively discounted. Second, *Sybil fragmentation*: an attacker creates multiple identities to claim credit for a single logical improvement split across many commits. The system requires cryptographically signed contributor identities tied to x402 payment addresses; creating a new identity incurs economic cost (the contributor must first pay for route lookups to establish a payment history), and attribution is weighted by the contributor's historical success rate, disadvantaging new accounts. Third, *contributor collusion*: two or more contributors coordinate to alternately "improve" and "revert" a route, generating mutual attribution. The continuous trust model (Section 6.3) detects oscillating schema changes and flags routes with high-frequency edits for manual review; contributors whose deltas are repeatedly reverted see their attribution share decay. These mitigations are defence-in-depth, not provably secure. Formal incentive compatibility analysis — including whether the mechanism is strategyproof under rational contributors — remains future work.

5.2.3. Site Owner Compensation

A critical design goal is that website owners can opt in to the shared graph and receive fair compensation for programmatic access to their services. In the opt-in model, a site owner registers their domain with the marketplace, and agent traffic routed through the shared graph generates micropayments to the site owner. This transforms the relationship 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.

5.2.4. Websites as Usage-Priced Endpoints

The opt-in model positions websites as usage-priced endpoints. This is particularly relevant for agent traffic, which is bursty, unpredictable, and bypasses traditional session-based monetisation. The agent always executes locally with its own credentials; the marketplace distributes knowledge (endpoint schemas, auth patterns), not access. Section 8.5 examines the broader implications of this shift from subscription to per-use pricing.

5.3. Dynamic Route Pricing

Routes are priced dynamically based on estimated rediscovery cost saved, route confidence and freshness score, current demand, and historical success rate. The effective fee is always bounded above by $c_{\text{rediscovery}}$ from Equation 2 — if the shared route is more expensive than 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 selects a diverse subset of GET endpoints, substitutes template parameters with captured examples, executes real HTTP requests, and requires a minimum 50% success rate with at least one verified endpoint.

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. Cloud infrastructure is used exclusively for route verification: validating that a route is functional, that its response shapes match documented schemas, and that its health score is accurate. The current implementation runs verification in E2B sandboxes; future work includes adding TEE attestation to provide cryptographic proof that a route was tested against live endpoints without tampering.

6.3. Continuous Trust Model

Rather than rigid quality tiers, route trust follows a continuous model inspired by ERC-8004 proportional trust [21]. Each route carries a trust profile composed of three signals.

The first signal is **execution feedback**. Agents that use a route submit signed feedback — a numeric score (0–100) with optional tags such as "auth_worked," "schema_match," or "timeout." Feedback is cryptographically signed to prevent Sybil manipulation, and aggregated feedback produces a rolling trust score visible to all consumers.

The second signal is **validator attestation**. Independent validators periodically test routes against live endpoints in sandboxed environments. Validator responses include a 0–100 confidence score and evidence such as response hashes and schema diffs. Validators can stake to back their attestations, and false attestations are slashable.

The third signal is **freshness decay**. Trust decays over time without fresh validation or positive execution feedback. 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.

The combined trust score determines route visibility and pricing in the marketplace. Low-trust routes are served with warnings and lower fees, while high-trust routes command premium pricing. Routes that fall below a minimum trust threshold after sufficient execution history are delisted.

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. At 464 KB and 3 ms cold start, Kuri is three orders of magnitude smaller than Playwright (80–150 MB) and starts in a fraction of the time (3 ms vs 1–3 s). 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 client for skill publishing and semantic search, a verification system for periodic endpoint re-testing,

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

and an execution engine with automatic auth refresh and retry logic.

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.

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 3 shows representative task examples.

Table 3: 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 4: 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 4 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 cumulative distribution of speedup ratios. The distribution is right-skewed: the majority of domains cluster between 2–8× speedup, with a long tail of 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).

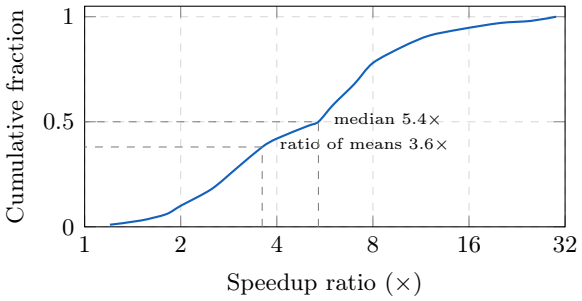

Figure 2: CDF of speedup ratios (Playwright latency / Unbrowse latency) across 94 domains. Log-scale x-axis. The gap between mean (3.6×) and median (5.4×) reflects right skew from domains with particularly fast API responses. Data points are smoothed from per-domain measurements; raw data available in 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.

Cold-start latency (Table 5) 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 acknowledge that the cold-start sample ($n = 20$) is small and limits the statistical power of these estimates; confidence intervals on the median and 90th percentile are wide. An expanded cold-start evaluation ($n \geq 100$) across a stratified sample of site categories, including sites with heavy bot detection, is in progress and will be reported in a subsequent version of this paper.

Table 5: 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)

7.5. Cost Analysis

Table 6 compares estimated per-task costs across execution paths.

Table 6: Estimated per-task cost comparison.[3]

Component	**Browser Rediscovery**	**Shared Graph Lookup**
Browser runtime	\$0.02–0.05	\$0.00
LLM reasoning tokens	\$0.04–0.35	\$0.00
Retries/failures	\$0.04–0.13	\$0.00
Shared route fee	\$0.00	\$0.005–0.02
Total estimated	\$0.10–0.53	\$0.005–0.02

The shared graph achieves 90–96% cost reduction per task for routes that are already indexed,

[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 frequently.

providing empirical grounding for 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 users generate more traffic, which discovers more routes, which attracts more agents. However, we have not yet measured the causal relationship between user 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.

7.7. Implications for the Agent Stack

The speedup and cost figures suggest a reclassification of browser automation in the agent stack: from default execution mode to fallback, invoked only when no callable route exists. This trajectory is corroborated by the Permission Manifests specification [36], which explicitly encourages agents to "prefer API calls when available." An agent that can resolve an intent in 630 ms for $0.01 will not voluntarily spend 3,400 ms and $0.10–0.53 through a browser.

The same reclassification applies to per-service integration abstractions. Each MCP server or bespoke API wrapper represents manual effort to expose a single service; the shared route graph replaces this with a collectively discovered registry where coverage scales with usage, not engineering headcount. The scarce resource shifts from the ability to reach a website to the ability to reach it *efficiently*—knowing which endpoint to call, with which parameters, authenticated how. This knowledge, maintained and priced in the shared graph, becomes the locus of value.

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 (Section 6.1) executes candidate routes in sandboxed environments and verifies response schemas against declared types; the continuous trust model (Section 6.3) aggregates signed execution feedback, so a malicious route's trust score degrades rapidly after negative reports; 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 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 5.2.3, 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 respects `robots.txt` directives and does not circumvent authentication barriers.

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.

8.4. Limitations

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 4 represent warmed-cache performance under controlled conditions; real-world gains vary with anti-bot middleware, token expiration, and regional latency. Cold-start performance (Table 5) 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. Simulation or deployment-scale data on x402 payment flows, contributor attribution incentive compatibility, and Sybil resistance remain areas for future work.

8.5. Architectural Implications

The shared route graph instantiates an architectural pattern with consequences beyond the system itself. We trace two that follow directly from the design.

8.5.1. Interface and Pricing Inversion

The dominant SaaS model exposes a GUI for humans, guards its shadow API behind authentication gates, and charges a flat subscription. Agent traffic is a poor fit for this model: it is bursty, parallelisable, and volume-variant by orders of magnitude. An agent chaining 200 API calls on Monday and zero on Tuesday has no natural home in per-seat-per-month pricing.

The shared route graph inverts both sides. First, an *interface inversion*: when internal endpoints are passively discovered and priced per use, the API surface becomes the primary machine-native interface and the GUI narrows to a rendering layer for the remaining human interactions. Second, a *pricing inversion*: subscription gives way to usage-priced access, extending the per-unit pricing already visible in infrastructure (Stripe per transaction, OpenAI per token) [34] to the broader web surface.

This inversion also offers a resolution to the adversarial equilibrium between sites and automated agents [33, 35]. Sites deploy bot-detection; agents deploy circumvention; both burn resources unproductively. The opt-in compensation model (Section 5.2.3) replaces this arms race with a market: sites receive micropayments for API-level access that is cheaper to serve than rendered pages, while agents gain reliable access at lower cost. The result is a Pareto improvement that becomes feasible once pricing shifts from subscription to per-use.

8.5.2. From Page Forest to Call Graph

The web today is a forest of silos, each requiring separate navigation, authentication, and DOM interpretation. The shared route graph collapses this surface into a collectively maintained call index: a unified addressing layer over the web's callable interfaces, growing as a side effect of normal agent activity. The distance between an agent's intent and its fulfilment shrinks from a multi-step browser session to a single parameterised call.

8.5.3. Design Constraints as Architecture

Several design choices were deliberately rejected: centralised cloud execution (credential leakage risk), subscription pricing (decouples payment from value), a central crawler (speculative rather than demand-driven growth), and proprietary protocols (lock-in). Each constraint closes a failure mode and reinforces the system's core properties—local execution, usage pricing, and open contribution.

8.6. 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.
4. **Comparative evaluation:** Formal comparison with existing API aggregation platforms (RapidAPI, Postman) and per-service integration approaches (MCP servers) on coverage, latency, and cost metrics.
5. **Write-operation benchmarks:** Extending evaluation beyond information-retrieval to multi-step authenticated workflows including form submissions, checkout flows, and state-mutating operations.

9. Conclusion

The proliferation of agent-to-web abstractions—MCP servers, browser automation frameworks, bespoke API wrappers—reflects a shared assumption: that the web's callable surface is inaccessible and must be mediated by heavyweight tooling. We have argued, and provided empirical evidence, that this assumption is wrong. The callable surface already exists as shadow APIs behind every website. What was missing was not another abstraction layer but a mechanism to discover, share, and price these interfaces collectively.

Unbrowse demonstrates that this mechanism is viable. Across 94 domains, cached route execution achieved 3.6× mean speedup (5.4× median) over Playwright with 90–96% cost reduction, and well-cached routes completed in under 100 ms. Cold-start discovery averages 12.4 s but amortises within 3–5 reuses. The system's market discipline—users can always defect to browser rediscovery—distinguishes it from speculative protocol designs and ensures contributors earn only when their routes deliver genuine value.

The deeper contribution is architectural. Rather than wrapping each website in a bespoke integration, the shared route graph treats the web's existing shadow APIs as a collectively maintained, usage-priced call graph. Browser automation, MCP adapters, and official API clients remain available as fallbacks, but the evidence suggests they are the exception rather than the rule: for the majority of information-retrieval tasks, a single cached API call suffices. Shadow APIs, it turns out, were always there. They just needed a reason to be found and a price to keep them maintained.

Formal economic analysis, incentive compatibility proofs, and deployment-scale validation of x402 payment flows remain as future work.

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. In *ACM Web Conference*, pp. 2222–2237, 2025.

[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.

[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. 938 F.3d 985 (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. Amoruso et al. The dead internet theory: A survey on artificial interactions and the future of social media. *arXiv:2502.00007*, 2025.

[34] R. Fernandez et al. The agentic economy. *arXiv:2505.15799*, 2025.

[35] H. Schütze et al. Scrapers selectively respect robots.txt directives: Evidence from a large-scale empirical study. *arXiv:2505.21733*, 2025.

[36] Lightweight Agent Standards Working Group. Permission manifests for web agents. *arXiv:2601.02371*, 2026.

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.