

# International Journal of Technology and Systems (IJTS)

**WebMCP: A Browser-Native navigator.modelContext Standard for  
Agentic Web Automation**



**CARI  
Journals**

## WebMCP: A Browser-Native navigator.modelContext Standard for Agentic Web Automation

Harsha Nooka , Suhas Madupuri 

<https://orcid.org/0009-0000-1680-6605> <https://orcid.org/0009-0000-5318-8703>

Accepted: 19<sup>th</sup> Sep, 2026, Received in Revised Form: 4<sup>th</sup> Sep, 2026, Published: 15<sup>th</sup> Oct, 2026

### Abstract

**Purpose:** This paper examines WebMCP, a proposed browser-native web platform standard that lets websites expose explicit, schema-described tools to large language model agents, as an alternative to actuation, in which an agent simulates the clicks, keystrokes, and navigation a human would perform.

**Methodology:** The paper analyzes WebMCP, incubated within the World Wide Web Consortium's Web Machine Learning Community Group and implemented as an origin trial in the Chrome browser beginning with Chrome 149. We describe WebMCP's architecture, including the browser-mediated JavaScript interface navigator.modelContext and a declarative HTML annotation form, and its relationship to Anthropic's Model Context Protocol; analyze its security and permission model; place it within a comparative design-space taxonomy alongside DOM automation, computer-use agents, and server-side Model Context Protocol integrations; and examine the emerging standardization landscape, including browser vendor positions and adjacent efforts such as the Agentic Commerce Protocol. A generalized use case illustrates how a composite retail website might expose search, filtering, and checkout tools under this model. All sources and data referenced in this manuscript are dated on or before June 30, 2026.

**Findings:** WebMCP allows a website to register tools that an agent invokes directly rather than inferring actions from pixels or DOM structure, with a security model built on origin isolation, a dedicated Permissions Policy directive, and a human-in-the-loop confirmation mechanism for consequential actions. Open problems remain in cross-site tool discovery, adversarial tool poisoning, and the evaluation of agent-web interaction quality.

**Unique Contribution to Theory, Policy and Practice:** For theory, the paper contributes a comparative design-space taxonomy that positions browser-native, declarative agent integration against actuation-based and server-side approaches. For practice, it gives web developers and agent builders a technical account of WebMCP's registration mechanisms, security model, and interaction flow. For policy, it informs browser vendors and standards bodies through its analysis of the standardization landscape and a research agenda for browser-native agent integration standards as this area continues to develop.

**Keywords:** *WebMCP, Model Context Protocol, Agentic Web, Browser Standards, Large Language Model Agents, Web Security*

**JEL Codes:** *L86, L15, O33*

## 1. Introduction

For most of the web's history, software that acted on a person's behalf inside a browser had to do so indirectly. Screen readers walked the accessibility tree to describe a page to a person who could not see it. Testing frameworks and robotic process automation tools located an element by its coordinates, its text content, or a brittle CSS selector and dispatched a synthetic click. Neither approach involved the website's developer in deciding what an automated actor was allowed to do or how it should do it; the automation worked despite the page, not with it. This indirection was tolerable when the actor was a deterministic test script written and maintained by someone who understood the page's structure. It is a much larger liability when the actor is a large language model agent that must infer intent from pixels or markup, that can be misled by content on the page it is trying to interpret, and that is increasingly being trusted to complete tasks with real consequences, such as filling out a form, comparing prices, or completing a purchase.

The technique of having an agent operate a page the way a person would, commonly called actuation, has nonetheless become the default integration path for agentic browsing, because it requires no cooperation from the website. Surveys of computer-use and graphical interface agents published through 2025 document rapid capability gains in this approach, alongside a persistent and well-documented fragility: small changes in a page's layout, wording, or DOM structure can break an agent's plan, and the interface an agent must reason about, a screenshot or an accessibility tree built for human perception, was never designed to be an unambiguous machine contract [3], [4], [5]. At the same time, a parallel and much older line of engineering established that when a system's designer can describe its capabilities in a structured, machine-readable form, whether that is a function signature, an OpenAPI document, or a tool schema consumed by a language model, the resulting integration is more reliable than one built by inferring behavior from an interface meant for people [7], [8]. Anthropic's Model Context Protocol formalized this idea for language model agents in general, giving agents a standard way to discover and call tools exposed by a server [1]. What remained unresolved through most of 2025 was how that same idea should apply inside the browser itself, where the party best positioned to describe a page's capabilities, the website's own developer, had no standard mechanism to do so directly to an agent operating on that page.

WebMCP is a proposal, incubated within the World Wide Web Consortium's Web Machine Learning Community Group and backed by engineers from Microsoft and Google, that closes this gap [2]. It introduces a browser-mediated JavaScript surface, `navigator.modelContext`, through which a page can register named, schema-described tools, alongside a declarative form that lets ordinary HTML forms opt into the same mechanism through markup rather than script. An agent operating on behalf of a user, whether that agent lives inside the browser itself, in a companion application, or in a cross-origin frame explicitly granted permission, invokes these tools directly, receiving structured results instead of a screenshot to reinterpret. Chrome began an origin trial of

the API with Chrome 149 in June 2026, giving developers a live implementation to test against for the first time [21], [22]. As of this writing, the proposal remains under active discussion; Mozilla opened its formal standards-position review in late May 2026, and the specification itself continues to evolve through the community group's public issue tracker [23].

This paper has three goals. First, it describes WebMCP's architecture, its two registration mechanisms, and its human-in-the-loop interaction model in enough technical detail to be useful to readers who have not yet read the proposal directly. Second, it analyzes the security and permission model the proposal builds on, situates the tool-poisoning and prompt-injection risks that have been documented for the server-side Model Context Protocol in the specific context of a browser-mediated variant, and asks which of those risks the browser's involvement actually mitigates. Third, it places WebMCP within a broader design-space taxonomy that includes DOM automation, computer-use agents, server-side Model Context Protocol integrations, and prior machine-readable web semantics such as structured data markup and the accessibility tree, so that the trade-offs of choosing a browser-native, declarative approach are visible alongside the alternatives rather than treated in isolation.

The scope of this paper is bounded in ways worth stating up front. It is not a specification document; where the proposal's text and this paper's summary of it might diverge because the specification has since changed, the authoritative source is the specification itself. It does not report new empirical measurements of WebMCP's performance or reliability relative to actuation-based agents, since the origin trial began only weeks before this manuscript's cutoff and no independent evaluation had been published by that date. Its use case in Section 8 is, correspondingly, illustrative rather than a report of a deployed integration. Finally, this manuscript's knowledge cutoff is June 30, 2026; the standardization process it describes was active and unsettled at that date, and readers consulting this paper later should treat its account of the proposal's status, browser support, and standards-body positions as a snapshot rather than a current status.

The remainder of the paper is organized as follows. Section 2 reviews related work in agent-web interaction, the Model Context Protocol, computer-use agents, and prior machine-readable web semantics. Section 3 describes WebMCP's architecture and API surface. Section 4 analyzes its security and permission model. Section 5 presents a comparative design-space taxonomy. Section 6 reviews the standardization landscape. Section 7 discusses evaluation considerations. Section 8 presents an illustrative use case. Section 9 discusses open challenges, Section 10 proposes a research agenda, Section 11 states this survey's limitations, and Section 12 concludes.

## **2. Background and Related Work**

### ***2.1 From Screen Scraping to Tool Calling: The Evolution of Agent-Web Interaction***

Automated interaction with web pages predates language model agents by decades, but the last three years produced a specific technical lineage worth tracing. Chain-of-thought prompting

demonstrated that a language model's intermediate reasoning could be elicited and used to guide multi-step behavior [6]. The ReAct framework showed that interleaving that reasoning with discrete actions against an external environment, such as issuing a search query and reading the result, let a model solve tasks that required both deliberation and interaction, and web navigation was among the environments used to demonstrate the approach [7]. Toolformer showed, separately, that a model could learn largely on its own when to invoke an external tool, foreshadowing the tool-calling interfaces that most production language model systems now expose [8]. Neither line of work was specific to the browser, but both established the pattern WebMCP now applies there: an agent that can call a well-defined function is more reliable than one that must infer an action from an unstructured observation.

Applied to the browser specifically, this pattern first appeared as agent-computer interfaces built for narrow domains. SWE-agent demonstrated that a purpose-built interface for reading and editing source code, rather than a general-purpose file system view, measurably improved an agent's ability to complete software engineering tasks, and the authors argued explicitly that the interface an agent uses is not incidental to its performance but a determinant of it [9]. That argument generalizes directly to the web: an agent's success on a task like completing a purchase or filing a support ticket depends heavily on whether the page exposes that task as a well-defined action or leaves the agent to reconstruct it from a form's visual layout.

## ***2.2 The Model Context Protocol and the Rise of Agent Tool Ecosystems***

Anthropic introduced the Model Context Protocol in November 2024 as an open standard for connecting language model applications to external tools, data sources, and systems through a common client-server architecture built around three primitives, tools that a model can invoke, resources it can read, and prompts that structure an interaction [1]. The protocol's data layer defines how a client discovers what a server offers and how it invokes it, while its transport layer is agnostic to whether the client and server run on the same machine or communicate over a network. Over the following year, the protocol accumulated a wide ecosystem of servers exposing databases, software-as-a-service platforms, and internal enterprise systems as agent-callable tools, a pace of adoption significant enough that the protocol's maintainers marked a one-year anniversary in November 2025 with a retrospective on the specification's evolution and the growth of its server ecosystem [20].

The Model Context Protocol, as originally conceived, describes an agent calling a server that the agent's operator configures in advance. It does not, by itself, describe how a person's own web browser, already open to a specific page with its own session, permissions, and rendered state, should expose that page's capabilities to an agent in a way that respects the browser's existing security boundaries. WebMCP is explicit about this gap: its proposal frames the browser as an intermediary that can offer many of the same primitives the Model Context Protocol defines, tools with names, descriptions, and schemas, while decoupling the page from any single version of that

protocol and applying security policies specific to the web platform, such as origin isolation and iframe permission delegation, that a generic server-based transport does not address [2]. In effect, WebMCP proposes the browser as a translation layer between whatever protocol an agent client speaks and the tools a page chooses to expose, rather than asking every website to implement a full external protocol server.

### ***2.3 Computer-Use and GUI Agents: Capabilities and Limits***

A substantial body of work has pursued the alternative path: agents that operate a graphical interface directly, the same way a person does, without requiring any cooperation from the application being operated. Surveys of this area, covering agents for general computer use, phone interfaces, and browsers specifically, describe systems that parse a screenshot or an accessibility tree, reason about the likely function of visible elements, and issue simulated pointer and keyboard events [3], [4], [5]. This approach has an important advantage that any browser-native, opt-in standard like WebMCP lacks by construction: it works on the vast majority of existing websites without any change to those websites, because it relies on the interface already built for human users rather than on a new interface a developer must choose to add.

The same surveys, however, converge on a consistent set of limitations. Because these agents reconstruct intent from an interface designed for human perception, they are vulnerable to layout changes that break no human user's experience but silently invalidate an agent's plan; they must expend model capacity and computation reasoning about visual or structural details irrelevant to the task; and because the content they read to decide what to do is the same content an attacker could manipulate, they are susceptible to adversarial content embedded in a page steering the agent toward an unintended action, a risk that is a browser-specific instance of the broader prompt-injection problem documented for tool-calling systems generally [19]. WebMCP's proposal explicitly frames itself as addressing developer exclusion from this loop: because actuation requires no cooperation from a site, the site's developer has no say in what the agent is permitted to do or how reliably it can do it, and WebMCP's authors argue that giving developers a direct channel to describe their site's capabilities is preferable to leaving that description to be inferred [2]. This paper's Section 5 returns to this trade-off directly, since neither approach dominates the other across every dimension that matters.

### ***2.4 Prior Art in Machine-Readable Web Semantics***

WebMCP is not the first attempt to give machines a more structured view of a web page than its rendered pixels. The accessibility tree, exposed by every modern browser to assistive technology, already provides a semantic, structured representation of a page's interactive elements, and the World Wide Web Consortium's accessibility guidance establishes requirements for how that structure should represent operable content so that it remains usable by assistive technology regardless of a page's visual design [12]. Structured data markup, such as the vocabularies promoted for search engine consumption, gives machines a way to read facts about a page's

content, though it was designed for indexing rather than for describing actions a reader could take. More recently, and largely in parallel with WebMCP, site operators have experimented with plain-text discovery files intended to give a language model a curated summary of a site's content, and commerce platforms have begun publishing protocols, such as the Agentic Commerce Protocol introduced by OpenAI and Stripe in September 2025, that let a retailer describe a structured checkout flow an agent can complete on a shopper's behalf [13].

These efforts share a premise with WebMCP, that a machine-readable description authored by the party who best understands a resource is more reliable than one inferred by an outside observer, but they differ in an important structural respect. Accessibility trees, structured data, and discovery files are fundamentally descriptive: they tell a consuming agent what is present or what the page is about. WebMCP is prescriptive in a stronger sense: it lets a page register an executable action, with a schema constraining its inputs and a function that runs with the page's own privileges when the action is invoked. This distinction matters for the security analysis in Section 4, because a prescriptive interface that executes code on invocation carries a different risk profile than a descriptive one that only supplies information for a model to reason about.

### **3. The WebMCP Proposal: Architecture and API Surface**

#### ***3.1 Design Goals***

The WebMCP proposal, authored by engineers from Microsoft and Google within the Web Machine Learning Community Group, states its motivation as increasing developer involvement in what its authors call the agentic web: giving the people who build a website a direct, low-effort way to describe what an agent may do on that site, rather than leaving that description to be inferred from the same interface built for human visitors [2]. The proposal identifies several concrete goals that shape the design choices described in the remainder of this section. It aims for code reuse, so that exposing an existing piece of client-side logic as a tool requires minimal modification to that logic. It aims to keep tool execution local to the page where practical, reducing round trips to a remote server and limiting how much of a user's session state must leave the browser to accomplish a task. It aims to let the browser mediate access and consent, so that a person retains visibility into which agent clients are requesting access to a page's tools. It aims to support interleaving agent actions with human input, so that a tool invocation can pause for an authentication step, a confirmation dialog, or any other moment where a person's direct involvement is required rather than optional. Finally, it aims to offer an ergonomic path for developers already familiar with the Model Context Protocol's concepts, while decoupling the page from any single version of that protocol [2].

#### ***3.2 The navigator.modelContext API***

The imperative core of the proposal is a new property on the navigator object, modelContext, exposing a registerTool method. A developer calls this method with an object describing a single

tool: a name that identifies it, a natural-language description an agent uses to decide when the tool is relevant, an inputSchema expressed in JSON Schema that constrains the parameters a caller may supply, and an execute function that runs when the tool is invoked. The execute function receives the validated parameters along with a second argument, an agent-facing context object, that lets the tool's own code request a synchronous interaction with the person at the keyboard in the middle of an otherwise asynchronous tool call [2]. Figure 1 illustrates this flow: an agent client discovers the tools a page has registered, invokes one with a set of arguments, and the page's own JavaScript executes with the same privileges any other script on that page would have, optionally pausing to ask the user directly before completing the action.

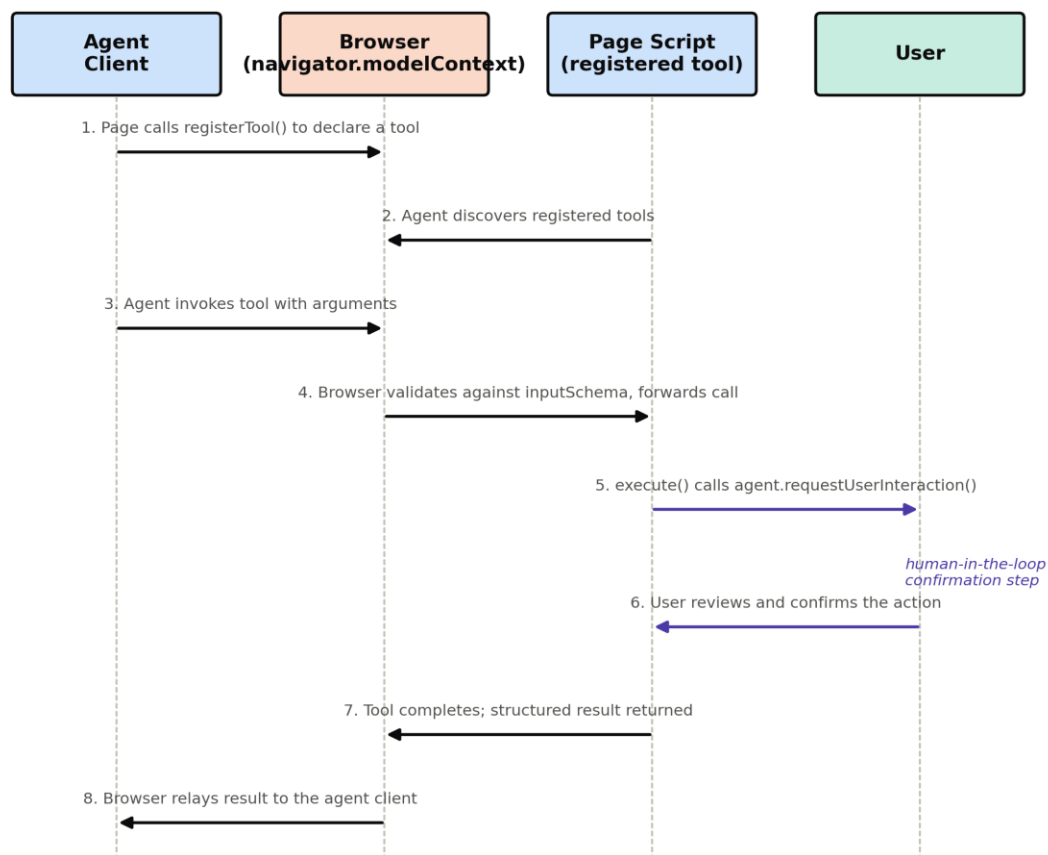


*Fig. 1. Sequence of a WebMCP tool invocation, from agent discovery through browser-mediated execution and an optional human-in-the-loop confirmation step.*

Tools registered this way can be withdrawn using the same AbortController-based cancellation pattern already used elsewhere in the web platform, so that a single-page application can register and unregister tools as its own internal view changes without needing a bespoke unregistration API. The proposal's authors emphasize that this design deliberately reuses concepts working web

developers already know, a JSON-Schema-described function and a standard cancellation primitive, rather than introducing an entirely new programming model specific to agents [2].

### ***3.3 Declarative Tool Registration via HTML***

Alongside the imperative registerTool interface, the proposal and its associated developer documentation describe a declarative path: annotations added directly to standard HTML form markup that let a form opt into becoming a WebMCP tool without a developer writing JavaScript to describe it. This path targets the large share of web tasks, such as filtering a list of results or submitting a simple form, that are already expressed as ordinary HTML forms and do not need the full expressiveness of a hand-written execute function. Reported early implementation experience singles out this declarative path as useful for structured, form-heavy tasks such as multi-passenger travel booking or date and time selection, where mapping an agent's structured input directly onto existing form fields removes a class of errors that would otherwise come from an agent guessing at a form's layout.

The coexistence of an imperative and a declarative registration path is a notable design choice relative to comparable web platform features, most of which pick one or the other. It reflects a judgment that the tasks worth exposing to agents span a spectrum from simple, form-shaped actions that benefit from a low-effort declarative annotation to complex, stateful actions, such as a multi-step checkout with conditional logic, that need the full flexibility of a JavaScript function.

### ***3.4 The Human-in-the-Loop Interaction Model***

Because a tool's execute function runs arbitrary page JavaScript, WebMCP's authors built a specific mechanism for that function to request a person's direct involvement mid-execution rather than only before or after the call. The agent context object passed to execute exposes a method, described in the proposal as agent.requestUserInteraction, that lets a tool pause, present a user interface element such as a confirmation dialog or an authentication prompt, and resume only once the person has responded [2]. This mechanism gives a tool author fine-grained control over which steps of a multi-step action require a person's explicit confirmation and which can proceed without interruption, directly instantiating the same intervention-point concept, letting a human meaningfully act at specific, well-defined points in an otherwise autonomous process, that a broader literature on human-in-the-loop system design has described in non-web-specific settings [14], [15]. Whether a given website author uses this mechanism conscientiously, reserving it for genuinely consequential actions rather than either overusing it in a way that trains users to dismiss it reflexively or underusing it in a way that lets an agent complete a harmful action unchecked, is a design responsibility WebMCP places on the site rather than one the standard enforces structurally, a point this paper returns to in Section 4 and Section 9.

The proposal also specifies that tool calls execute sequentially rather than concurrently within a given page context, which the authors note as a deliberate simplification that avoids a class of race

conditions that would otherwise arise if two tool invocations could interleave their effects on shared page state [2]. This constraint trades away some potential throughput for a substantially simpler mental model of what state a tool's execute function can assume when it runs, which matters directly for a developer trying to reason about whether their tool implementation is safe to expose to an agent they do not fully control.

## **4. Security and Permission Model**

### ***4.1 Origin Isolation and the Permissions Policy***

WebMCP's threat model starts from an assumption familiar to anyone who has worked with other sensitive web platform features: a capability this powerful, letting an outside caller invoke code that runs with a page's own privileges, must be constrained by the same origin-based security boundaries that govern the rest of the platform, and must be possible for a page or a site operator to disable entirely. According to Chrome's implementation documentation, a document must be origin-isolated to use the API, and the feature is unavailable if a page has relaxed that isolation by setting `document.domain`, a legacy mechanism that widens a page's trust boundary and is incompatible with the stronger guarantees WebMCP depends on. Access is further gated by a dedicated entry in the web platform's Permissions Policy framework, defaulting to a same-origin-only scope, so that a cross-origin iframe cannot register or expose tools unless the embedding page explicitly grants it permission through an `allow` attribute. This places WebMCP's default security posture in line with other capability-sensitive web platform features that adopted an explicit, opt-in delegation model for cross-origin use rather than an implicit, always-available one.

This design has a direct consequence for how a page composed of content from multiple origins, such as a retail site embedding a third-party payment widget in an iframe, must think about tool exposure. The embedding page decides, through the Permissions Policy directive, whether an embedded frame is allowed to register tools at all, which keeps the decision about what a cross-origin component may expose to an agent in the hands of the page that chose to embed it rather than leaving that decision entirely to the embedded component itself. This mirrors the general pattern established for cross-origin communication on the web, where a receiving context must explicitly opt in to trust a message from another origin rather than trusting it by default [16].

### ***4.2 Threat Model: Tool Poisoning and Prompt Injection***

The security literature that grew up around the server-side Model Context Protocol during 2025 documented a specific and now well-known attack pattern commonly called tool poisoning: an attacker controls or compromises a tool's registration, most often its natural-language description, and embeds instructions in that description that are invisible or innocuous to the human developer who approved the tool but that an agent's underlying language model reads and follows as though they were legitimate instructions from its operator. Independent security researchers documented working demonstrations of this pattern against Model Context Protocol servers during the first half

of 2025, and the Open Worldwide Application Security Project subsequently catalogued tool poisoning as one of its top identified risks specific to the Model Context Protocol [19], [20]. A closely related risk, indirect prompt injection, does not require a malicious tool registration at all; it only requires that content an agent reads while completing a task, whether that is a tool's output or, in the browser-native case, the surrounding page itself, contain text crafted to be interpreted by the agent's underlying model as an instruction rather than as data [19].

WebMCP's browser-mediated design changes this threat model in ways that cut in both directions rather than uniformly improving on it. On one hand, because a tool's registration lives in the page's own script and is visible to anyone who can read that page's source, and because the browser, not an opaque remote server the agent's operator must trust sight unseen, mediates every invocation, a user or a browser vendor has more opportunity to inspect, log, or restrict what a given origin's tools are permitted to do than they typically would for an arbitrary remote Model Context Protocol server whose implementation is not visible to the client at all. On the other hand, WebMCP does not remove the fundamental prompt-injection surface: a malicious or compromised site can still write a tool description, or the surrounding page content an agent reads to decide how to use that tool, in a way designed to manipulate the agent's behavior, and because tool code executes with the full privileges of its origin, a successful manipulation can act on any capability that origin already has, including a user's authenticated session on that site. The proposal's sequential-execution guarantee and its origin-isolation requirement reduce specific classes of failure, such as cross-tool race conditions and cross-origin capability leakage, but they do not by themselves validate that a given tool's description or behavior is trustworthy, a judgment that still rests on whatever trust the user or their agent client places in the site's origin.

#### ***4.3 Consent, Auditability, and Sequential Execution***

The proposal's stated design gives the browser two additional levers for managing this residual risk: a consent surface, letting a person see which client applications are requesting access to a given page's tools and approve or deny that access, and the human-in-the-loop confirmation mechanism described in Section 3.4, letting a tool itself pause for explicit confirmation before completing a consequential step. Neither lever is self-enforcing. The consent surface depends on the browser presenting a request in a way a person can meaningfully evaluate, a problem with a long history of poor outcomes elsewhere on the web platform when permission prompts are frequent, generic, or presented at a moment disconnected from the action they govern. The confirmation mechanism depends on the site's own developer choosing to invoke it for the right steps, which is a voluntary, per-site design decision rather than a property the standard can guarantee across every implementation. Table I summarizes the principal risks discussed in this section against the mitigations WebMCP's current design provides and the residual exposure that remains.

**Table I. Threat Categories Relevant to WEBMCP, the Mitigations its Current Design Provides, and Residual Exposure.**

| Threat Category                              | WebMCP Mitigation                                                            | Residual Exposure                                                                                      |
|----------------------------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| Cross-origin capability leakage              | Origin isolation requirement; Permissions Policy gating iframe access        | A permitted cross-origin frame still executes with the delegating page's granted scope                 |
| Tool description poisoning                   | Tool source is visible in page script, inspectable by user or browser vendor | Visibility does not equal verification; a page can still ship a manipulative description               |
| Indirect prompt injection via page content   | Not directly addressed by the API surface itself                             | Agent must still defend against adversarial content when reasoning about a page                        |
| Race conditions across concurrent tool calls | Sequential execution of tool calls within a page context                     | Does not address races between a tool call and unrelated page script                                   |
| Uninformed or fatigued consent               | Browser-mediated consent surface for client applications                     | Effectiveness depends on prompt design and frequency, a general weakness of browser permission prompts |

The overall picture that emerges is that WebMCP relocates several security decisions to a point where they are more visible and more consistently enforceable, the browser's own trust boundary, than they are for actuation-based agents or for opaque remote tool servers, without eliminating the underlying problem that an agent's behavior is only as trustworthy as the content and code it is willing to treat as authoritative. This is consistent with the broader finding in human-automation research that a well-designed interface can improve a supervisor's ability to detect and correct a failure but cannot substitute for the supervisor's own calibrated vigilance, particularly once a system's reliability record encourages complacency [17], [18].

## 5. A Comparative Design-Space Taxonomy for Agent-Web Interaction

The preceding sections described WebMCP on its own terms. This section places it alongside the three other approaches an agent builder can currently choose from when deciding how a language model agent should interact with a website: actuation of the human interface, either through DOM inspection or through pixel-level computer use; a server-side Model Context Protocol integration maintained independently of the website itself; and reuse of the accessibility tree that browsers already expose for assistive technology. None of these four approaches dominates the others across every dimension that matters for a practical deployment, and the choice among them depends heavily on whether the party building the agent controls the website it needs to interact with.

Actuation-based approaches, whether driven by DOM queries or by a vision model interpreting screenshots, have the singular advantage of universality: they work, with varying reliability, on

any website already built for human users, requiring no cooperation or code change from that site's developer [3], [4], [5]. Their central weakness is exactly the fragility discussed in Section 2.3: the interface being operated was designed for human perception and human error tolerance, not for unambiguous machine interpretation, so a layout change, an unexpected modal dialog, or adversarial content on the page can each derail an agent's plan in ways a structured tool call would not.

Server-side Model Context Protocol integrations solve the reliability problem by giving an agent a stable, versioned, explicitly documented tool interface, but that interface is typically built and maintained by the agent's operator or by a third-party integrator, independent of the website's own developer and, in many cases, independent of the website entirely. This has two consequences. It means the integration can drift out of sync with the underlying site's actual behavior without the site's developer being involved in fixing it, and it means a user's authenticated, in-browser session state, such as their current shopping cart or a partially completed form, is generally not directly available to a remote server-side tool the same way it is available to code running in the page itself.

The accessibility tree offers a third path, structured and semantic rather than purely visual, and already present on effectively every website without any additional developer effort, since it is a byproduct of standards-compliant markup rather than a bespoke integration [12]. Its structure, however, was designed to answer the question of what a screen reader user can perceive and operate, not the question of what discrete, well-defined actions a page supports; two buttons that are visually and semantically identical to a screen reader may correspond to entirely different application-level actions that only the page's own code can distinguish reliably.

WebMCP occupies a fourth position: like a server-side Model Context Protocol integration, it gives an agent an explicit, schema-described tool interface rather than an inferred one, but like the accessibility tree, it is authored by the site's own developer, runs with access to the page's actual in-browser state, and requires no separate infrastructure to operate. Its cost, relative to the other three approaches, is that it only works on sites whose developers have chosen to implement it, which means an agent operating across the general web will likely need to fall back to actuation or to the accessibility tree for the large majority of sites that have not adopted the standard, at least during the period immediately following any such standard's introduction. Table II summarizes this four-way comparison across the dimensions most relevant to a practical deployment decision.

**Table ii. Comparison of Agent-To-Website Interaction Approaches across Dimensions Relevant to Reliability, Coverage, and Developer Involvement.**

| Approach                                       | Requires Site Cooperation                 | Reliability to Layout Change               | Access to In-Browser Session State             | Developer Control Over Exposed Actions                        |
|------------------------------------------------|-------------------------------------------|--------------------------------------------|------------------------------------------------|---------------------------------------------------------------|
| DOM or vision-based actuation                  | No                                        | Low; brittle to layout and wording changes | Indirect, via the rendered page only           | None; behavior is inferred, not declared                      |
| Server-side Model Context Protocol integration | Partial; typically built by a third party | High for the declared tool surface         | Limited; generally outside the browser session | High for the declared tools, but decoupled from the live page |
| Accessibility tree consumption                 | No; a byproduct of standards compliance   | Moderate; semantic but not action-oriented | Indirect, via the rendered page only           | None; structure reflects markup, not application logic        |
| WebMCP (navigator.modelContext)                | Yes; requires explicit adoption           | High for the declared tool surface         | Direct; executes within the page's own context | High; tools are authored and scoped by the site               |

This taxonomy suggests that the approaches are complementary rather than strictly competing, and the strongest practical position, at least during any transition period, is an agent client that prefers a page's declared WebMCP tools where they exist and falls back to actuation or accessibility-tree-based inference where they do not, rather than a client that commits to a single approach across the entire web. Several of the browser agent products already deployed by mid-2026 describe exactly this kind of layered strategy, treating a structured tool interface as a preferred, higher-confidence path and interface actuation as a general-purpose fallback, though the specific tool-preference logic used by any single deployed product was not, as of this manuscript's cutoff, documented in enough independently verifiable detail to characterize precisely.

## 6. Standardization Landscape and Ecosystem Positioning

### 6.1 Browser Vendor Positions

WebMCP is being incubated within the Web Machine Learning Community Group, a World Wide Web Consortium community group rather than a formal standards-track working group, which is the customary early venue for a web platform proposal before it accumulates enough cross-vendor agreement to advance further [2]. As of this manuscript's cutoff, its most concrete implementation signal came from Google, which shipped an origin trial of the API in Chrome beginning with Chrome 149 in June 2026, giving developers a live, flag-gated implementation to build against and giving the community group real usage data to inform the specification's continued evolution

[21], [22]. An origin trial is, by design, an experimental and time-limited deployment mechanism the Chromium project uses to gather implementation feedback before a feature is considered for unflagged, general availability, and its presence indicates active investment rather than a settled commitment to ship the feature permanently in its trial form.

Mozilla's formal review process for new web platform features, tracked publicly through its standards-positions repository, opened its WebMCP entry in late May 2026, with the review recorded as in a proposed, not yet decided, state as of this manuscript's cutoff [23]. The proposal's own authorship, drawn from engineers at both Microsoft and Google, indicates at least preliminary cross-vendor interest from two of the three major browser engine maintainers, though authorship of a community group proposal is a weaker signal than a formal, published standards position, and this paper deliberately does not characterize Mozilla's eventual position, or Apple's, since neither had been settled as of the cutoff date and this survey's own instructions require that it not speculate about developments after that date.

This pattern, a feature proposed and prototyped by some browser vendors while others have not yet stated a position, is a common and unremarkable stage in web standards development, but it has a direct practical implication for any developer or researcher evaluating WebMCP in the near term: a tool registered through `navigator.modelContext` will, for as long as browser support remains partial, only be reachable by agents running in or through a browser that implements the API, which means a website adopting WebMCP during this period should treat it as an additional, progressively enhanced capability layered on top of whatever actuation-compatible interface the site already presents to human users and to agents that cannot yet reach the new API.

## ***6.2 Relationship to Adjacent Standards Efforts***

WebMCP did not emerge in isolation, and understanding its likely trajectory benefits from situating it among several adjacent, roughly contemporaneous efforts aimed at making the web more legible to agents, even though these efforts differ in scope and are not formally coordinated with one another. The Model Context Protocol itself, described in Section 2.2, remains the broader standard WebMCP explicitly aligns with, and the protocol's continued growth, marked by its one-year anniversary retrospective in November 2025, suggests that whatever browser-native mechanism eventually standardizes for the web, compatibility with that protocol's tool and resource primitives will likely remain a design constraint rather than an incidental similarity [1], [24].

In the commerce domain specifically, OpenAI and Stripe introduced the Agentic Commerce Protocol in September 2025, an open specification that lets a retailer describe a structured checkout flow a shopping agent can complete on a buyer's behalf, launched alongside an integration letting people complete purchases from participating merchants directly within a conversational agent interface [13]. The Agentic Commerce Protocol and WebMCP address an overlapping problem, letting an agent complete a transaction reliably, from different architectural starting points: the

Agentic Commerce Protocol defines a checkout-specific interaction contract between a buyer's agent and a merchant's commerce backend, largely independent of whether the buyer is looking at the merchant's own web page at the time, while WebMCP defines a general-purpose, in-page tool mechanism that a retailer could use to expose a checkout tool alongside search, filtering, and any other page-specific capability, scoped to the page the shopper is actually viewing. A retailer adopting both would plausibly use the general WebMCP mechanism for in-page tasks and a commerce-specific protocol for the checkout step itself, though no publicly documented integration combining the two had been reported as of this manuscript's cutoff.

Separately, and with less formal standardization behind them, a number of site operators experimented during 2025 and into 2026 with plain-text discovery files intended to summarize a site's content for a language model, in the spirit of the long-established robots exclusion convention but aimed at guiding an agent's understanding rather than restricting a crawler's access. These efforts remain descriptive in the sense discussed in Section 2.4, telling an agent what a site contains rather than what actions it supports, and they are best understood as a complement to WebMCP's prescriptive tool registration rather than a substitute for it. Taken together, the pattern across all of these efforts, the Model Context Protocol, the Agentic Commerce Protocol, discovery files, and WebMCP itself, is a broad, multi-front industry response to the same underlying observation this paper opened with: that inferring an agent's actions from an interface built for people is less reliable than being told, directly and in a structured form, what that agent is permitted and equipped to do.

## 7. Evaluation Considerations and Open Benchmarks

Benchmarks built for actuation-based web agents, most prominently environments that measure whether an agent can complete a realistic task against a browsable website by clicking and typing as a person would, established the field's standard way of comparing agent capability [10]. These benchmarks, and the broader suites that measure an agent's performance across operating systems, databases, and other software environments in addition to the web, score an agent by whether it reaches a correct final state, largely independent of how it got there [11]. That scoring approach is a reasonable fit for actuation, where how an agent got there, meaning which sequence of clicks and keystrokes it used, is not really a separate variable from whether it succeeded, since actuation is the only channel available.

WebMCP complicates this picture in a way current benchmarks are not built to measure. Once a task can be completed through either actuation or a declared tool call, final-state task success alone no longer distinguishes the two paths, even though they differ substantially in the properties that matter for a production deployment: reliability across minor page changes, latency, the amount of a model's reasoning capacity consumed reconstructing intent from an interface rather than executing a known action, and the size of the surface an adversary could exploit. A benchmark that only scores task completion would report the same result for an agent that succeeded by correctly inferring a checkout button's function from a screenshot and an agent that succeeded by

calling a checkout tool with a validated schema, even though the latter is, on the argument this paper has made throughout, a meaningfully more robust way to have arrived at the same outcome.

A second gap concerns coverage rather than depth. Existing web-agent benchmarks were built against sites that do not implement WebMCP, for the straightforward reason that the standard did not exist when those benchmarks were constructed. Evaluating WebMCP specifically therefore requires either a new benchmark environment built around sites that have adopted the standard, which as of this manuscript's cutoff would necessarily be a small and non-representative set given how recently the origin trial began, or an evaluation methodology that can meaningfully compare an agent's performance on the same underlying task across both an actuation path and a WebMCP path on sites instrumented to support both, so that the comparison isolates the interaction mechanism rather than conflating it with differences between sites.

A third gap concerns the security properties discussed in Section 4. A benchmark that only measures whether a benign agent completes a benign task says nothing about how a given implementation, or a given site's tool descriptions, behaves under adversarial conditions, such as a page containing content crafted to manipulate a tool's description or a co-located, independently untrusted iframe attempting to exploit a permission misconfiguration. Evaluating this dimension requires a red-team-style methodology closer to the approach already used to document tool-poisoning attacks against server-side Model Context Protocol deployments, adapted to WebMCP's specific origin-isolation and permission-policy mechanisms, rather than an extension of a standard task-completion benchmark [19], [20].

Taken together, these three gaps, mechanism-aware scoring, representative site coverage, and adversarial evaluation, define what a meaningful WebMCP-specific benchmark would need to provide beyond what existing web-agent evaluation infrastructure already offers. None of the three is technically difficult in isolation, but building all three into a single, broadly adopted benchmark is a substantial undertaking that, as of this manuscript's cutoff, no published work had yet completed for WebMCP specifically. Section 10 returns to this gap as part of this paper's proposed research agenda.

## **8. Illustrative Use Case: Agent-Assisted Task Completion on a Composite Retail Site**

To make the preceding architectural and security discussion concrete, this section walks through a generalized, composite illustration of how a retail website might adopt WebMCP to support an agent completing a multi-step shopping task. The scenario is a synthesis built from the capabilities documented in WebMCP's own proposal and developer documentation rather than a report of any specific, named deployment, since no fully documented production integration of this kind had been published as of this manuscript's cutoff.

### **8.1 Scenario and Tool Registration**

Consider a retail site that wants an agent acting on a shopper's behalf to be able to search its catalog, narrow results by attributes such as size or color, add an item to a cart, and complete checkout, while keeping the final purchase step subject to the shopper's explicit confirmation. Using the imperative `navigator.modelContext.registerTool` interface described in Section 3.2, the site's developer registers four tools corresponding to these steps, each with a name, a natural-language description, and a JSON Schema describing its parameters. The search and filter tools are read-only and low-risk, and their execute functions can run to completion without pausing for confirmation. The add-to-cart tool changes session state but remains easily reversible, since removing an item from a cart is itself a trivial, low-risk action. The checkout tool is the one place in this flow where the site's developer chooses to invoke the `agent.requestUserInteraction` mechanism described in Section 3.4, pausing the tool's execution to present the shopper with a final, explicit confirmation naming the item, price, and shipping destination before the order is placed. Table III summarizes these four illustrative tool registrations.

**Table iii. Illustrative WEBMCP Tool Registrations for a Composite Retail Scenario, Showing How Risk Level Maps to the Human-In-The-Loop Confirmation Choice.**

| Tool Name       | Function                                                       | Risk Level                                   | Confirmation Behavior                                                                                    |
|-----------------|----------------------------------------------------------------|----------------------------------------------|----------------------------------------------------------------------------------------------------------|
| search_products | Query the catalog by keyword                                   | Low; read-only                               | Executes without interruption                                                                            |
| filter_results  | Narrow results by structured attributes                        | Low; read-only                               | Executes without interruption                                                                            |
| add_to_cart     | Add an item and quantity to the session cart                   | Low to moderate; easily reversible           | Executes without interruption; visible in cart state afterward                                           |
| checkout        | Complete purchase with a specified payment and shipping method | High; consequential and difficult to reverse | Pauses via <code>agent.requestUserInteraction</code> for explicit shopper confirmation before completing |

## 8.2 Interaction Flow

Figure 2 traces the resulting interaction. An agent acting on the shopper's behalf discovers the four registered tools when it begins working on the page, calls `search_products` and `filter_results` in sequence to narrow the catalog to items matching the shopper's stated preferences, and calls `add_to_cart` once a suitable item is identified. Each of these calls returns a structured result, such as a list of matching product identifiers with their prices, that the agent can reason over directly, without needing to parse a rendered results page or infer which visual element corresponds to which product. When the agent proceeds to checkout, the tool's execute function invokes `agent.requestUserInteraction` before finalizing the order, and the shopper's own confirmation, given directly to the page rather than relayed through the agent, is what actually authorizes the

purchase. Because this confirmation step runs as part of the page's own tool code rather than as a separate step the agent must remember to insert, the checkout-time confirmation cannot be silently skipped by an agent that has been manipulated into treating it as optional, which is one of the concrete practical benefits of implementing the human-in-the-loop checkpoint as a browser-mediated interaction rather than as an instruction the agent is merely expected to follow.

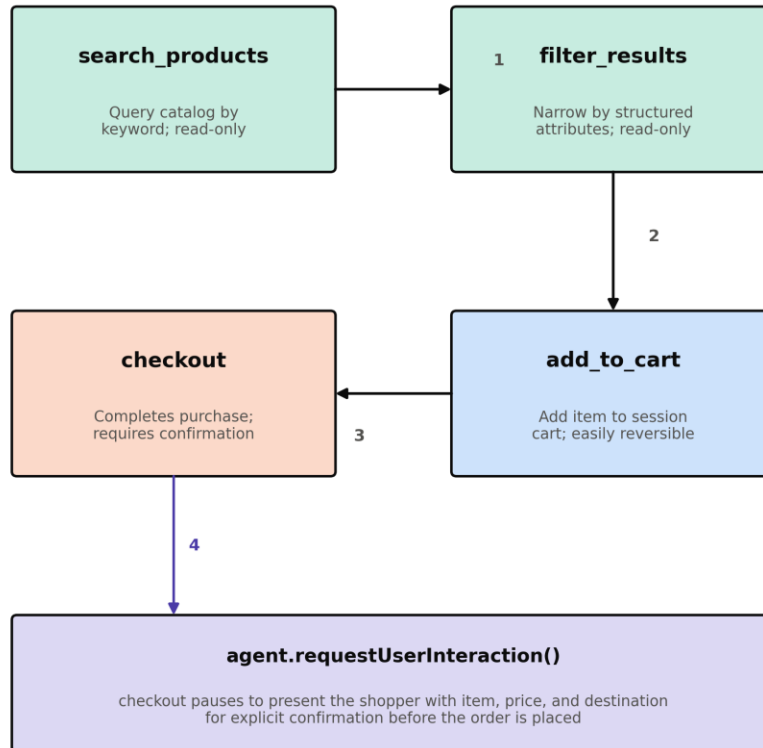


*Fig. 2. Interaction flow for the composite retail use case, showing four registered WebMCP tools and the checkout-time human-in-the-loop confirmation step.*

### 8.3 Discussion

This scenario illustrates several points argued more abstractly earlier in the paper. The search and filter tools show the reliability benefit described in Section 2.3: an agent calling `search_products` with a structured query and receiving a structured list of results back is not exposed to the layout fragility that would affect an agent trying to locate and operate a search box and results list by inspecting the rendered page, and it does not need to spend reasoning capacity distinguishing a genuine result from an advertisement or an unrelated page element. The tiered confirmation behavior across the four tools shows the proportional, risk-scaled approach to human intervention argued for in the security discussion of Section 4.3: applying the same friction to every tool call regardless of consequence would either slow down the low-risk search and filter steps for no safety

benefit or, if confirmation prompts became routine enough to be dismissed reflexively, weaken the very checkout confirmation the design most depends on.

The scenario also illustrates a limitation acknowledged directly in Section 4.2: registering these four tools does not, by itself, guarantee that their descriptions are accurate or that their execute functions behave as described. A shopper's agent, or the browser mediating access on the shopper's behalf, still has to place some baseline trust in the retail site's origin, the same trust a shopper already extends by choosing to visit and transact with that site in an ordinary browser session. What WebMCP changes is not the need for that baseline trust but the reliability and auditability of the interaction built on top of it, replacing an agent's inference about what a page's interface probably does with an explicit, schema-constrained declaration of what it is designed to do, checkable in principle by anyone who can read the page's own source.

## **9. Challenges and Open Problems**

This section draws together five open problems raised across the preceding sections, none of which had a settled answer in the published record as of this manuscript's June 30, 2026 cutoff.

### ***9.1 The Bimodal Web During Partial Adoption***

For as long as WebMCP support remains limited to an origin trial in a single browser engine, as it was at this manuscript's cutoff, any agent that wants broad web coverage must implement both a WebMCP-aware path and a fallback actuation path, and must decide, for a given page, which to trust when both are technically available. This is not a novel problem in web platform history; browsers and web developers have managed feature-detection and progressive enhancement for new capabilities for as long as the platform has evolved incrementally. It is, however, a problem with sharper stakes here than in most past instances, because the fallback path, actuation, is exactly the less reliable and more security-sensitive mechanism this standard was introduced to move away from, which means the transition period itself, not just the eventual steady state, is a meaningful object of study.

### ***9.2 Tool Description Trustworthiness at Scale***

Section 4.2 established that WebMCP does not, by itself, verify that a tool's description or behavior matches what it claims to do; it makes that description visible and places it under the browser's origin-based trust model rather than an opaque third-party server's. Visibility is necessary but not sufficient for trustworthiness at the scale a general-purpose web standard implies. A browser vendor, an agent operator, or a third-party auditor could in principle build reputation systems, static analysis tools, or runtime monitoring specifically for WebMCP tool registrations, analogous to the reputation and review mechanisms that have grown up around browser extensions and mobile application marketplaces, but no such mechanism specific to WebMCP had been published or standardized as of this manuscript's cutoff, and the proposal itself does not specify one.

### ***9.3 Cross-Site Tool Composition***

The scenario in Section 8 involved a single site's tools operating in isolation. Many realistic agent tasks span multiple sites, comparing prices across retailers, or gathering information from one site to complete an action on another, and WebMCP's origin-isolated design, appropriately for security reasons, does not provide a native mechanism for an agent to compose tools from multiple origins into a single coherent transaction with any shared guarantees beyond what the agent itself coordinates externally. Whether some future extension of the standard, or an entirely separate cross-origin coordination layer, should address this gap remains an open design question the community group had not resolved as of this manuscript's cutoff, and the tension is a real one: any mechanism that let tools from different origins share state or coordinate directly would reopen exactly the cross-origin trust boundary that origin isolation was designed to close.

### ***9.4 Discovery Without Navigation***

An agent currently discovers a page's WebMCP tools only once it has actually loaded that page, which means an agent trying to decide which of several candidate sites is worth visiting for a given task cannot yet inspect their tool surfaces without first navigating to each one. WebMCP's own proposal acknowledges this limitation directly and discusses, without committing to, a manifest-based alternative in which a site could declare its available tools in a separate, fetchable document, similar in spirit to how a web application manifest declares an installable application's metadata without requiring a browser to first render the application, though the proposal also notes that an agent would still need to navigate to the page to actually execute a discovered tool [2]. Resolving this gap well would materially change how an agent plans which sites to involve in a task at all, rather than only how it behaves once it has already chosen one, but as of this manuscript's cutoff the community group had deferred this direction rather than specified it.

### ***9.5 Interplay with Accessibility***

WebMCP's own proposal frames one of its benefits as enabling conversational interfaces as an alternative to the accessibility tree, which is a claim worth scrutinizing rather than accepting at face value [2]. For a person who relies on assistive technology and is directly operating a conversational agent themselves, a well-designed tool interface could plausibly offer a more direct and less effortful way to accomplish a task than navigating a complex page's accessibility tree element by element, which would be a genuine accessibility benefit. But the same framing carries a risk if it is read, incorrectly, as license to treat WebMCP tool coverage as a substitute for maintaining a genuinely accessible page structure: a site that under-invests in its accessibility tree on the assumption that assistive technology users will instead be served by an agent operating its WebMCP tools would be making an accessibility regression for any user, human or automated, who is not using such an agent, which is a large population that plausibly includes many assistive technology users themselves. The established requirement that operable content remain properly

represented in a page's accessibility structure regardless of what other interfaces a site also offers is not superseded by the existence of a tool-calling interface layered on top of it [12].

## 10. Future Research Directions

Building on the open problems identified in Section 9, this section proposes four directions for future research.

First, and most immediately actionable, the field needs a WebMCP-specific evaluation methodology of the kind described in Section 7: benchmarks that score the interaction mechanism, not only the final task outcome, across a representative set of sites that support both a WebMCP path and an actuation fallback, together with an adversarial evaluation track modeled on the tool-poisoning research already conducted against server-side Model Context Protocol deployments but adapted to WebMCP's origin-isolation and permissions-policy mechanisms specifically.

Second, empirical research is needed on how site developers actually use the human-in-the-loop confirmation mechanism described in Section 3.4 once real-world adoption moves beyond an origin trial. Because the standard leaves the decision of which actions warrant a pause entirely to each site's own developer, there is a meaningful risk of inconsistent practice across the web, with some sites over-using confirmation prompts in ways that train users toward reflexive approval and others under-using them in ways that leave consequential actions insufficiently checked. A study analogous to the guideline-development work already established for human-AI interaction more broadly, but specific to this tool-confirmation pattern, could produce concrete, evidence-based recommendations rather than leaving each site to rediscover the same trade-offs independently [14].

Third, research is needed on cross-site tool composition and manifest-based discovery, the two gaps identified in Section 9.3 and Section 9.4, with particular attention to whether either can be addressed without reopening the cross-origin trust boundary that origin isolation was designed to protect. This is likely to require close collaboration between the web security research community and the standards body incubating WebMCP, since a solution that satisfies functional demand for cross-site agent workflows but weakens the platform's existing origin-based security model would trade one class of problem for another rather than resolving it.

Fourth, longitudinal research is needed on the relationship between WebMCP adoption and web accessibility outcomes, directly testing the concern raised in Section 9.5: whether sites that adopt WebMCP tend, in practice, to maintain or improve their underlying accessibility tree quality, or whether tool-calling coverage for agents becomes a substitute that correlates with accessibility regression for the broader population of users, both human and automated, who do not go through an agent using those tools. This question cannot be answered from the standard's specification alone and requires empirical measurement across sites as adoption matures beyond the state observable at this manuscript's cutoff.

## 11. Limitations of This Survey

This survey has four limitations worth stating plainly. First, it describes a standard that was, at the time of writing, an active proposal under discussion within a World Wide Web Consortium community group and implemented only as a time-limited origin trial in a single browser engine; the specification's details, the set of browsers implementing it, and the positions of standards bodies reviewing it were all still capable of changing after this manuscript's June 30, 2026 cutoff, and this paper's account of them should be read as a snapshot of that date rather than a current status. Second, this paper's analysis of WebMCP's security properties in Section 4 is a structural and comparative analysis grounded in the proposal's documented design and in the broader, published record of tool-poisoning and prompt-injection research for the related server-side Model Context Protocol; it is not a report of an original penetration test or formal security audit of any specific WebMCP implementation, and readers making a security-sensitive adoption decision should seek a dedicated audit of the implementation they intend to rely on. Third, the use case in Section 8 is, by design, an illustrative composite built from the capabilities documented in WebMCP's proposal and developer materials, not a report of a named, deployed integration, since no fully documented production deployment of this kind had been published as of the cutoff; the tool-registration and confirmation patterns it describes should be read as a plausible application of the standard's documented capabilities rather than as measured outcomes. Fourth, this survey's source set, drawn from the standard's own primary documentation, a small number of independent technical news accounts of its origin trial, and the broader academic and industry literature on language model agents and web security, is necessarily incomplete for a topic this recent and fast-moving, and it should be supplemented with the community group's own issue tracker and specification text, which remain the authoritative and continuously updated source, for any reader who needs the current state of the proposal rather than its state as of this manuscript's cutoff.

## 12. Conclusion

WebMCP proposes a specific, concrete answer to a problem this paper has argued is structural rather than incidental: an agent that must infer its actions from an interface built for human perception will always be less reliable, and harder to secure, than one that is told directly and in a structured form what a page's developer intends it to be able to do. By introducing a browser-mediated `navigator.modelContext` interface, alongside a declarative HTML annotation path, WebMCP gives website developers a way to participate directly in the agentic web rather than being reduced to the passive target of an outside agent's inference, while keeping the browser positioned as the arbiter of origin isolation, permission scope, and, through the `agent.requestUserInteraction` mechanism, the moments at which a person's explicit confirmation is required before a consequential action proceeds.

This paper examined that proposal's architecture, its alignment with Anthropic's Model Context Protocol, and its security and permission model, and situated it within a comparative taxonomy

alongside DOM and vision-based actuation, server-side Model Context Protocol integrations, and accessibility-tree consumption, none of which fully subsumes the others. It reviewed a standardization landscape that, as of its June 30, 2026 cutoff, showed real implementation investment from Google through Chrome's origin trial beginning with Chrome 149, active but unresolved review from Mozilla, and a broader industry pattern, visible also in the Model Context Protocol's continued growth and in OpenAI and Stripe's Agentic Commerce Protocol, of parallel efforts converging on the same underlying insight from different starting points. It identified open problems in cross-site tool composition, discovery without navigation, tool description trustworthiness at scale, and the risk that tool-calling coverage for agents could be mistaken for a substitute rather than a complement to genuine accessibility investment, and it proposed a research agenda addressing each.

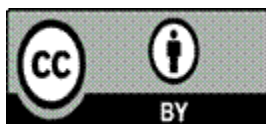
The central argument of this paper is that the choice of interaction mechanism between an agent and a website is not a minor implementation detail to be resolved after an agent's underlying reasoning capability is built, but a determinant of how reliably, securely, and legibly that capability can be exercised on the open web. Whether WebMCP specifically, in its current or a substantially revised future form, becomes the mechanism the web platform ultimately standardizes on was not yet settled as of this manuscript's cutoff, and depends on considerations, cross-vendor consensus, real-world security experience, and evidence of genuine reliability improvement over actuation, that only continued implementation experience and independent evaluation can resolve. What this paper has argued is that the underlying problem WebMCP addresses is real, that its specific design choices respond to that problem in ways grounded in established web platform security practice and in the broader human-in-the-loop design literature, and that the field now has, for the first time, a concrete, implemented proposal against which those design choices can be studied empirically rather than debated only in the abstract.

## References

- [1] Anthropic, "Introducing the Model Context Protocol," Nov. 2024.
- [2] B. Walderman, A. Nolan, D. Bokan, K. Sagar, and H. Van Opstal, "WebMCP proposal," Web Machine Learning Community Group, [webmachinelearning/webmcp](https://webmachinelearning.org/webmcp/), Aug. 2025.
- [3] D. Nguyen, J. Chen, Y. Wang, G. Wu, N. Park, Z. Hu, H. Lyu, J. Wu, R. Aponte, Y. Xia, X. Li, J. Shi, H. Chen, V. D. Lai, Z. Xie, S. Kim, R. Zhang, T. Yu, M. Tanjim, N. K. Ahmed, P. Mathur, S. Yoon, L. Yao, B. Kveton, J. Kil, T. H. Nguyen, T. Bui, T. Zhou, R. A. Rossi, and F. Deroncourt, "GUI agents: A survey," arXiv:2412.13501, Dec. 2024.
- [4] X. Hu, T. Xiong, B. Yi, Z. Wei, R. Xiao, Y. Chen, J. Ye, M. Tao, X. Zhou, Z. Zhao, Y. Li, S. Xu, S. Wang, X. Xu, S. Qiao, Z. Wang, K. Kuang, T. Zeng, L. Wang, J. Li, Y. E. Jiang, W. Zhou, G. Wang, K. Yin, Z. Zhao, H. Yang, F. Wu, S. Zhang, and F. Wu, "OS agents: A survey

- on MLLM-based agents for computer, phone and browser use," in Proc. 63rd Annu. Meeting Assoc. Computational Linguistics (ACL 2025), 2025.
- [5] P. J. Sager, B. Meyer, P. Yan, R. von Wartburg-Kottler, L. Etaiwi, A. Enayati, G. Nobel, A. Abdulkadir, B. F. Grewe, and T. Stadelmann, "A comprehensive survey of agents for computer use: Foundations, challenges, and future directions," arXiv:2501.16150, Jan. 2025.
  - [6] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in Proc. 36th Conf. Neural Information Processing Systems (NeurIPS 2022), New Orleans, LA, USA, 2022.
  - [7] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in Proc. 11th Int. Conf. Learning Representations (ICLR 2023), Kigali, Rwanda, 2023.
  - [8] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language models can teach themselves to use tools," in Proc. 37th Conf. Neural Information Processing Systems (NeurIPS 2023), New Orleans, LA, USA, 2023.
  - [9] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in Proc. 38th Conf. Neural Information Processing Systems (NeurIPS 2024), Vancouver, Canada, 2024.
  - [10] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig, "WebArena: A realistic web environment for building autonomous agents," arXiv:2307.13854, Jul. 2023.
  - [11] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang, "AgentBench: Evaluating LLMs as agents," arXiv:2308.03688, Aug. 2023.
  - [12] World Wide Web Consortium, "Web Content Accessibility Guidelines (WCAG) 2.2," W3C Recommendation, Oct. 2023.
  - [13] OpenAI, "Buy it in ChatGPT: Instant checkout and the Agentic Commerce Protocol," Sept. 2025.
  - [14] S. Amershi, D. Weld, M. Vorvoreanu, A. Fournery, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz, "Guidelines for human-AI interaction," in Proc. 2019 CHI Conf. Human Factors in Computing Systems, Glasgow, Scotland, UK, 2019, pp. 1-13.
  - [15] E. Horvitz, "Principles of mixed-initiative user interfaces," in Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, USA, 1999, pp. 159-166.

- [16] OWASP Foundation, "Cross-site scripting prevention and cross-origin messaging guidance," OWASP Cheat Sheet Series, accessed Jun. 2026.
- [17] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," Human Factors, vol. 39, no. 2, pp. 230-253, 1997.
- [18] J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," Human Factors, vol. 46, no. 1, pp. 50-80, 2004.
- [19] S. Willison, "Model Context Protocol has prompt injection security problems," Apr. 9, 2025.
- [20] OWASP Foundation, "MCP03:2025, Tool Poisoning," OWASP Model Context Protocol Top 10, 2025.
- [21] Chrome for Developers Blog, "Join the WebMCP origin trial," Jun. 2026.
- [22] InfoQ, "WebMCP standard proposal for agentic web actuation now available in Chrome (origin trials)," Jun. 2026.
- [23] Mozilla, "WebMCP," standards-positions, Issue 1412, opened May 28, 2026.
- [24] Model Context Protocol Blog, "One year of MCP: November 2025 spec release," Nov. 25, 2025.



2026 by the Authors. This Article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>)