International Journal of Engineering Research and Development
e- ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 22, Issue 7 (July 2026), PP 97-101

Trust-Aware AI Orchestration for Intelligent Web Applications: A Reference Architecture for Retrieval, Browser Agents and Edge Inference

Venkata Krishna Teja Kolli
Independent Researcher, United States
Corresponding Author: Venkata Krishna Teja Kolli, venkatakrishnatejakolli@gmail.com

ABSTRACT
Intelligent web applications increasingly combine large language models, retrieval-augmented generation, browser automation and local inference to provide adaptive user experiences. However, the web also introduces open-ended inputs, changing interfaces, untrusted documents, privacy-sensitive user data and tool actions that can affect external systems. This paper proposes a Trust-Aware AI Orchestration framework for intelligent web applications. The framework separates intent understanding, retrieval, browser-agent execution, edge inference, validation and human approval into explicit layers connected by provenance and least-privilege controls. The contribution is a reference architecture, an operational workflow and an evaluation protocol that application teams can use before deploying AI-enabled web features. The framework is designed to reduce overreliance on model output, limit prompt-injection impact, preserve data minimization and support traceable decisions across user-facing web workflows. The paper concludes that future intelligent web applications should be evaluated not only by task success but also by evidence quality, latency, privacy exposure, recoverability and controlled agency.

Date of Submission: 06-07-2026 Date of acceptance: 17-07-2026

I. INTRODUCTION

Web applications have moved from static information systems to adaptive environments that interpret user intent, retrieve evidence, generate recommendations and sometimes execute actions across external services. Search bars become conversational assistants, dashboards summarize operational data, coding portals explain repositories, commerce sites personalize product discovery and enterprise portals automate routine workflows. The common element across these examples is the use of artificial intelligence as an orchestration layer above the traditional web application stack.

This change creates a useful but difficult engineering problem. A conventional web application can be tested around fixed screens, deterministic business rules and bounded user inputs. An intelligent web application must handle natural language, untrusted documents, retrieved web pages, dynamically changing interfaces, private session data and model outputs that may be plausible without being correct. Web-agent benchmarks such as Mind2Web and WebArena demonstrate that real websites remain challenging for autonomous agents because pages are long, heterogeneous and domain specific [3], [4]. Browser-agent ecosystems continue to improve evaluation, but the gap between task completion and dependable production behavior remains material [7].

The research question addressed in this paper is: how can AI components be organized in web applications so that useful intelligence is provided while privacy, provenance, controllable agency and operational reliability remain explicit design properties? The paper answers this question by proposing a Trust-Aware AI Orchestration framework. The framework is not a new model architecture; rather, it is a system architecture for composing models, retrieval systems, browser tools, validation steps and human approval in a web application.

The paper makes three contributions. First, it defines a layered reference architecture for intelligent web applications that separates intent routing, retrieval, local inference, browser actions and response validation. Second, it maps common application risks, including prompt injection, sensitive information disclosure, excessive agency and overreliance, to concrete control points. Third, it proposes an evaluation protocol that measures task success together with evidence quality, privacy exposure, latency, recovery and auditability. The framework is intended for researchers and practitioners building AI-powered web systems where user trust and operational control matter as much as novelty.

II. RELATED WORK

Retrieval-augmented generation introduced a practical pattern for combining parametric language models with external knowledge sources. Lewis et al. showed that coupling generation with non-parametric memory can improve specificity and factuality for knowledge-intensive tasks [1]. In web applications, this pattern is attractive because product catalogs, help centers, project documentation and user records change faster than model weights. Yet retrieval also imports untrusted content into the model context, which means provenance, ranking and isolation have to be treated as application responsibilities rather than optional implementation details.

A second relevant line of work studies HTML understanding and web agents. Gur et al. showed that large language models can transfer to HTML understanding tasks and later proposed WebAgent, a modular agent that plans, summarizes long HTML and acts through generated programs [2], [5]. Mind2Web and WebArena broadened evaluation beyond simplified websites by using real or realistic domains and multi-step tasks [3], [4]. These studies show that AI can support web interaction, but they also show that current systems are brittle when pages are long, task sequences are compositional or interfaces change.

A third line of work concerns browser-side and edge inference. WebLLM demonstrated that browser-based large language model inference can use WebGPU and WebAssembly to make private and locally powered applications more practical [9]. More recent work on LlamaWeb points to memory planning and quantized execution as important for broad device coverage [10]. These developments are important for intelligent web applications because sensitive tasks can sometimes be handled locally, reducing server exposure and improving responsiveness. However, client-side inference also requires model distribution, capability gating and fallbacks when device resources are insufficient.

Finally, security and governance literature emphasizes that generative AI systems create risks beyond conventional web application security. Indirect prompt injection shows that instructions hidden in retrieved content can affect LLM-integrated applications [6]. OWASP identifies prompt injection, insecure output handling, sensitive information disclosure, excessive agency and overreliance as key LLM application risks [11]. NIST AI RMF 1.0 and the Generative AI Profile frame trustworthy AI as a lifecycle concern that includes governance, mapping, measurement and management [12], [13]. The proposed framework builds on these ideas by embedding controls into the application architecture.

III. PROPOSED FRAMEWORK

The proposed Trust-Aware AI Orchestration framework treats intelligence as a controlled service inside a web application rather than a single prompt attached to an interface. Its core assumption is that model output should not directly become user-facing advice or external action unless the system can explain where the input came from, which permissions were used, which validations were applied and whether human approval was required.

A. Architectural Layers

The first layer is the user intent layer. It receives natural language input, UI events and contextual signals from the current web session. This layer normalizes the request, identifies task type and estimates whether the request is informational, advisory or action-taking. The second layer is the intent and risk router. It assigns a risk class based on data sensitivity, external action potential, ambiguity and user role. Low-risk questions may use retrieval and generation directly; high-risk workflows require stronger evidence, approval and logging.

The third layer is the retrieval layer. It accesses approved sources such as application databases, documentation, user-uploaded files or selected web resources. Retrieved content is stored with source identifiers, timestamps and trust labels. The fourth layer is the browser-agent layer. It is used only when the application must observe or interact with web pages. The agent receives constrained actions rather than unrestricted access, and its observations are separated from trusted developer instructions.

The fifth layer is the edge inference layer. When the task contains sensitive data or requires low latency, compatible models may run in the browser through WebGPU or WebAssembly-backed runtimes. The final layer is the response and action gate. This gate checks citation coverage, validates structured outputs, blocks unsafe operations, asks for user confirmation when needed and records an audit event. Together these layers form an application-level trust boundary.

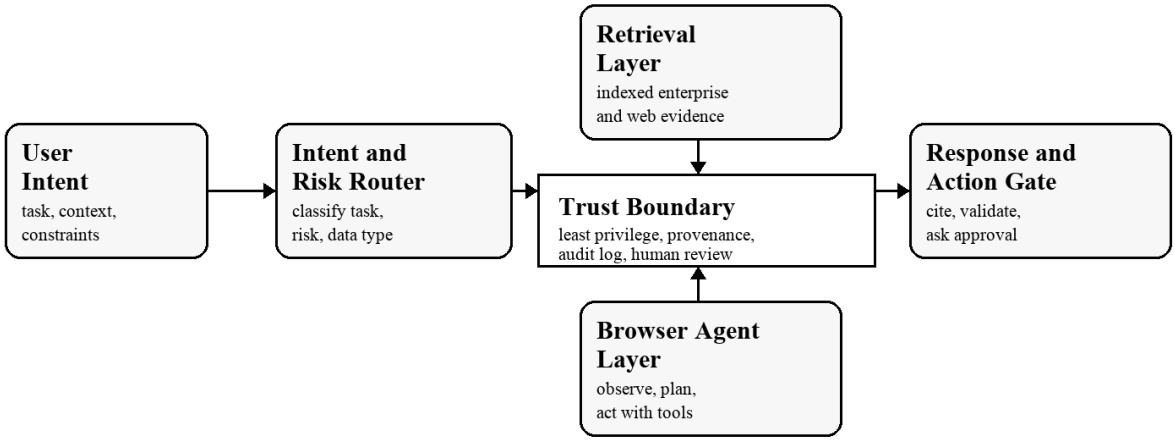

Figure 1: Trust-aware orchestration flow for intelligent web applications

B. Control Principles

The framework uses four control principles. The first is provenance by default: generated answers should preserve links between claims and the evidence that supported them. The second is least privilege: an agent should receive only the tools and data needed for the current task. The third is staged autonomy: the system may explain, draft or simulate before it is allowed to execute. The fourth is recoverability: every consequential action should have a review path, rollback path or compensating action. These principles make the intelligent behavior visible to developers, reviewers and users.

Table 1: Mapping of intelligent web application concerns to framework controls

Concern	Risk in Web Context	Framework Control
Prompt injection	Untrusted pages or documents may contain instructions that conflict with user intent.	Separate trusted instructions from retrieved content; scan and label external content; restrict tool access.
Sensitive data exposure	User session data, files or enterprise records may be copied into model context unnecessarily.	Apply data minimization, local inference where feasible and redaction before remote calls.
Excessive agency	An agent may click, submit, purchase or modify data without sufficient confirmation.	Use action classes, approval gates, least privilege tokens and explicit undo or rollback paths.
Confabulation	Generated answers may sound correct while lacking evidence.	Require citation coverage for factual claims; prefer structured extraction for critical fields.
Interface drift	Website markup and UI flows may change after testing.	Use robust selectors, observation summaries, canary tasks and continuous regression checks.
Overreliance	Users may accept model recommendations without review.	Display confidence limitations, source trails and human review checkpoints for consequential decisions.

IV. OPERATIONAL WORKFLOW

A typical workflow begins when a user asks the web application to perform a goal such as summarizing documents, comparing products, generating a support response or updating a record. The intent layer classifies the request into one of three categories. Informational tasks produce an answer, advisory tasks produce a recommendation, and action tasks attempt to change state in a system. This classification is important because a single model prompt should not decide its own authority.

After classification, the risk router assigns a risk level. A request to summarize public help documentation may be low risk. A request to analyze uploaded contracts may be medium risk because private text is involved. A request to submit a form, send a message or change account settings is high risk because it creates external consequences. The router then chooses an orchestration plan: retrieval only, retrieval plus local inference, browser-agent planning, or browser-agent execution with approval.

The retrieval layer collects candidate evidence. Each item is labeled by source type, access scope and freshness. The model receives only the subset of evidence required for the task. When content is untrusted, it is passed as quoted data rather than as instruction. The browser-agent layer follows a similar pattern: it receives a task-specific action budget and approved tools, then returns observations and proposed actions to the gate. The gate can accept, reject or require clarification before any user-facing answer or state-changing action is completed.

The workflow ends with response assembly and audit logging. Response assembly includes factual checks, citation matching, schema validation and policy checks. Audit logging records the user request, selected plan, data sources, model calls, tool calls, approvals and final result. Logs should avoid storing raw sensitive content where possible; hashed identifiers and structured metadata are often sufficient for later review. The purpose of the log is not surveillance but accountability and debugging.

V. EVALUATION PROTOCOL

Intelligent web applications are often evaluated by task success alone. This is necessary but incomplete. A system that completes a task by exposing unnecessary private data, relying on weak evidence or taking irreversible actions without approval is not dependable. The proposed evaluation protocol therefore combines functional, evidential, security and operational metrics.

Functional evaluation measures whether the application completed the intended task. Web-agent benchmarks show that long-horizon tasks are difficult even when models perform well on isolated prompts [4], [7]. For production systems, task success should be measured across stable regression tasks and dynamic tasks that reflect real interface updates. Evidential evaluation measures whether factual claims are supported by approved sources. The evaluator should check citation coverage, source freshness and whether the answer distinguishes known facts from assumptions.

Security evaluation tests the application against direct and indirect prompt injection, unsafe output handling, excessive agency and sensitive information disclosure. The protocol should include adversarial pages, uploaded documents with hostile instructions, malformed tool outputs and attempts to trigger actions outside the user's authority. Operational evaluation measures latency, cost, fallback behavior and recovery. A useful web system should degrade gracefully when a model is unavailable, a browser action fails or a local device cannot run edge inference.

Table 2: Evaluation dimensions for the proposed framework

Dimension	Example Measure	Acceptance Question
Task success	Completion rate on representative workflows.	Did the application complete the user's intended goal?
Evidence quality	Percentage of factual claims linked to approved sources.	Can the user or reviewer trace important claims to evidence?
Privacy exposure	Sensitive fields sent to remote services per task.	Was only necessary data used, and was local inference considered?
Agency control	Number of state-changing actions requiring approval.	Did the system ask before irreversible or external actions?
Robustness	Performance under changed UI, missing data or malformed content.	Does the application recover without unsafe behavior?
Auditability	Completeness of plan, source and action logs.	Can a reviewer reconstruct what happened after the task?

VI. ILLUSTRATIVE APPLICATION SCENARIOS

The first scenario is an AI help center embedded in a software-as-a-service application. A user asks why an integration failed. The retrieval layer searches product documentation, recent incident reports and the user's non-sensitive configuration metadata. The response gate requires citations for troubleshooting steps and blocks any recommendation that asks the user to reveal secrets. If the user requests an automatic fix, the task is reclassified as action-taking and routed through approval.

The second scenario is an intelligent administrative portal. A manager asks the system to draft a response based on support tickets and account history. The system can use RAG to assemble evidence but must minimize personally identifiable information. The draft is shown with source links and uncertainty notes. Sending the message remains a human-approved action. This scenario demonstrates staged autonomy: the model can prepare work, but the user remains responsible for communication.

The third scenario is a browser-based research assistant for public websites. The agent can browse, summarize and compare sources. Because web pages may contain indirect prompt-injection attempts, page content is treated as untrusted data. The agent cannot access private account tools unless the user explicitly grants a scoped permission. The final answer separates source facts, model synthesis and assumptions. This pattern helps maintain usefulness without assuming that the open web is a trusted instruction channel.

The fourth scenario is a privacy-sensitive form assistant. A local model running in the browser helps explain form fields and draft entries before submission. Remote inference is used only for non-sensitive guidance or when the user consents. Browser-side inference does not remove all risks, but it changes the data-flow boundary and may reduce exposure for routine tasks. This scenario is especially relevant as WebGPU and WebAssembly make client-side model execution more capable [9], [10], [14], [15].

VII. DISCUSSION

The proposed framework emphasizes architecture because many failures of intelligent web applications are integration failures rather than model failures alone. A strong model can still produce unsafe behavior when it receives untrusted content as instruction, receives excessive tool permissions or has no validation layer. Conversely, a smaller model can be useful when the application supplies narrow context, clear task boundaries and strong workflow controls.

There are trade-offs. Provenance tracking and approval gates add engineering effort and may increase latency. Local inference improves privacy in some cases but requires device capability checks, model packaging and fallback plans. Strict tool permissions reduce risk but may lower task success for open-ended workflows. These trade-offs should be visible design decisions. The framework does not claim to eliminate risk; it aims to place risk at explicit control points where it can be measured and improved.

The framework also supports incremental adoption. An organization can begin with citation-aware retrieval for low-risk informational tasks. It can then add local inference for sensitive tasks, browser-agent planning for complex navigation and approval-gated execution for state-changing workflows. This staged path is useful because intelligent web applications often evolve from assistant features into agentic workflows. Without explicit architecture, autonomy can expand faster than governance.

The main limitation of this paper is that it presents a reference architecture and evaluation protocol rather than results from a deployed system. Future work should implement the framework in a controlled web application and compare it against simpler prompt-based and agent-based baselines. Useful experiments would measure task success, privacy exposure, prompt-injection resistance, latency and user trust under identical tasks. Another direction is to develop reusable middleware that enforces provenance labels and approval policies across model providers and browser automation tools.

VIII. CONCLUSION

AI is becoming a central design element in modern web applications. The same capabilities that make applications more helpful also introduce new failure modes around untrusted content, privacy, agency and user overreliance. This paper proposed a Trust-Aware AI Orchestration framework that organizes retrieval, browser agents, edge inference, validation and human approval into explicit layers. The framework gives developers a practical way to build intelligent web applications whose behavior can be traced, constrained and evaluated.

The key conclusion is that intelligent web applications should be treated as socio-technical systems, not just model interfaces. Successful deployment requires evidence management, permission design, security testing, user approval and operational recovery. By evaluating task completion together with provenance, privacy exposure, agency control and auditability, researchers and practitioners can build web applications that are not only intelligent but also dependable.

REFERENCES

[1] P. Lewis et al., Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, Advances in Neural Information Processing Systems, 2020.
[2] I. Gur et al., Understanding HTML with Large Language Models, arXiv:2210.03945, 2022.
[3] X. Deng et al., Mind2Web: Towards a Generalist Agent for the Web, arXiv:2306.06070, 2023.
[4] S. Zhou et al., WebArena: A Realistic Web Environment for Building Autonomous Agents, arXiv:2307.13854, 2023.
[5] I. Gur et al., A Real-World WebAgent with Planning, Long Context Understanding, and Program Synthesis, arXiv:2307.12856, 2023.
[6] K. Greshake et al., Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection, arXiv:2302.12173, 2023.
[7] Y. Pan et al., WebCanvas: Benchmarking Web Agents in Online Environments, arXiv:2406.12373, 2024.
[8] T. Le Sellier De Chezelles et al., The BrowserGym Ecosystem for Web Agent Research, arXiv:2412.05467, 2024.
[9] C. F. Ruan et al., WebLLM: A High-Performance In-Browser LLM Inference Engine, arXiv:2412.15803, 2024.
[10] R. Levine et al., Llamas on the Web: Memory-Efficient, Performance-Portable, and Multi-Precision LLM Inference with WebGPU, arXiv:2605.20706, 2026.
[11] OWASP Foundation, OWASP Top 10 for Large Language Model Applications, OWASP GenAI Security Project, 2025.
[12] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, 2023.
[13] National Institute of Standards and Technology, Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile, NIST AI 600-1, 2024.
[14] World Wide Web Consortium, WebGPU Specification, W3C Candidate Recommendation, 2025.
[15] WebAssembly Community Group, WebAssembly Core Specification, 2026.