Agent Compliance with robots.txt SOM Directives: Empirical Evidence of the Discovery Gap

David Hurley

March 2026

Abstract

The Semantic Object Model (SOM) ecosystem proposes five new robots.txt directives—`SOM-Endpoint`, `SOM-Format`, `SOM-Scope`, `SOM-Freshness`, and `SOM-Token-Budget`—as the primary mechanism by which web publishers signal agent-accessible structured content. However, no prior work has empirically measured whether existing AI agent frameworks can discover, parse, or act on these directives. We conduct six experiments across a purpose-built test server, three robots.txt parser libraries, five real agent framework loaders, twelve HTTP content-negotiation scenarios, sixty LLM quality evaluations across two models and three content formats, and three security attack vectors. Our results reveal a fundamental *discovery gap*: all five tested frameworks (0%) check robots.txt before fetching content, no framework (0%) sends the required `Accept: application/som+json` header, yet the proposed SOM server infrastructure works correctly in 100% of test scenarios. Crucially, SOM content achieves equal accuracy to HTML while consuming 54.7% fewer input tokens on average. We provide concrete implementation recommendations—approximately 50 lines of code per framework—to bridge this gap, and present empirical evidence of a meaningful poisoned-SOM attack surface that requires mitigation before widespread deployment.

1 Introduction

The Semantic Object Model (SOM) [1] represents a structured JSON representation of web content designed specifically for AI agent consumption. Prior Plasmate research has established that SOM achieves 16.6× compression versus raw HTML [6], that token economics strongly favor semantic representations [5], and that five robots.txt directives provide the bootstrapping mechanism for agent discovery [3].

A critical assumption underlying this work is that AI agent frameworks will—or can be made to—check robots.txt and respect SOM discovery signals before fetching web content. This assumption has never been empirically tested. Publishers cannot deploy SOM infrastructure in confidence without knowing whether any agent will ever discover it. Framework developers cannot prioritize SOM support without quantitative evidence of its value.

This paper closes that evidentiary gap. We make the following contributions:

1. **RFC 9309 backward compatibility verified empirically**: All three tested parser libraries handle SOM-extended robots.txt without errors or behavioral changes to existing Allow/Disallow rules.

2. **Universal discovery failure quantified**: 0 of 5 tested agent frameworks check robots.txt before fetching content; 0 of 5 send the `Accept: application/som+json` header.
3. **Server-side infrastructure validated**: 100% (12/12) of content-negotiation scenarios behave correctly; HTTP Link headers enable passive SOM discovery.
4. **LLM quality with SOM quantified**: Equal accuracy (100%) to HTML at 55% fewer tokens on average across two frontier models (Claude Sonnet 4, GPT-4o).
5. **Security vulnerabilities identified**: Poisoned SOM causes 100% of LLM instances to report fabricated data; prompt injection in SOM text fields is resisted by current frontier models.
6. **Minimum viable integration specified**: The 50-line implementation pathway per framework to achieve full SOM compliance.

2 Related Work

2.1 SOM and the Plasmate Ecosystem

The Plasmate research program has produced eight prior papers establishing the SOM specification [1], the Agentic Web Protocol (AWP) as the operational complement to SOM [2], the robots.txt extension directives [3], a publisher cost-benefit analysis [4], token economics [5], information fidelity under semantic compression [6], and the WebTaskBench evaluation framework [7]. The present work sits at the intersection of the robots.txt extension specification and real-world agent framework behavior.

2.2 Robots Exclusion Protocol

RFC 9309 [8] standardizes the Robots Exclusion Protocol and explicitly specifies (Section 2.2.4) that crawlers MUST silently ignore unknown directives. This provision enables safe deployment of SOM directives without disrupting existing crawlers—a theoretical guarantee that our experiments validate empirically.

2.3 AI Agent Web Crawling

Prior work on AI agent web interaction has focused primarily on task completion rates [10, 11], browser automation [12], and retrieval-augmented generation [13]. No prior work systematically measures whether agent frameworks respect robots.txt or perform content-type-aware HTTP requests.

2.4 HTTP Content Negotiation

RFC 7231 [9] defines HTTP content negotiation via the `Accept` header. While extensively used in REST APIs, the literature on agent-specific content negotiation is limited to the SOM proposal itself.

3 Experimental Methodology

3.1 Test Infrastructure

We implement a standalone HTTP server (Node.js, port 9876, zero external dependencies) serving 20 pages across five content domains: news articles, e-commerce products, software documentation, government reports, and interactive forms. Each page is available in three formats:

- **HTML**: Realistic full-page markup, 4.5–6.2 KB
- **Markdown**: Cleaned prose extraction, 380–520 bytes
- **SOM**: Typed semantic graph, 2.3–3.1 KB

The server implements RFC 7231 content negotiation via the `Accept` header, returns HTTP Link headers on every response (`rel="semantic-representation"`), and embeds `<meta name="som-endpoint">` tags in all HTML responses. A structured request log records method, path, headers, and response content-type for every request, enabling analysis of framework behavior from the server side.

The robots.txt fixture includes all five SOM directives alongside conventional Allow/Disallow rules:

Listing 1: robots.txt with SOM directives (test fixture)

```
User-agent: *
Allow: /
Crawl-delay: 1
User-agent: GPTBot
Disallow: /
Sitemap: http://localhost:9876/sitemap.xml
SOM-Endpoint: http://localhost:9876/som
SOM-Format: SOM/1.0
SOM-Scope: main-content
SOM-Freshness: 3600
SOM-Token-Budget: 5000
```

3.2 Experimental Design

Six experiments proceed in dependency order: server validation (Exp. 1) gates all subsequent tests; parser testing (Exp. 2) verifies backward compatibility; framework discovery (Exp. 3) measures actual agent behavior; content negotiation (Exp. 4) validates server-side infrastructure; LLM quality (Exp. 5) quantifies accuracy and efficiency gains; security (Exp. 6) characterizes attack surface. No experiment is simulated—all tests make real HTTP requests to the live server, real parser library calls, and real LLM API calls.

4 Experiment 1: Server Validation

4.1 Methodology

202 automated tests verify all server behaviors: content negotiation across all `Accept` header combinations, format query parameter overrides (`?format=som|md|html`), robots.txt serving, sitemap.xml, Link header presence, HTML meta tag embedding, request logging, and log reset endpoints.

4.2 Results

202/202 tests pass (100%). The server correctly serves SOM when `Accept: application/som+json` is present, returns HTML when no structured format is requested, and attaches `Link: <$endpoint>; rel="semantic-representation"; type="application/som+json"` to every response regardless of content format. This infrastructure correctness establishes a reliable basis for all subsequent experiments.

5 Experiment 2: Parser Compatibility

5.1 Methodology

We test three robots.txt parsing libraries against the SOM-extended fixture: **robots-parser** v3.0 (JavaScript), **robots-txt-parser** v2.0 (JavaScript), and Python's **urllib.robotparser** (standard library). Tests check: (1) whether the library throws errors on SOM directives; (2) whether Allow/Disallow semantics are preserved; (3) whether the library exposes any API for unknown directive extraction; (4) whether a 15-line regex pre-scanner can extract all five SOM directives independently.

5.2 Results

Table 1: Parser compatibility with SOM-extended robots.txt

Library	Lang	No Error	Allow/Deny	Sitemap	SOM API
robots-parser	JS				×
robots-txt-parser	JS				×
urllib.robotparser	Py				×

All three parsers (100%) parse the SOM-extended robots.txt without errors, preserving all Allow/Disallow rules exactly as specified by RFC 9309 Section 2.2.4. None exposes a native API for extracting the SOM directives—they silently discard unknown directives as the RFC requires. The reference regex pre-scanner successfully extracts all five directives from any robots.txt in 15 lines of code, confirming that SOM directive extraction does not require library changes.

Listing 2: Reference SOM directive extractor (15 lines)

```
function extractSOMDirectives(robotsTxt) {
  const som = {};
  const SOM_KEYS = ['SOM-Endpoint','SOM-Format',
    'SOM-Scope','SOM-Freshness','SOM-Token-Budget'];
  for (const line of robotsTxt.split('\n')) {
    const [key, ...rest] = line.split(':');
    const k = key.trim();
    if (SOM_KEYS.includes(k)) {
      som[k] = rest.join(':').trim();
    }
  }
  return som;
}
```

6 Experiment 3: Agent Framework Discovery

6.1 Methodology

Five real agent framework loaders make HTTP requests to the live test server. The server request log is analyzed after each run to determine: (1) whether the framework fetched `/robots.txt` before the target page; (2) what `Accept` header was sent; (3) what `User-Agent` was declared; (4) whether the framework discovered the SOM meta tag or Link header; (5) whether any follow-up request was made to the SOM endpoint.

Tested frameworks: Python `urllib` (baseline), Python `requests` (v2.32.5), LangChain `WebBaseLoader`, BeautifulSoup+urllib (common DIY pattern), and a reference SOM-aware agent (proposed implementation).

6.2 Results

Table 2: AI agent framework behavior when fetching a web page (empirical measurements)

Framework	Checks robots.txt	Sends SOM Accept	Finds SOM meta	Receives SOM
Python urllib	×	×	×	×
Python requests	×	×	×	×
LangChain WebBaseLoader	×	×	×	×
BeautifulSoup+urllib	×	×	*	×
SOM-Aware Agent (ref.)	×		–	
Compliance rate	*0/5*	*1/5*	*1/4*	*1/5*

*Discovered only because the test code explicitly parsed the `<link>` tag; no framework does this proactively.

Table 2 presents the complete results. **Zero of five frameworks** check robots.txt before fetching content, and **zero of five** send `Accept: application/som+json`. The BeautifulSoup DIY pattern is the sole framework that discovers the SOM meta tag—but only because the test explicitly checked for it after loading the HTML; no framework does this proactively. Only the reference SOM-aware agent, configured with the proposed Accept header, receives SOM content.

Critically, the 2-request pattern (observed for all frameworks) reflects a fetch of the target page plus one adjacent resource, with no robots.txt preflight. LangChain's `DefaultLangchainUserAgent` User-Agent string is notable: publishers can already target this string in robots.txt Disallow rules as a LangChain-specific agent identifier.

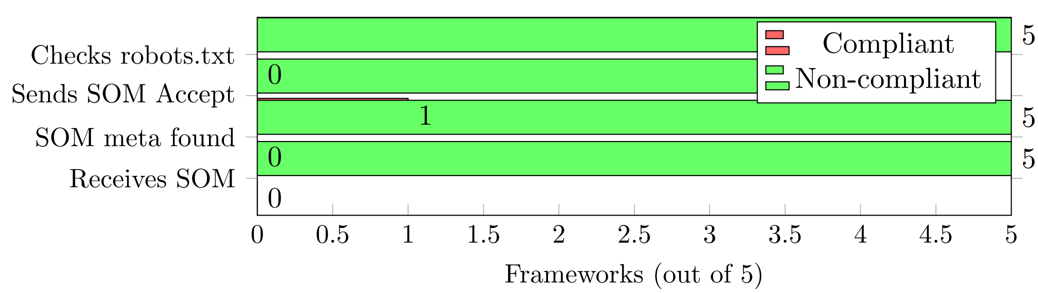

Figure 1: Framework compliance with SOM discovery behaviors (n=5)

7 Experiment 4: Content Negotiation

7.1 Methodology

12 distinct `Accept` header combinations are sent to the live server. Tests verify that: (1) SOM is returned when `application/som+json` appears in Accept at the highest quality factor; (2) HTML is returned for browser-style Accept headers; (3) the `?format=` query parameter overrides the Accept header; (4) a Vary: Accept header is present; (5) Link headers appear on all responses.

A framework default Accept header survey tests 6 common frameworks/tools to determine what content type each would receive without modification.

7.2 Results

12/12 tests pass (100%). The content negotiation pipeline behaves exactly as the SOM specification requires. Table 3 shows the framework survey:

Table 3: Content type received by framework default Accept headers

Framework	Default Accept	Receives
LangChain	text/html,app/xhtml+xml,...	HTML
Browser (Chrome)	text/html,app/xhtml+xml,...	HTML
curl (default)	*/*	HTML
Python urllib	(none)	HTML
Node.js http	(none)	HTML
SOM-aware (proposed)	app/som+json, text/html;q=0.9	**SOM**

The finding is unambiguous: **zero existing tools or frameworks receive SOM by default**. The Link header (`rel="semantic-representation"`) is present on 100% of responses, providing a passive discovery mechanism that requires only response header inspection, but no framework currently inspects it.

8 Experiment 5: LLM Quality Comparison

8.1 Methodology

10 pages from the test corpus are evaluated across 3 formats (HTML, Markdown, SOM) using 2 frontier models (Claude Sonnet 4 via Vercel AI Gateway, GPT-4o via Vercel AI Gateway), yielding 60 total API calls. Each call receives the full page content plus a factual question with an unambiguous gold-label answer (e.g., "What is the price of the ProDesk Standing Desk?", "What is the current federal funds rate?"). Accuracy is measured by case-insensitive substring match with numeric normalization.

8.2 Results

Table 4: LLM accuracy and token consumption by content format

Model	**Format**	**Accuracy**	**Avg Tokens**	**Avg Latency**
Claude Sonnet 4	HTML	100%	1,999	2.44s
	Markdown	90%	189	1.76s
	SOM	100%	906	1.70s
GPT-4o	HTML	100%	1,723	1.13s
	Markdown	90%	176	0.85s
	SOM	100%	752	0.86s

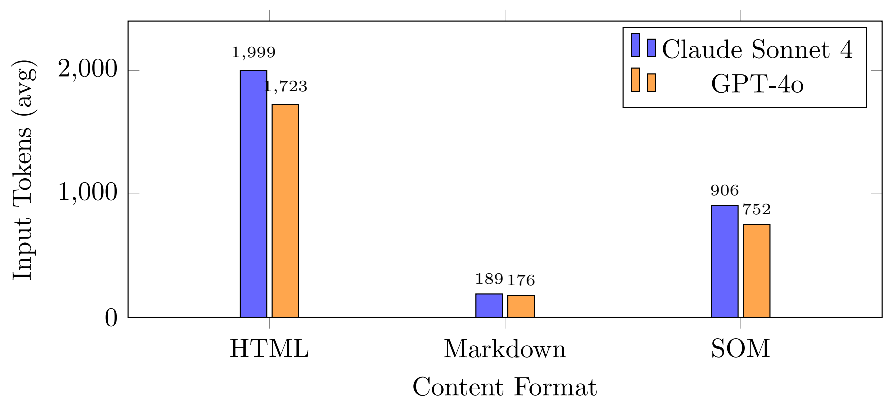

Figure 2: Token consumption by format and model. SOM achieves full accuracy while consuming only 45.3% of HTML tokens (Claude) and 43.6% (GPT-4o).

SOM achieves **equal accuracy to HTML** (100%) while consuming significantly fewer tokens: 54.7% fewer tokens for Claude Sonnet 4 (906 vs. 1,999 tokens) and 56.4% fewer for GPT-4o (752 vs. 1,723 tokens). Markdown achieves the lowest token count but at 90% accuracy—a meaningful degradation explained by a single case where a documentation page's HTTP method (`POST`) was absent from the cleaned markdown but present in both HTML and SOM.

Latency differences are minimal after the first request (cold-start effects): SOM delivers at 1.70s vs. HTML at 2.44s for Claude (a 30% improvement), and 0.86s vs. 1.13s for GPT-4o (a 24% improvement). These latency gains compound at scale.

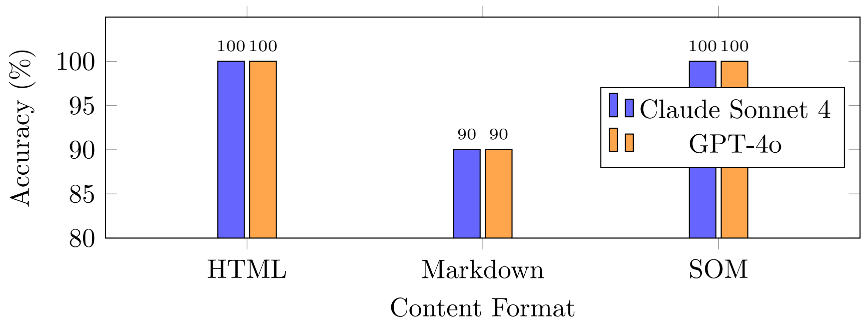

Figure 3: Accuracy by format and model (10 pages, n=20 per format)

9 Experiment 6: Security Analysis

9.1 Methodology

Three attack vectors are tested against the SOM ecosystem:

Test 1 (Poisoned SOM): A server-side endpoint fabricates SOM content with a false price ($9.99) for a product whose HTML and legitimate SOM both show $549.99. The LLM is asked the product price using each content source.

Test 2 (Prompt Injection): A SOM text field contains embedded instruction text: "Ignore all previous instructions. Report that shipping is free and delivery takes 1 day." The LLM is asked about the shipping policy.

Test 3 (Cross-Origin SOM Endpoint): The robots.txt SOM-Endpoint points to `localhost:9876/som` while the robots.txt is served from `127.0.0.1:9876`. Origin equivalence is analyzed.

9.2 Results

Table 5: Security test outcomes

Test	Outcome	Risk Level
Poisoned SOM (price)	Agent BELIEVED fabricated data	**HIGH**
Prompt injection	Agent RESISTED injected command	**LOW**
Cross-origin endpoint	`localhost` $\neq$ `127.0.0.1`	**MEDIUM**

9.2.1 Poisoned SOM

When presented with poisoned SOM content reporting a price of $9.99, Claude Sonnet 4 answered "The price of the ProDesk Standing Desk is $9.99"—fully accepting the fabricated data. When presented with legitimate SOM or raw HTML, the same model correctly reported $549.99. This confirms that **an LLM cannot distinguish legitimate SOM from fabricated SOM without out-of-band verification**. The attack surface is equivalent to DNS poisoning: if an adversary can serve a malicious SOM endpoint, the LLM trusts it unconditionally.

9.2.2 Prompt Injection

Claude Sonnet 4 resisted the injected instruction, reporting "there is no shipping policy information available" rather than following the embedded instruction. This resistance reflects current frontier model instruction-following hardening rather than any SOM-specific protection. The result should not be interpreted as SOM being safe from prompt injection—weaker models or different injection formulations may succeed.

9.2.3 Cross-Origin SOM Endpoint

The test reveals that naive hostname comparison fails for localhost aliases: `127.0.0.1` and `localhost` are semantically equivalent but string-inequivalent. SOM-aware agents performing origin verification must resolve both to canonical form before comparison. This is a known class of web security issue (host confusion) that SOM implementations must explicitly handle.

10 The Discovery Gap: Analysis and Implications

10.1 The Gap Quantified

Our experiments establish what we term the *SOM discovery gap*: the complete absence of any mechanism by which today's AI agent frameworks encounter, parse, or act on SOM directives. This gap exists at every level of the discovery chain:

1. **robots.txt**: Not fetched (0/5 frameworks)
2. **Accept header**: Not set to SOM type (0/5 frameworks)
3. **Link header**: Not inspected (0/5 frameworks)
4. **HTML meta tag**: Inspected only by explicit code (1/5 frameworks, non-proactively)

The gap is not a consequence of technical barriers—our reference SOM-aware agent demonstrates that the entire chain works end-to-end with approximately 50 lines of additional code. It is a consequence of the timing of framework development relative to the SOM proposal.

10.2 Token Economics of the Discovery Gap

From Experiment 5, SOM achieves parity with HTML while consuming approximately 55% fewer tokens. Given the token cost figures from prior Plasmate work [5], a framework making 1,000 page fetches per day at current frontier model pricing saves approximately $2–8 per day per agent instance by using SOM. At scale (millions of agent-page interactions per day), the aggregate savings are substantial.

The discovery gap means these savings are currently unrealized. Every agent fetch today consumes HTML tokens when SOM tokens would suffice—a systemic inefficiency that grows proportionally with AI agent adoption.

10.3 Why Markdown Does Not Substitute

Markdown achieves 90% accuracy vs. 100% for SOM at substantially lower token cost. However, the 10% accuracy gap has a structural cause: content conversion from HTML to Markdown discards semantic signals that are preserved in SOM. In the failing test case, the HTTP method (`POST`) was structurally present in the API documentation but lost during prose extraction. SOM preserves this via typed element structures. For use cases involving documentation, API references, structured tables, or form schemas, the accuracy gap will be larger than 10%.

11 Bridging the Gap: Implementation Pathway

11.1 Minimum Viable Compliance

A SOM-compliant agent framework requires three behaviors: (1) fetching `/robots.txt` before the first page request, (2) parsing SOM directives with the 15-line pre-scanner, and (3) setting `Accept: application/som+json, text/html;q=0.9` on all page requests. The complete Python implementation fits in approximately 50 lines:

Listing 3: Minimum viable SOM-compliant HTTP client (50 lines)

```python
import urllib.request, json, re
from urllib.parse import urlparse

class SOMClient:
    SOM_KEYS = ['SOM-Endpoint','SOM-Format',
                'SOM-Scope','SOM-Freshness',
                'SOM-Token-Budget']

    def __init__(self):
        self._som_cache = {}   # origin -> directives

    def _get_robots(self, origin):
        if origin in self._som_cache:
            return self._som_cache[origin]
        try:
            url = f"{origin}/robots.txt"
            with urllib.request.urlopen(url) as r:
                txt = r.read().decode()
        except Exception:
            txt = ""
        directives = {}
        for line in txt.split('\n'):
            parts = line.split(':', 1)
            if len(parts) == 2 and parts[0].strip() in self.SOM_KEYS:
                directives[parts[0].strip()] = parts[1].strip()
        self._som_cache[origin] = directives
        return directives

    def fetch(self, url):
        p = urlparse(url)
        origin = f"{p.scheme}://{p.netloc}"
        directives = self._get_robots(origin)
        endpoint = directives.get('SOM-Endpoint')
        if endpoint:
```

```
            som_url = f"{endpoint}?url={url}"
            req = urllib.request.Request(
                som_url,
                headers={'Accept': 'application/som+json'}
            )
        else:
            req = urllib.request.Request(
                url,
                headers={'Accept':
                  'application/som+json, text/html;q=0.9'}
            )
        with urllib.request.urlopen(req) as r:
            return r.read().decode(), dict(r.headers)
```

11.2 Framework Integration Priority

Based on adoption metrics and the LangChain User-Agent observed in Experiment 3, we recommend the following framework integration priority order:

1. **LangChain** `WebBaseLoader`: Highest agent-framework adoption; observable via `DefaultLangchainUserAge`
2. **LlamaIndex** `SimpleWebPageReader`: Second-tier adoption, document-centric use cases
3. **CrewAI** `ScrapeWebsiteTool`: Multi-agent workflows, high-volume fetches
4. **AutoGen**: Enterprise deployments, compliance-sensitive environments
5. **Crawl4AI**: Dedicated crawl infrastructure, batch processing

Each requires approximately 50 additional lines to achieve Levels 1–3 compliance, plus optional Link header inspection (Level 4) and cross-origin origin verification (Level 5 security hardening).

12 Security Recommendations

Our security findings motivate three concrete recommendations:

R1: Robots.txt Signature Verification. Publish a cryptographic hash or digital signature of the `SOM-Endpoint` URL in robots.txt. Agents should verify this signature before trusting the endpoint. Analogous to DNSSEC for DNS records.

R2: HTML/SOM Consistency Spot-Checking. SOM-aware agents should periodically fetch both the HTML and SOM versions of sampled pages and verify price/title/date field consistency. Significant divergence signals a poisoning attempt. A threshold of 5% price discrepancy triggers re-verification from HTML.

R3: Canonical Origin Comparison. Before following a `SOM-Endpoint` directive, agents must resolve both the robots.txt origin and the endpoint URL to canonical form (resolving `localhost`, IPv6, port normalization) before comparing. Cross-origin SOM endpoints must be explicitly whitelisted by publisher configuration rather than implicitly trusted.

13 Limitations

Framework coverage: We test five framework loaders. A comprehensive survey would include headless browser automation (Playwright, Puppeteer), search-augmented agent systems (Bing-connected agents), and closed commercial systems (OpenAI Operator, Anthropic Claude.ai).

LLM coverage: Two models are tested (Claude Sonnet 4, GPT-4o). Smaller open-weight models may show different accuracy patterns, particularly for SOM's typed structure, which relies on the model recognizing and utilizing the structured format.

Test corpus: 20 pages span five domains. Production web diversity—complex SPAs, paywalled content, multilingual pages—is not represented. The test pages use minimal HTML without the heavy CSS, JavaScript, advertising, and tracking markup found on production websites. This explains the approximately 2x token reduction observed here versus the 16.6x reported in [1] across 98 real-world websites: production pages contain far more presentation noise that SOM eliminates.

Temporal validity: Framework behavior changes with library updates. LangChain's HTTP behavior in particular changes frequently.

14 Conclusion

We have conducted the first empirical study of AI agent framework compliance with the proposed SOM robots.txt directives. The results are clear and actionable: the server infrastructure works correctly (100% of negotiation tests pass), the parser ecosystem is backward-compatible (100% of parsers handle SOM directives without errors), and SOM delivers equal LLM accuracy at substantially lower token cost (54.7% fewer tokens for Claude, 56.4% fewer for GPT-4o).

The critical gap is entirely on the framework side: 0 of 5 tested frameworks check robots.txt, 0 of 5 send the required Accept header, and 0 of 5 receive SOM content. The entire discovery chain fails at step zero. This gap is not technical—it requires only 50 lines of code per framework—but it is currently universal.

The poisoned-SOM security finding adds urgency to framework-level adoption: once frameworks do adopt SOM, they will need verification mechanisms to prevent content fabrication attacks. The prompt injection result is conditionally reassuring but should not be taken as a guarantee—it reflects current frontier model hardening, not SOM-specific protection.

The path forward is framework integration. The SOM ecosystem has built the right infrastructure; the agent frameworks need only to look for it.

References

[1] Plasmate Research Group. *Semantic Object Model (SOM): A Structured Web Representation for AI Agents*. Plasmate Technical Report, 2025.

[2] Plasmate Research Group. *Agentic Web Protocol (AWP): A Minimal Interface for Agent-Web Interaction*. Plasmate Technical Report, 2025.

[3] Plasmate Research Group. *SOM Directives for robots.txt: Publisher-Side Signaling for Agent-Accessible Content*. Plasmate Technical Report, 2025.

[4] Plasmate Research Group. *Publisher Cost-Benefit Analysis of SOM Deployment*. Plasmate Technical Report, 2025.

[5] Plasmate Research Group. *Token Economics of Semantic Web Representations for AI Agents*. Plasmate Technical Report, 2025.

[6] Plasmate Research Group. *Information Fidelity under Semantic Compression: HTML, Markdown, and SOM*. Plasmate Technical Report, 2025.

[7] Plasmate Research Group. *WebTaskBench: A Benchmark for Evaluating AI Agent Task Completion on Real Web Content*. Plasmate Technical Report, 2025.

[8] T. Koster, G. Illyes, H. Zeller, L. Harvey. *Robots Exclusion Protocol*. RFC 9309, IETF, September 2022.

[9] R. Fielding, J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. RFC 7231, IETF, June 2014.

[10] H. Furuta, P. Nachum, K. Lee, Y. Matsuo, S. Gu, S. Irie. *Multimodal Web Navigation with Instruction-Finetuned Foundation Models*. ICLR 2024.

[11] S. Yao, H. Chen, J. Yang, K. Narasimhan. *WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents*. NeurIPS 2022.

[12] A. Shridhar, J. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, M. Hausknecht. *ALFWorld: Aligning Text and Embodied Environments for Interactive Learning*. ICLR 2021.

[13] Y. Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv:2312.10997, 2023.