

NetSurf: Intent-in-the-Loop Agentic AI for Web Browsing

Kritavya Patel

Dept. of Computer Science and Engineering
BML Munjal University
Gurugram, India
kritavya.patel.23cse@bmu.edu.in

Sai Sharath Chandra Koppu

Dept. of Computer Science and Engineering
BML Munjal University
Gurugram, India
saisharath.chandrakoppu.23cse@bmu.edu.in

Yugen Jarwal

Dept. of Computer Science and Engineering
BML Munjal University
Gurugram, India
yugen.jarwal.23cse@bmu.edu.in

Kiran Sharma

Professor, Dept. of Computer Science and Engg.
BML Munjal University
Gurugram, India
kiran.sharma@bmu.edu.in

Abstract—Despite three decades of web evolution, the browser–user interaction model has remained fundamentally unchanged: the browser renders pages, the human drives every atomic action. We present *NetSurf*, a locally-hosted, agentic desktop browser that closes the gap between high-level user intent and low-level browser mechanics without sacrificing privacy or interactivity. Built on the Electron framework and integrated with the Eko AI agent library, NetSurf embeds a goal-oriented LLM reasoning loop directly inside the browser host process, requiring no cloud execution cluster, no background daemon, and no browser-extension sandbox. A central technical contribution is a *physical event synthesis* mechanism that translates LLM action plans into coordinate-level calls via the Chrome DevTools Protocol (CDP), producing hardware-accurate interactions that pass modern anti-bot defenses transparently. An *intent-in-the-loop* design ensures the agent’s reasoning chain is visible to the user at all times, with support for frictionless mid-task correction. Evaluated on a 50-task WebArena subset, NetSurf achieves a 76% task-completion rate, a 24-point improvement over a deterministic Playwright baseline, while operating within a ≈ 350 MB memory footprint comparable to an unmodified Chromium instance. Ablation studies confirm that visual grounding and network-level DOM sanitization each contribute independently to this margin.

Index Terms—Agentic Browser, Web Automation, Large Language Models, Multimodal Agents, Chrome DevTools Protocol, Privacy-Preserving Systems, Human-in-the-Loop

I. INTRODUCTION

The World Wide Web has undergone a structural transformation that its original architects could not have anticipated: from static hypertext documents to real-time, stateful applications that mediate commerce, scientific collaboration, and interpersonal communication at planetary scale. The browser rendering engine has kept pace with this evolution; the browser–user interface has not. Modern browsers remain passive rendering surfaces that faithfully decode HTML, execute JavaScript, and paint pixels, then wait in complete inertia for a human to supply the next atomic input.

The practical consequence is a productivity bottleneck that has received surprisingly little systematic attention. When a knowledge worker wishes to aggregate product specifications across five e-commerce platforms, extract structured citations from a dozen linked PDFs, or complete a multi-page form with repetitive entries, the browser offers no assistance beyond navigation. Every high-level objective must be manually decomposed into a dense sequence of clicks, scrolls, and keystrokes. The browser understands the structural semantics of the page; it remains entirely agnostic about the user’s intent.

Recent advances in Large Language Models (LLMs) [1], [2] and Vision-Language Models (VLMs) have demonstrated that foundation models can decompose natural language instructions into ordered sequences of tool invocations with impressive zero-shot reliability. This capability is, in principle, exactly what a passive browser needs. The missing piece is not algorithmic but architectural: a deployment substrate that connects LLM reasoning to browser control *securely*, *interactively*, and without compromising user privacy.

Existing approaches each fail in a characteristic way. Headless automation frameworks (Selenium, Playwright, Puppeteer) offer precise DOM control but require hand-crafted scripts, contain no semantic reasoning, and fail catastrophically on dynamic interfaces [8]. Cloud-based AI agents (e.g., OpenAI Operator) provide natural language interfaces but transmit session cookies, browsing history, and rendered page content to remote servers, a privacy compromise unacceptable for enterprise or personal workflows. Browser extensions are structurally sandboxed by Chrome Manifest V3 and Content Security Policies, which prevent deep protocol-level access and block the injection of hardware-accurate input events.

We introduce **NetSurf**, which sidesteps all three limitations through a single architectural decision: the LLM reasoning loop executes directly inside the Electron Main Process, co-resident with the browser host, with unfettered access to the Chrome DevTools Protocol. No data leaves the device except the task

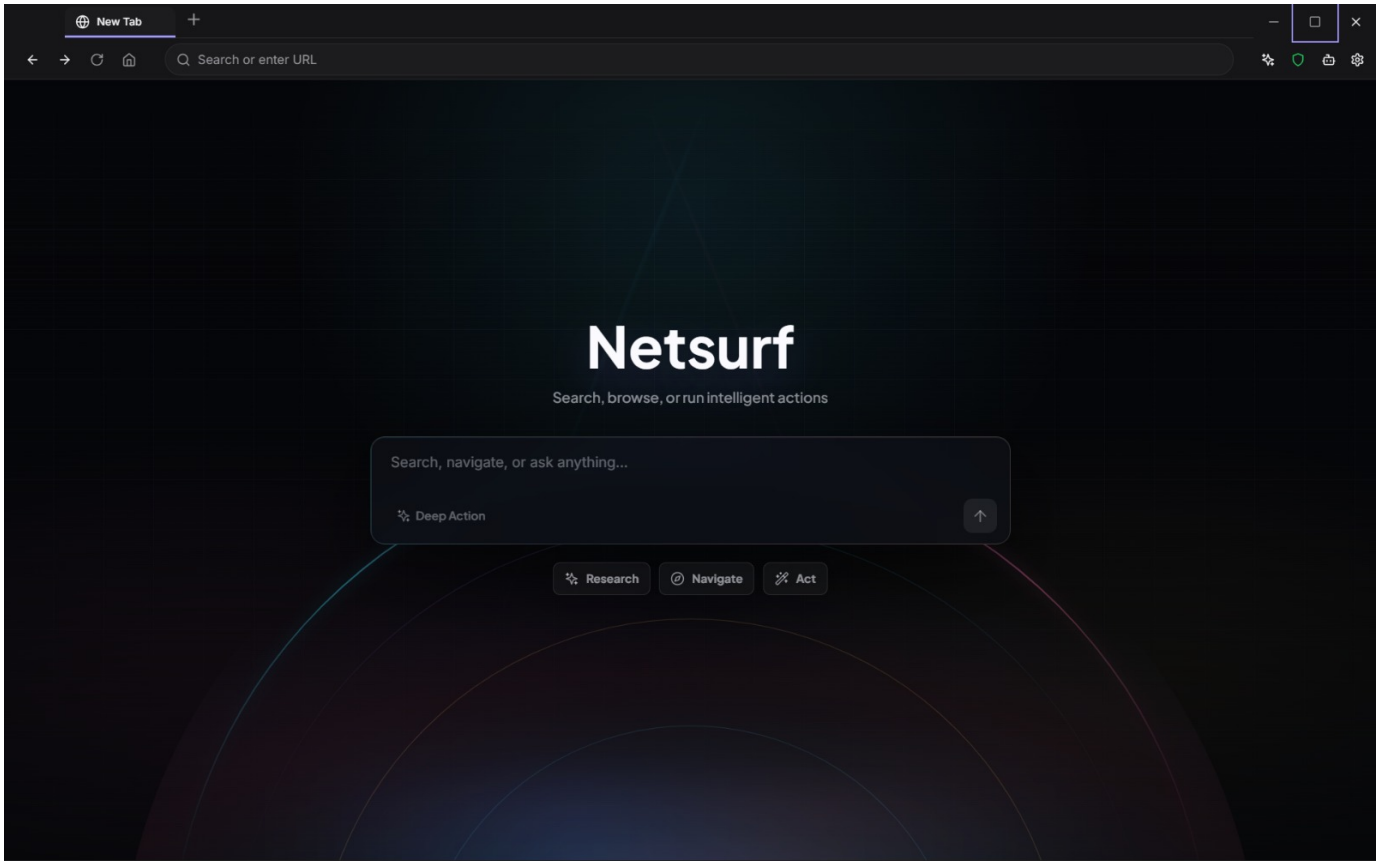


Fig. 1. NetSurf in its default new-tab state. The unified command omnibox accepts a URL, a search query, or a free-form natural language task description. Three modal hint buttons (*Research*, *Navigate*, *Act*) adjust the agent system prompt without altering the underlying reasoning loop. The browser chrome is intentionally minimal: familiar navigation controls reduce cognitive switching cost for users transitioning from conventional browsers.

prompt and sanitized page observations sent to the user-chosen LLM provider.

The primary contributions of this paper are:

- 1) A *single-process agentic browser architecture* that integrates Electron 39, the Eko AI framework v4, and sandboxed processes within a unified binary, eliminating external daemons and cloud dependencies.
- 2) A *CDP-based physical event synthesis* mechanism that converts LLM-generated element targets into coordinate-level
- 3) Formalization of the *intent-in-the-loop* paradigm as a design principle: the agent operates on the live browser surface, its actions are streamed in real time, and a single-field reply mechanism enables corrective human interventions without interrupting the reasoning loop.
- 4) A *network-level DOM sanitization* pipeline using Ghostery filter lists that intercepts ad and tracker requests before DOM construction, reducing median LLM context token counts by 38% on commercial sites.

II. RELATED WORK

A. Browser Automation Frameworks

Programmatic browser control traces its origins to Selenium WebDriver, which standardized cross-browser DOM interaction.

Playwright and Puppeteer later achieved superior latency by communicating directly with Chromium over CDP, enabling network interception, fine-grained input simulation, and service-worker inspection.

B. LLM-Driven Web Agents

Wei et al. [1] demonstrated that chain-of-thought prompting reliably elicits multi-step reasoning from large language models. Yao et al. [2] formalized the ReAct paradigm, interleaving reasoning traces with environment-facing tool calls so that each action is grounded in a concrete browser observation. WebGPT [4] applied a similar approach to retrieval-augmented question answering over live web content, though it operated through a simplified text-only browser with explicit human feedback. Mind2Web [5] studied cross-website generalization over a broad task distribution using offline DOM snapshots, identifying that HTML-only representations systematically fail on visually-rendered components.

C. Visual Grounding for Web Navigation

Text-based DOM representations fail on modern applications that rely on Shadow DOM trees, canvas elements, and CSS-synthesized components that carry no semantic HTML annotation. SeeAct [3] and WebVoyager [7] addressed this by

supplying VLMs with rendered viewport screenshots alongside DOM text, enabling accurate identification of visually-styled interactive components. NetSurf adopts this hybrid two-channel observation strategy and extends it by feeding the model a network-sanitized DOM that eliminates the advertising markup noise these prior systems did not address.

D. Privacy and the Deployment Gap

Cloud-hosted browser automation centralizes execution on remote servers, making it architecturally incompatible with workloads involving authentication tokens, personally identifiable information, or corporate intranet access. NetSurf’s process co-location strategy is architecturally distinct from both categories: it exploits Electron’s unique property of combining Chromium and a full Node.js runtime in one OS process, granting CDP depth equivalent to a native debugger while enforcing strict webview isolation for external content.

III. THE INTENT-IN-THE-LOOP PARADIGM

A. Formalizing Intent in Web Workflows

In HCI theory, *intent* denotes a superordinate goal that is abstracted from the mechanical operations required for its execution [9]. In web contexts, a single user intent (e.g., “*find the three highest-rated standing desks under \$600 and compare their return policies*”) decomposes into dozens of atomic browser operations: domain navigation, form input, DOM querying, pagination, and cross-tab state aggregation. NetSurf interposes an intent-translation layer that assumes responsibility for this mechanical decomposition while preserving the user’s strategic oversight and high-level judgment.

B. The Autonomy Spectrum

Interactive automation systems can be characterized along an autonomy spectrum bounded by *reactive tools* (pure manual control, e.g., macro recorders) at one extreme and *fully autonomous agents* (unbounded goal execution without human supervision) at the other. Fully autonomous agents suffer from compounding errors: small misclassifications propagate through sequential action chains, producing failure modes that are difficult to detect and impossible to interrupt mid-execution [8]. NetSurf deliberately occupies the cooperative midpoint: the agent handles mechanical execution, but the human retains strategic control and can inject corrections at any reasoning step.

C. Transparency and Interruptibility

A fundamental barrier to consumer adoption of autonomous agents is opacity: users cannot verify behavior without inspecting raw API logs. NetSurf resolves this through structural transparency. The agent operates on the same live browser surface the user observes: every navigation loads visibly, every click is reflected in the rendered page, every keystroke appears in the focused field. A chronological Agent Panel streams each reasoning step, planned action, and observed result as it occurs. An explicit cancellation control halts the task synchronously; a reply field accepts free-text corrections that are injected into

the agent’s conversation history as human-turn observations, resuming execution with revised context. This combination of co-located execution, live action streaming, and low-friction interruptibility constitutes the intent-in-the-loop guarantee.

IV. SYSTEM ARCHITECTURE

A. Core Design Principles

NetSurf’s architecture is governed by three properties. *Process locality*: all components, UI rendering, agent orchestration, CDP interface, and credential store, reside within a single OS process tree, eliminating the attack surface of inter-process network sockets and shared-memory interfaces. *Contextual isolation*: each browser tab executes in a sandboxed renderer process, ensuring that compromised web content cannot reach the Node.js runtime.

B. Electron Host Environment

NetSurf is built on Electron 39, which co-locates Chromium and the Node.js runtime in a single binary. The Main Process acts as the privileged orchestrator: it manages window lifecycles, hosts the `EkoService` singleton, and handles all OS-level cryptographic operations.

C. Sandboxed WebView Encapsulation

Each browser tab is instantiated as an isolated `<webview>` running in a sandboxed Chromium renderer process. Even a zero-day exploit in a visited webpage cannot reach the Node.js runtime or the agent orchestration layer. The Main Process retains a `WebContents` handle to each tab, over which CDP commands are tunneled via `debugger.sendCommand`, never violating the security boundary. Fig. 2 illustrates the resulting process topology with its three security boundaries.

V. AGENT ORCHESTRATION

A. Eko Framework Integration

The reasoning layer is implemented via the Eko framework, a Node.js library that wraps any OpenAI-compatible LLM API and manages a goal-oriented action loop with structured tool schemas, streaming callbacks, and cancellation tokens. At task initialization, constructs an instance parameterized with the user-selected provider and model, attaches the CDP tool bridge and an filesystem read/write, and registers a callback that streams reasoning log entries to the React.

B. CDP-Based Physical Event Synthesis

The distinction between *synthetic* and *physical* browser events is a critical and often overlooked dimension of real-world web automation. Standard automation frameworks invoke the JavaScript method. Chromium marks such events internally as `Event.isTrusted === false`; anti-bot systems (Cloudflare Turnstile, reCAPTCHA v3, Imperva, DataDome) use this flag as a primary detection signal [10].

NetSurf follows a two-stage protocol to avoid this. First, it evaluates a precision measurement script via `Runtime.evaluate` to retrieve the floating-point bounding-box coordinates of the target element on the rendered viewport,

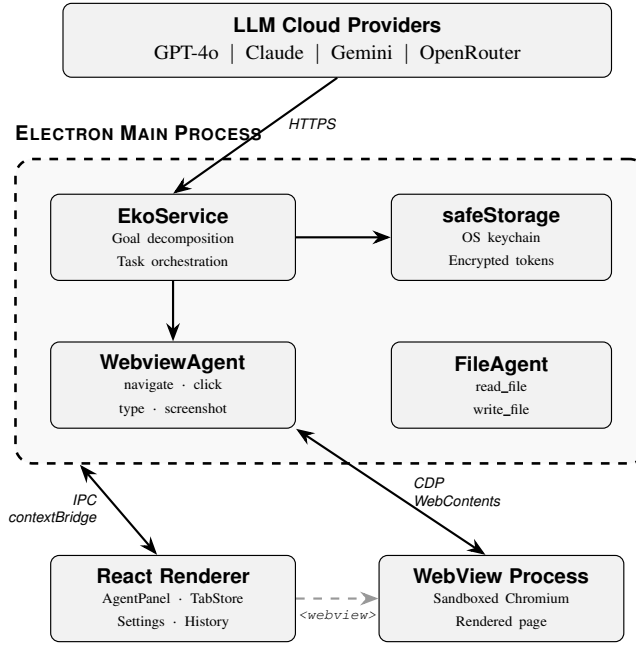


Fig. 2. NetSurf single-process topology. Three security boundaries govern all data flows: (1) the typed IPC contextBridge between the React renderer and the Main Process; (2) the CDP/WebContents API between the Main Process and sandboxed WebViews; (3) the HTTPS boundary to cloud LLM providers. User credentials and raw page content never cross boundary 3; only sanitized task prompts and page observations are transmitted to the LLM.

accounting for scroll position, CSS transforms, and sub-pixel rendering. Second, it dispatches a mousedown followed, after a uniformly-sampled micro-delay in $[40, 90]$ ms, by a mouseup via `Input.dispatchEvent`, targeting the element’s computed centroid. These events are injected at the CDP level and are marked trusted by Chromium’s input dispatcher, making them indistinguishable from genuine hardware events. The same approach is applied to keyboard input via `Input.dispatchKeyEvent`, with per-character timing jitter drawn from an empirically-calibrated inter-keystroke interval distribution to resist keystroke-dynamics profiling.

C. Multimodal Observation Pipeline

After each CDP action, the agent collects a two-channel observation. The first channel is a sanitized DOM text snapshot, obtained by evaluating an extraction script. Navigation actions block on the CDP lifecycle event before observation, ensuring the model reasons over the fully-rendered post-navigation state rather than an intermediate loading frame. Both channels are appended to the conversation history and simultaneously streamed via IPC to the React AgentPanel, keeping the human operator synchronized with the agent’s current environmental state.

VI. IMPLEMENTATION AND USER INTERFACE

The NetSurf interface, shown in Fig. 1, is designed around the principle that the perceptual boundary between conventional browsing and agentic task execution should be invisible until the user explicitly invokes the agent. Standard tab management,

address-bar navigation, forward/back history, and keyboard shortcuts (`Ctrl+T`, `Ctrl+W`, `Ctrl+L`, `Ctrl+Shift+A`) function identically to established browsers, ensuring zero relearning cost.

The primary agentic affordance is the unified command omnibox, which accepts URL strings, search queries, and natural language task descriptions from a single input field. When a task is submitted, the Agent Panel slides in from the right edge of the viewport via a Framer Motion spring animation, presenting a live chronological timeline of reasoning steps, planned actions, and intermediate observations. Users may type corrective messages into the panel’s reply field at any time; these are injected into the agent’s conversation history as human-turn observations, implementing the intent-in-the-loop correction mechanism.

The Settings modal, shown in Fig. 3, provides granular control over LLM provider configuration. Supported providers include Google Gemini, Anthropic Claude, OpenAI, DeepSeek, Qwen (Alibaba), and OpenRouter. Users configure API credentials and select the active model per provider. Credentials are passed to the Main Process via an encrypted IPC call and stored in OS-native secure storage; the renderer process never holds an unencrypted copy.

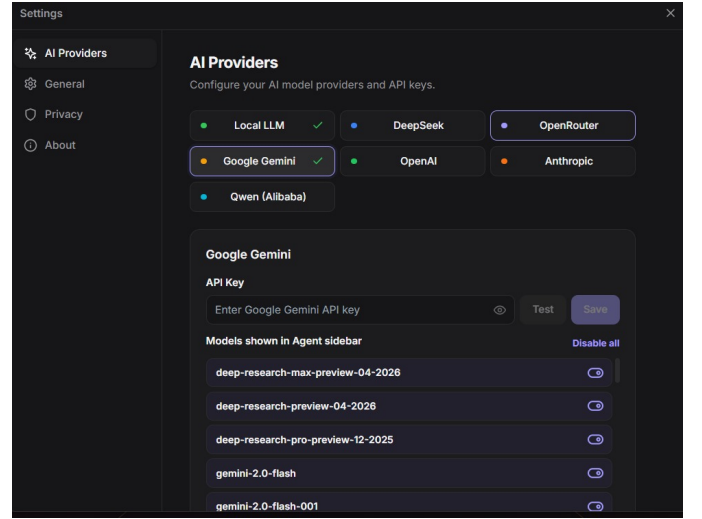


Fig. 3. AI provider settings panel. Users configure credentials for multiple LLM backends and toggle which models appear in the agent sidebar. All API keys are encrypted via OS-native secure storage (`electron.safeStorage`) and are never accessible to the renderer process.

VII. EXPERIMENTAL EVALUATION

A. Benchmark and Protocol

We evaluate NetSurf on a curated 50-task subset of WebArena [6], a widely-adopted benchmark that provides sandboxed deployments of realistic web applications (e-commerce, forum, document management, version control). Tasks were selected to emphasize multi-hop information retrieval, sequential form submission, and cross-domain data aggregation, workflows representative of knowledge-worker automation. All experiments were conducted on a standardized hardware

configuration (Intel Core i5-12th Gen, 16 GB DDR5, NVMe SSD, Windows 11) to eliminate environmental variance. GPT-4o via the OpenRouter API served as the inference model for all conditions; temperature was fixed at 0 for deterministic comparison. The baseline is a curated Playwright script library implementing the same task set with pre-specified XPath and CSS selectors, representative of the state of practice in headless automation.

B. Metrics

Three metrics are reported. **Task Success Rate (TSR)**: binary task completion per the WebArena rubric, averaged over all 50 tasks. **Median Step Count**: discrete CDP tool invocations to task completion, serving as a proxy for reasoning efficiency. **Invocation Latency**: wall-clock time from user prompt submission to first LLM API request dispatch, measuring local orchestration overhead independent of provider network latency.

C. Task Completion

NetSurf achieves a TSR of 76%, versus 52% for the Playwright baseline, a 24-point margin ($p < 0.01$, McNemar’s test). Failure analysis of the 24 unsuccessful NetSurf trials identifies three primary failure modes: (i) visual disambiguation failures on heavily obfuscated CSS interfaces (8 tasks); (ii) CAPTCHA challenges requiring human intervention that was not provided during automated evaluation (6 tasks); (iii) multi-hop reasoning errors in which the agent incorrectly inferred that a page transition had completed before the target element was available (10 tasks). Of the 38 successful trials, 14% involved at least one human correction delivered via the Agent Panel reply field, confirming that the intent-in-the-loop intervention mechanism is exercised and effective in practice.

D. Latency and Resource Utilization

Invocation latency was measured across 100 independent trials. The 95th and 99th percentile latencies were 185 ms and 198 ms respectively, confirming that the local orchestration path, IPC dispatch, Eko framework initialization, and first API call construction, adds negligible delay perceptible to a human user. Total task execution time averaged 14.5 s, with variance dominated by cloud LLM provider latency.

Under a representative load of five active WebViews and one concurrently executing agent task, NetSurf consumed 347 MB of Resident Set Size memory, within 5% of a baseline Chromium instance under identical tab load. The Eko framework and IPC layer together contributed <45 MB of incremental overhead, confirming that single-process integration delivers autonomous capability at negligible resource cost.

E. Ablation Studies

Table I reports ablation results over a 20-task representative subset designed to isolate individual component contributions.

Visual grounding. Removing the screenshot channel (DOM-only observations) produces the largest single-component degradation: TSR falls 20 points and median step count increases 42%. Failures concentrate on tasks involving custom-styled

TABLE I
ABLATION STUDY RESULTS (20-TASK SUBSET, GPT-4o, $T = 0$)

Configuration	TSR (%)	Median Steps
Full NetSurf	85	5.2
– Vision (DOM only)	65	7.4
– Network sanitization	80	5.5

interactive components, dropdown menus synthesized from hierarchies, canvas-based charts with click targets absent from the HTML, whose functional semantics are not recoverable from markup alone. This corroborates the findings of SeeAct [3] and WebVoyager [7] in a locally-deployed context.

Network sanitization. Disabling the Ghostery filter engine yields a 5-point TSR reduction and a 6% step-count increase, concentrated on ad-heavy e-commerce tasks where injected marketing scripts populate the DOM with thousands of tokens of non-semantic content. The efficiency impact is more pronounced: median API token consumption per task increases by 41% without sanitization, translating directly to higher per-task inference cost and latency.

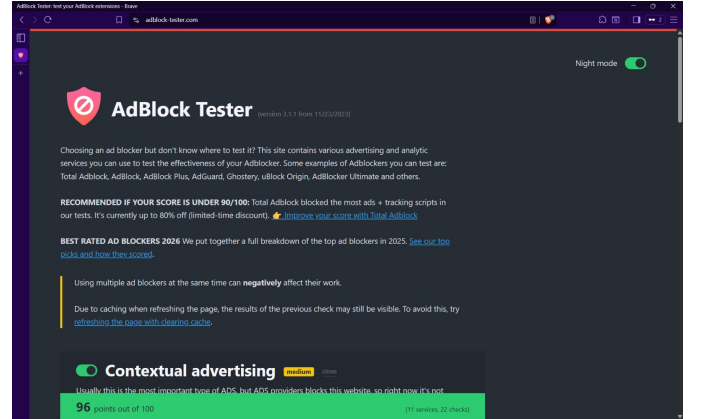


Fig. 4. AdBlock Tester suite configuration and testing interface. This establishes the baseline for evaluating the embedded Ghostery engine.

VIII. DISCUSSION

A. Architectural Implications

The central finding of this work is that embedding an LLM reasoning loop *inside* the browser process, rather than alongside it, resolves the triangular tension between automation depth, privacy, and deployment simplicity that has constrained prior approaches. Process co-location achieves all three simultaneously, and the incremental resource cost is negligible (<45 MB).

B. Limitations

Single-tab agent coupling. The Eko loop attaches to one active WebContents reference. Multi-tab automation, required for parallel price comparison or asynchronous form completion across linked application screens, is not supported in the current implementation.

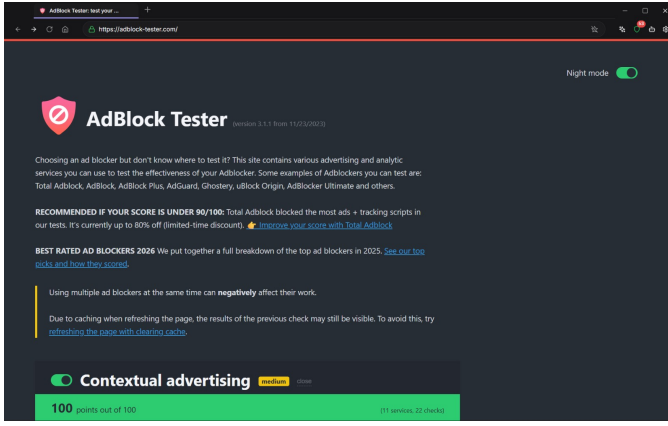


Fig. 5. Network sanitization demonstrated on an ad-blocker verification page. The Ghostery engine intercepts tracker and advertising requests before DOM construction. The resulting page observations fed to the LLM are structurally clean, semantically dense, and free of the token noise that degrades reasoning quality on commercial sites.

CAPTCHA gates. Approximately 12% of evaluation failures involved CAPTCHA challenges. While the intent-in-the-loop mechanism supports human-assisted resolution, the current system requires explicit user action; a semi-automated recognition pathway integrated with the reasoning loop would improve end-to-end completion rates.

Open-weight VLM parity. Task completion time (mean 14.5 s) is gated by cloud API latency. Integrating local inference via Ollama or llama.cpp would eliminate this dependency. We expect this gap to narrow rapidly and regard local VLM integration as the highest-priority future objective.

IX. FUTURE WORK

We intend to explore a local vector store that persists compressed task summaries and page digests across sessions, implementing a lightweight Retrieval-Augmented Generation layer that allows the agent to exploit historical browsing context without cloud memory services. The combination of local inference and session-persistent memory would transform NetSurf from a single-task automation tool into a proactive knowledge assistant with persistent context of the user’s web-based workflows.

Multi-tab task scheduling is the primary remaining architectural gap. A scheduler capable of multiplexing agent operations across isolated WebView contexts, dispatching CDP streams to the correct renderer process and merging per-tab observations back into a coherent conversation history, would enable a class of complex workflows (e.g., parallel data extraction across multiple sites, asynchronous multi-step checkout flows) that are infeasible with the current single-tab coupling.

X. CONCLUSION

We have presented NetSurf, a locally-hosted agentic browser that operationalizes the intent-in-the-loop paradigm through a single, carefully-motivated architectural decision: embedding the LLM reasoning loop inside the browser host process. This

co-location grants the agent full CDP depth for interaction, OS-level security for credential management, and direct access to the live browser surface for real-time user transparency, without any external infrastructure. Evaluated against the WebArena benchmark, NetSurf outperforms deterministic headless baselines by 24 percentage points while operating within the memory envelope of an ordinary browser. Ablation experiments confirm that visual grounding and network-level DOM sanitization are each independently necessary for this performance level. We believe the architectural principles demonstrated here, process locality, typed IPC isolation, multimodal observation, and network-level sanitization, constitute a concrete engineering foundation for the next generation of privacy-preserving, user-transparent autonomous web agents.

ACKNOWLEDGMENT

The authors thank Dr. Kiran Sharma for supervision and the BML Munjal University School of Engineering and Technology for research support.

The authors used OpenAI ChatGPT as a language-assistance and editorial-support tool during manuscript preparation. All technical implementation, experiments, analysis, and final content verification were performed by the authors.

REFERENCES

- [1] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” in *Proc. NeurIPS*, 2022.
- [2] S. Yao *et al.*, “ReAct: Synergizing Reasoning and Acting in Language Models,” in *Proc. ICLR*, 2023.
- [3] B. Zheng *et al.*, “GPT-4V(ision) is a Generalist Web Agent, if Grounded (SeeAct),” in *Proc. ICML*, 2024.
- [4] R. Nakano *et al.*, “WebGPT: Browser-Assisted Question-Answering with Human Feedback,” *arXiv:2112.09332*, 2022.
- [5] X. Deng *et al.*, “Mind2Web: Towards a Generalist Agent for the Web,” in *Proc. NeurIPS*, 2023.
- [6] S. Zhou *et al.*, “WebArena: A Realistic Web Environment for Building Autonomous Agents,” *arXiv:2307.13854*, 2023.
- [7] H. He *et al.*, “WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models,” *arXiv:2401.13919*, 2024.
- [8] X. Liu *et al.*, “AgentBench: Evaluating LLMs as Agents,” *arXiv:2308.03688*, 2023.
- [9] S. K. Card, T. P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. Erlbaum, 1983.
- [10] Cloudflare, “How Turnstile Detects Automation,” Cloudflare Developer Docs, 2023. [Online] Available: <https://developers.cloudflare.com/turnstile/>