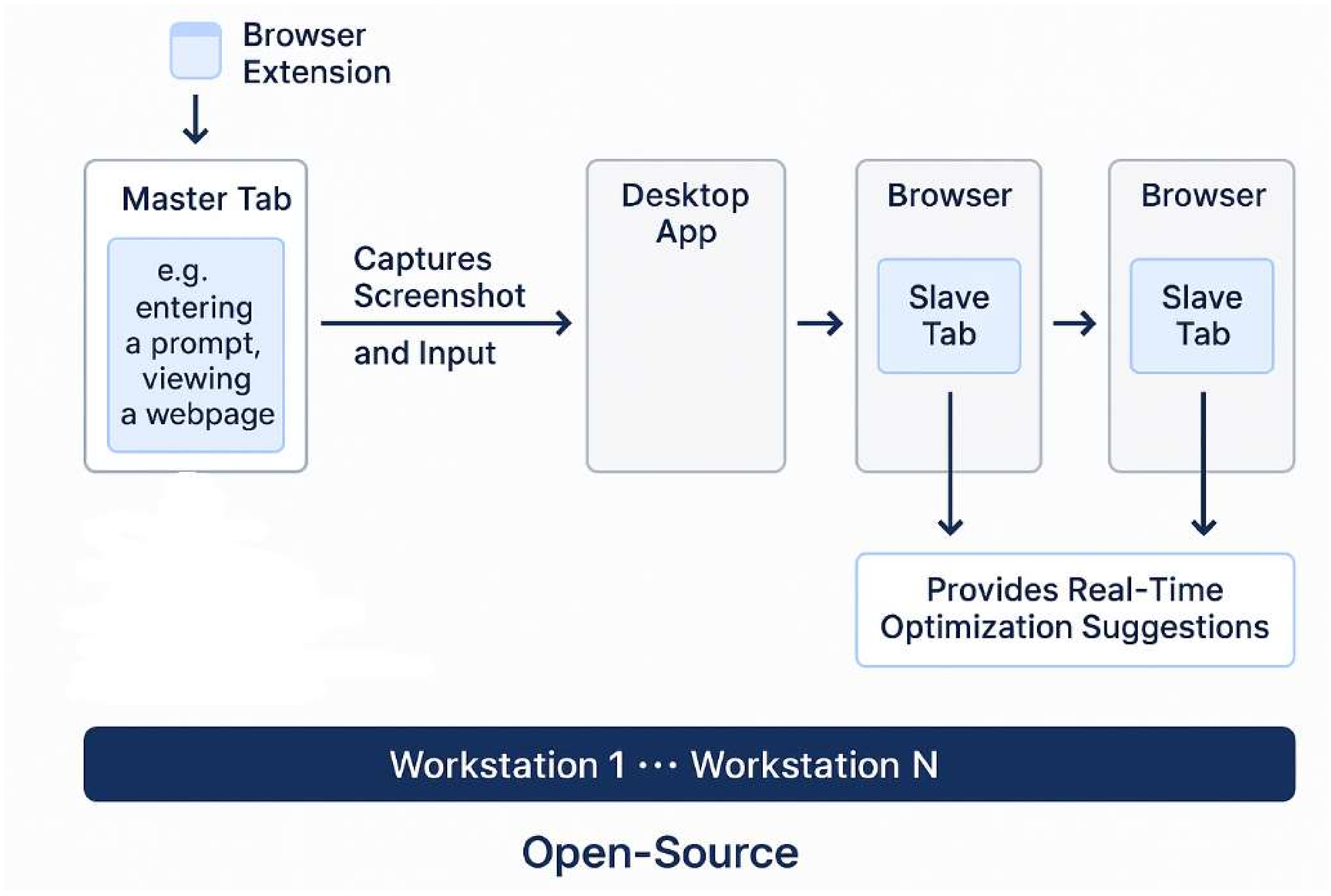
An Open-Source Browser-Based Master–Slave AI Agent Framework for Workspace Orchestration


Browser Extension
Master Tab
e.g. entering a prompt, viewing a webpage
Captures Screenshot
and Input
Desktop App
Browser
Slave Tab
Browser
Slave Tab
Provides Real-Time Optimization Suggestions
Workstation 1 ··· Workstation N
Open-Source

Paper by

optimando.ai

Abstract: We introduce a novel open-source framework, developed and published by **Oscar Schreyer and the AI consultation and automation integration company optimando.ai**, that transforms individual browser tabs into collaborative AI agents, enabling real-time orchestration and optimization of complex digital AI workspaces. The open-source system from **optimando.ai** helps users improve their digital work in real time by using a main browser tab (the master) for user activity and connecting it with AI-powered helper tabs (slaves) that analyze screenshots and inputs to give smart suggestions based on the user's goals; for example, if someone is building an IT job board and watches a video about n8n, the system can automatically suggest better GDPR-compliant workflows by using the business context and showing the results as text, visuals, or audio; the master tab can also be replaced with a camera input in future setups .In this system, a lightweight browser extension registers each tab with a unique ID and links it to a local orchestrator app. Users designate any tab as a master (primary interface) or slave (auxiliary agent) via the extension. Master tabs handle user interactions (e.g. with chatbots like ChatGPT, Claude, or Google Gemini, or browser apps and websites such as Google Sheets, YouTube, Amazon, email, news portals, or any other web-based service), while helper tabs each run a distinct LLM chatbot (local or cloud) to perform supporting tasks. The extension captures only the browser’s visible content (screenshots and text inputs) from the master tab and routes it through the orchestrator to selected slave agent tabs. Slave agents can then do prompt optimization, tree-of-thought expansion, data retrieval, context augmentation, and more. All agents operate under “man-in-the-loop” control: the user supervises and approves outputs before use. Crucially, this system is open-source and browser-only, ensuring transparency, user control, and data sovereignty. We distinguish our approach from single-agent AI browsers (e.g. Opera Neon or Google’s Project Mariner) by enabling true multi-agent coordination across tabs, extensible to multi-monitor and team scenarios. We detail the system architecture, use cases, and ethical considerations, and outline the framework’s unique contributions and open-source vision.

Introduction Modern knowledge work often involves juggling dozens of browser tabs and applications. Switching between them incurs cognitive overhead and lost time: one study found users jump between apps and browser tabs ~1,200 times per day, wasting up to 4 hours per week just reorienting. Agentic AI promises to reduce this task-switching by acting as a “glue” between apps. For example, users could instruct an AI to gather information or complete tasks spanning multiple websites without manually navigating them. Recent efforts like OpenAI’s Operator, Google’s Project Mariner, and Opera’s agentic browser demonstrate that AI can autonomously browse and act on the web for users. However, these systems typically deploy single agent models embedded in a browser or cloud service.

By contrast, the system developed by **optimando.ai** is designed to support real-time activity optimization based on predefined global context-driven goals. It operates as a backend suggestion engine that continuously monitors in-browser user activity and delivers context-aware prompts or insights aligned with a user’s objective. This turns the browser into a live optimization surface—something no single-agent system currently enables.

Related Work AI Agents and Browsers. Contemporary AI agent tools typically focus on one primary agent that operates in a browser or cloud environment. For example, Google’s Project Mariner (2025) is an experimental browser agent: users chat with the agent and it can autonomously visit sites to complete tasks (e.g. ticket purchase). Mariner’s updated version runs in the cloud so it can handle multiple tasks concurrently, but it remains a single agent that users cannot directly interrupt or extend. Similarly, OpenAI’s Operator (also known as “Computer-Using Agent”) uses GPT-4o with vision to see and click on webpages. Operator can run parallel tasks through separate chat threads, but again it is a singular agent per conversation.

Browsers themselves are beginning to integrate AI agents. Opera Neon (2025) is billed as the first “agentic browser”: it embeds a native AI that can chat with the user and a separate “Browser Operator” that automates web tasks (forms, shopping, etc.). Opera also demonstrates a more ambitious “AI engine” that, in the cloud, can work on user-specified projects offline and do multitasking in parallel. However, Opera’s agents are proprietary, deeply integrated into one browser, and not open for user modification. Opera One (a related product) has introduced AI Tab Commands: a feature where a built-in assistant can group or close tabs on command (e.g. “group all tabs about ancient Rome”). This helps manage tab clutter, but it still uses a single AI interface per browser to organize tabs, without supporting multiple cooperating agents.

Outside the browser domain, research on LLM-based Multi-Agent Systems (MAS) is rapidly growing. Tran et al. (2025) survey LLM-MAS and note that groups of LLM-based agents can “coordinate and solve complex tasks collectively at scale”. Emerging orchestration frameworks (e.g. AWS Bedrock’s multi-agent service or

Microsoft's AI Foundry) allow specialized agents to collaborate under a supervisor, and enterprises are experimenting with central "Agent OS" platforms that integrate many agents. But these systems operate at the level of backend services or applications, not at the level of coordinating a user's browser environment. Crucially, we found no published work on orchestrating multiple AI agents distributed across browser tabs as a unified workspace.

Privacy and Control. As AI agents gain browser abilities, privacy and user control have become key concerns. Opera emphasizes that Neon's automation runs locally in the browser to "preserve users' privacy and security", and OpenAI's Operator is designed to let users takeover control for sensitive inputs (takeover mode) and not collect screenshots of private data. Our framework similarly restricts observation to in-browser content, and keeps all orchestration local. Moreover, by being fully open-source, the system ensures transparency: users can audit how agents process data and even opt to self-host any component. This open-source approach aligns with calls for trustworthy AI: licensed under Apache/MIT-like terms, the software grants users the freedom to inspect, modify, and control every aspect.

In summary, while there is related work on AI agents in browsers and on multi-agent AI systems, our framework uniquely combines both: we enable many specialized AI agents (each a distinct LLM) to operate collaboratively across the user's browser workspace. This open-source, browser-based multi-agent architecture has not been documented before in academic or industry literature.

Master/Slave Roles and Feedback Loop Users can tag tabs as master (leading tasks, direct user focus) or slave (dedicated to background sub-tasks). These background tasks run automatically, triggered by activity in the master tab. Masters are often user-facing tasks (e.g. a chatbot or any browser-based application). Slaves provide assistance: for example, while a user debates a business strategy in a chat (master), one slave agent might track the user’s goal progression, another might fetch relevant statistics, and a third might generate a relevant infographic.

For example, while a user chats online in a forum, email client, or customer service portal (master tab), one slave agent might suggest clearer phrasing, another could optimize tone and politeness based on the communication context, and a third might check spelling, consistency, or alignment with previous messages. These agents operate silently in the background, analyzing the content in real time and providing actionable suggestions as the user types. This transforms the browser into a live optimization environment—supporting users like a virtual writing coach that continuously enhances the quality and impact of their digital communication.

Even more broadly, if a user watches a YouTube video (master tab)—for example, a product walkthrough of a new AI tool—slave agents can assist by cross-referencing this information with the user's current goals or projects. One agent might highlight how the showcased tool could be integrated into the user’s existing tech stack. Another might suggest more efficient or better-suited alternatives based on predefined system constraints or preferences. A third agent could retrieve relevant use cases or success stories, helping the user assess practical value. Together, they act as a live research and recommendation engine—turning passive content consumption into actionable insights aligned with the user’s broader objectives.

The user sees slave outputs as suggestions or context inserts in real time. This creates an optimization loop: the user steers the main task, slaves continuously augment it, and the user approves or refines the results. The human stays “in the loop” at every step, aligning with best practices in trustworthy AI.

Technical and Ethical Considerations The extension only captures the browser’s visible content (screen images and user-entered text), not the full desktop. This guarantees all agent inputs are limited to in-browser context, protecting unrelated private data. Captured content is sent through the orchestrator to slave agents. No data is ever sent to outbound servers or external services unless explicitly configured by the user. Being fully open source, the software’s code is transparent and available for public review, ensuring accountability and trust. The connected LLMs may communicate externally (e.g., GPT-4 via API), but this behavior is entirely under user control. Alternatively, local LLMs can be used based on the user’s preferences or requirements.

Feature Comparison The following table presents a verified comparison of the proposed framework by **optimando.ai** with major agentic browser solutions available as of 2025. The focus is on user autonomy, open architecture, real-time assistance, and extensibility in practical work environments.

Feature Comparison

Feature	Proposed Framework	Opera Neon	Google Project Mariner
Realtime backend optimization suggester	Yes (context-aware AI suggestions tied to global user goals)	No	No
Multi-agent coordination	Yes (multiple slaves per tab)	Partial	Partial
Browser tab as agent	Yes (users tag tabs)	No	No
Open-source	Yes (user-run)	No	No
User-controlled LLM selection	Yes (any open or cloud LLM)	No	No
Data sovereignty	High (all local, opt-in cloud)	Medium	Low
Man-in-loop by design	Yes	Yes	Yes
Unique agent IDs (multi-user)	Yes	No	No
Multitasking / parallel tasks	Yes (multiple slaves)	Yes	Yes

Evaluation and Distinction:

While many browser-based AI solutions focus on embedded or single-agent automation, the proposed framework by optimando.ai is the first to offer an open-source, tab-based multi-agent system with full user control, local execution, and real-time feedback loops. The ability to assign agents per tab and run them in parallel—while respecting privacy boundaries and project-specific global goals—sets this architecture apart. Most critically, the platform's role as a continuous backend optimization suggester is not replicated by any known system, enabling live augmentation of user workflows across content types, tools, and teams.