AgentClick: A Skill-Based Human-in-the-Loop Review Layer for Terminal AI Agents

Haomin Zhuang*
University of Notre Dame
South Bend, USA
hzhuang2@nd.edu

Hanwen Xing*
University of Southern California
Los Angeles, USA
harvenx01@gmail.com

Xiangliang Zhang
University of Notre Dame
South Bend, USA
xzhang33@nd.edu

Abstract

Recent autonomous AI agents such as Codex, and Claude Code have made it increasingly practical for users to delegate complex tasks, including writing emails, executing code, issuing shell commands, and carrying out multi-step plans. However, despite these capabilities, human–agent interaction still largely happens through terminal interfaces or remote text-based channels such as Discord. These interaction modes are often inefficient and unfriendly: long text outputs are difficult to read and review, proposed actions lack clear structure and visual context, and users must express feedback by typing detailed corrections, which is cumbersome and often discourages effective collaboration. As a result, non-expert users in particular face a high barrier to working productively with agents.

To address this gap, we present **AgentClick**, an interactive review layer for terminal-based agents. AgentClick is implemented as a localhost npm server paired with a skill-based plugin that connects the running agent to a browser interface, allowing users to supervise and collaborate with agents through a structured web UI rather than raw terminal text alone. The system supports a range of human-in-the-loop workflows, including email drafting and revision, plan review and modification, memory management, trajectory inspection and visualization, and error localization during agent execution. It also turns code generation and execution into a reviewable process, enabling users to inspect and intervene before consequential actions are taken. In addition, AgentClick supports persistent preference capture through editable memory and remote access over HTTP, allowing users to review agents running on servers from their personal devices. Our goal is to lower the barrier for non-expert users and improve the efficiency and quality of human–agent co-work. The code is available at https://github.com/agentlayer-io/AgentClick/tree/main.

ACM Reference Format:

Haomin Zhuang, Hanwen Xing, and Xiangliang Zhang. 2026. AgentClick: A Skill-Based Human-in-the-Loop Review Layer for Terminal AI Agents. In *ACM Conference on AI and Agentic Systems (CAIS '26), May 26–29, 2026, San Jose, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3786335.3813232

*Equal Contribution

This work is licensed under a Creative Commons Attribution 4.0 International License.
CAIS '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2415-2/26/05
https://doi.org/10.1145/3786335.3813232

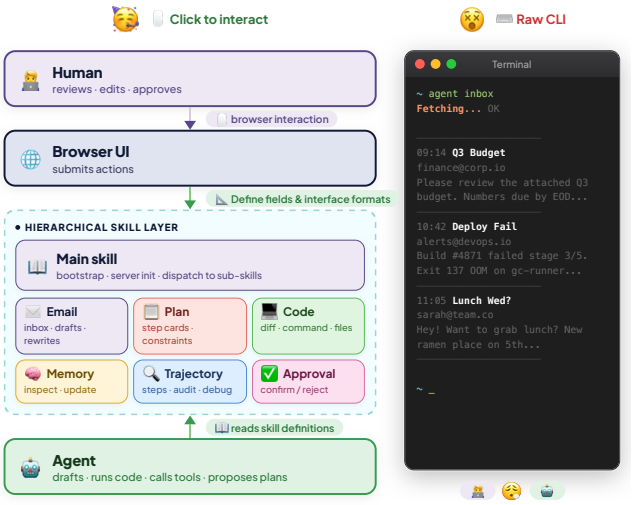


Figure 1: Demo of email review via AgentClick vs. terminal

Keywords: human-in-the-loop, AI agents, agent skills

1 Introduction

Agentic AI systems are increasingly capable of carrying out long-horizon tasks such as triaging inboxes, executing code pipelines, and orchestrating multi-step workflows [4, 5]. Recent systems such as Codex [8], OpenClaw [10], and Claude Code [1] make it practical for users to delegate increasingly consequential work to agents, including writing emails, planning actions, editing files, running shell commands, and interacting with external tools. This shift creates new opportunities for productivity, but also raises a central interaction challenge: how can humans effectively collaborate with autonomous agents while retaining meaningful visibility, control, and opportunities for correction [7]? At the current stage, interaction with such agents still happens primarily through terminal interfaces or remote text-based channels such as Discord. These modes are poorly suited to human–agent co-work for three reasons. First, terminal output interleaves reasoning traces, tool logs, and proposed actions in a single stream, making it difficult to identify what requires review. Second, feedback is cumbersome: users must type free-form corrections even for small local edits. Last, consequential actions—sending email, executing code, committing to a plan—are presented as opaque events or binary prompts, leaving little room for targeted inspection or modification. These barriers are especially pronounced for non-expert users, and become more acute when agents run on remote or headless infrastructure, where

the terminal is not only a poor collaboration medium but often an inaccessible one. We present **AgentClick**, an interactive review layer for terminal-based agents that improves human–agent collaboration rather than merely gating execution. AgentClick runs as a localhost npm server paired with a skill-based plugin, exposing a browser-based interface in place of raw terminal text. It surfaces structured review UIs tailored to different action types: email drafting, plan editing, memory management, trajectory visualization, and code inspection. Users interact at the level of the artifact being produced rather than at the level of terminal output. AgentClick also supports persistent preference capture, so user edits feed into agent memory for future sessions, and serves its interface over HTTP for remote access from personal devices. Figure 1 contrasts a terminal-only session with the same session in AgentClick's Email Review UI.

2 System Design

2.1 Human–Browser–Backend–Agent Architecture

AgentClick is built around four components: *human*, *browser*, *backend*, and *agent*. The *agent* executes tasks autonomously—drafting emails, proposing plans, running code, and calling tools—but rather than acting immediately, it submits proposals to the backend and waits for a review outcome. The *backend* acts as the coordination layer: it receives proposals from the agent, maintains session state and user preferences, serves the browser interface over HTTP, records user edits, and returns approval or correction signals to the agent. The *browser* exposes task-specific views tailored to different action types—email review, plan editing, memory inspection, trajectory visualization, and code review—giving the user structured access to agent proposals at the artifact level rather than the terminal level. The *human* reviews, edits, and approves these proposals through the browser, with corrections optionally persisted to agent memory for future sessions.

2.2 Interactive Review Flow

When the *agent* reaches a consequential step, it submits a proposal to the *backend*, which creates a review session and surfaces it in the *browser* UI. Depending on the artifact type, the *human* user may approve, directly edit, delete content, adjust constraints, or request a targeted rewrite; the result is returned to the agent, which incorporates the feedback and continues. This goes beyond binary accept-or-reject: users can intervene at the level of the artifact, which is particularly useful when output is mostly correct but requires localized changes. The same flow supports monitoring and debugging. Because intermediate artifacts are surfaced before final execution, users can track agent progress, locate errors early, and intervene before a problematic action is committed.

2.3 Hierarchical Skill-Based Integration

AgentClick connects the backend and the agent through a hierarchical skill design. Rather than requiring a dedicated SDK or direct code-level integration, the agent consumes a structured set of Markdown skills that define how to launch AgentClick, communicate with the backend, and invoke task-specific review workflows. This approach has two key advantages. First, skills are lightweight: they require no instrumentation of the agent codebase and add no runtime dependencies. Second, they are portable: any agent that can read and follow Markdown instructions—such as Claude Code or OpenClaw—can adopt AgentClick without modification. In this sense, skills act as the glue layer between agent execution and browser-based human review [12].

At the top level, a *main skill* handles shared integration logic and serves as both a bootstrap mechanism and a dispatcher. It instructs the agent how to initialize or connect to the local AgentClick server, establishes the common protocol for submitting proposals and receiving review results, and routes the agent to the appropriate sub-skill based on task intent—without requiring the agent to implement each review mode from scratch.

Below this, AgentClick defines a set of *sub-skills* for concrete task scenarios such as email, planning, trajectory inspection, code review, memory management, and generic approval. Each sub-skill specifies the task-specific payload schema, expected review actions, and result handling conventions for its domain. For example, the email sub-skill defines how inbox items, drafts, and paragraph-level rewrites are represented; the planning sub-skill defines how steps, constraints, and edits are exchanged; and the trajectory sub-skill defines how intermediate execution traces are surfaced for inspection. This hierarchical design separates shared coordination logic from domain-specific review behavior, making the system easier to extend with new review modes over time.

2.4 Task-Specific Review Interfaces

A key design principle of AgentClick is that different agent outputs require different review affordances. Instead of presenting all agent proposals in a single generic interface, AgentClick provides task-specific browser UIs that match the structure of the artifact.

Email Review. Although agents can already connect to Gmail via tool calls, email interaction through a terminal remains impractical: users have no convenient way to browse and select specific messages, raw text rendering makes threads hard to read, and modifying a generated reply requires retyping an entire instruction even for a small change. AgentClick's Email UI addresses these limitations by combining inbox browsing, full message rendering, and reply drafting in a single interface (refer to Fig. 2). Users can select a specific email, request a draft only when needed, and edit the reply at the paragraph level with per-section rewrite, edit, and delete controls. The agent monitors user actions in real time and can update drafts accordingly. Final sending is gated on explicit user confirmation. Figure 2 illustrates the contrast between terminal-based email interaction and the AgentClick Email UI.

Plan Review. Terminal-based planning has several practical problems. Agent reasoning and plan steps are interleaved in a single output stream, forcing users to scroll back to reconstruct the full plan. Long plans are tedious to read as raw text. And when the agent pauses for user input mid-plan, users must make choices without visibility into subsequent steps—while any conversational exchange at that point risks disrupting the agent's ability to surface later decision points correctly. AgentClick's Plan UI addresses these issues by presenting the entire proposed workflow upfront as structured step cards before execution begins. Each card is typed and annotated—distinguishing tool calls, file operations, code execution, and other action categories—so users can scan the plan at a glance

Figure 2: Terminal vs. AgentClick Browser UI for email review. In the terminal (top), agent-drafted emails are hard to review and edit. AgentClick (bottom) renders the full inbox, message content, and editable reply drafts, with execution gated on explicit user confirmation.

and direct attention to steps that warrant closer review. Individual steps can be edited in place: users may modify descriptions, adjust constraints, reorder steps, or remove unnecessary actions. This structured representation makes long-horizon plans both more legible and more precisely correctable than their terminal equivalents.

Code Review. Although agents already output diffs for proposed code changes, terminal-based review has practical limitations: large changesets are displayed as a flat stream of additions and deletions that is difficult to parse, and diff output lacks surrounding context, making it hard to understand whether a change is correct without opening the file separately. AgentClick's Code UI addresses this by presenting each proposed change as a structured review artifact (refer to Fig. 7). Users see the proposed command alongside a natural-language explanation, a list of affected files, and a unified diff—but can also expand any file to view its full content with the change in context. Rather than accepting or rejecting the entire changeset, users can approve individual hunks, annotate specific lines with targeted instructions, or request a partial rewrite of selected sections. This turns code execution from an opaque terminal event into a reviewable, precisely correctable process.

Memory Management. Memory handling is a persistent pain point in current agent systems. Agents typically abstract over memory operations entirely, giving users no visibility into what is being stored, retrieved, or discarded. The problem is especially acute during context compaction: when a session grows long, the agent automatically summarizes and clears the conversation, but the resulting summary often omits content the user actually needs, and any memory that should persist must be manually reloaded in the next session by locating the relevant file in the project tree. AgentClick's Memory UI addresses this by making memory state explicit and controllable at every stage. Users can inspect all currently loaded memory entries, review proposed updates before they are committed, and guide how session memory is constructed during compaction. When compaction occurs, AgentClick generates a structured memory summary and surfaces it for user review and editing before it is saved (refer to Fig. 4 in Appendix). All available memory files are listed in a single panel, allowing users to load or unload entries with a single click rather than navigating the file system. This gives users direct control over what the agent knows, what it forgets, and what carries forward across sessions.

Trajectory Inspection. In current agent systems, execution trajectories are hidden by default: users see only the final output, with no visibility into the sequence of tool calls, retries, and intermediate decisions that produced it. This makes it difficult to diagnose unexpected behavior—such as excessive token consumption—even after a task completes successfully. AgentClick's Trajectory UI makes the full execution trace visible and structured (refer to Fig. 6 in Appendix). Tool calls, failed attempts, and error recoveries are highlighted distinctly, allowing users to follow the agent's decision path at a glance and pinpoint where inefficient or undesirable behavior occurred. Users can also annotate specific steps with guidance, which is saved to memory and used to inform the agent's behavior on similar tasks in future sessions. This turns trajectory inspection from a post-hoc debugging tool into an active channel for human oversight and iterative agent improvement.

2.5 Remote and Cross-Device Review

Although AgentClick starts as a local server, an important deployment setting is agents running on remote or headless infrastructure such as cloud VMs, shared servers, or containers. In these cases, users often interact with agents indirectly through text-based channels such as Discord, with no convenient access to the underlying terminal. AgentClick addresses this by exposing the review layer over HTTP, so the same browser-based interfaces are accessible remotely from any device. For example, a user running an OpenClaw agent on a Mac Mini can receive an approval prompt in Discord, open the AgentClick URL on their phone, review the full email draft or plan in the browser, make edits, and confirm—without ever touching the terminal. Rather than parsing long textual updates or typing corrections through chat, users can inspect drafts, adjust plans, monitor trajectories, and approve actions through a structured web interface on their own device.

2.6 Preference Memory and Adaptation

To reduce repeated correction effort across sessions, AgentClick captures user preferences from review interactions. When users edit or delete parts of an artifact and provide an explicit reason, the system records this feedback in a structured `MEMORY.md` file that the agent incorporates in future runs. This is especially useful for recurring tasks such as email writing, where similar stylistic

corrections would otherwise be applied repeatedly. Over time, the review layer functions not only as an approval interface for current actions, but also as a lightweight mechanism for accumulating user preferences and improving future human–agent co-work.

3 Evaluation

Evaluation Goals. We evaluate AgentClick through the lens of human–agent collaboration rather than autonomous task completion in isolation. Our goal is to assess whether AgentClick fills the interaction gaps identified in Section 1: limited visibility into proposed actions, cumbersome localized correction, and weak support for remote or cross-device oversight. We focus on whether the system enables users to communicate intent more effectively, intervene on structured artifacts before execution, and supervise agents in deployment settings where terminal-only interaction is inconvenient or unavailable. We exercise these three capabilities through representative task walkthroughs: plan review and constraint injection before execution, targeted artifact rewrite for better alignment with user intent, and remote review of an agent running on headless infrastructure. In this demo paper, we treat these walkthroughs as capability-oriented illustrations rather than a formal controlled user study. We release the full codebase and demo scripts (and a video) to support reproducibility and enable others to verify and extend the showcased workflows (https://github.com/agentlayer-io/AgentClick/tree/main).

3.1 Task Walkthroughs

Task 1 — Plan review and targeted constraint injection. As shown in Fig. 5, Claude Code was asked to generate a plan for training ResNet-18 on CIFAR-10. In the terminal, the agent surfaced the plan as a multiple-choice prompt ("Which framework? 1. PyTorch 2. TensorFlow 3. Type something"), offering no visibility into downstream steps at the time of choosing. In AgentClick's Plan UI, the same plan was presented as structured step cards showing the full pipeline, including framework configuration, training loop implementation, execution, checkpointing, and final evaluation. The user then injected two targeted constraints before execution: for the training loop step, "Save checkpoint every 10 epochs + best model by val accuracy," and for the execution step, "Monitor GPU utilization; ensure >90% utilization for efficiency." The agent subsequently executed the modified plan with these constraints incorporated. This walkthrough demonstrates how AgentClick allows users to express intent at the level of the plan artifact itself, rather than requiring them to anticipate all constraints upfront in a terminal exchange or interrupt the workflow later after execution has begun.

Task 2 — Email rewrite for intent preference. Claude Code was asked to reply to an email about a presentation draft. The agent's initial reply was written in formal English and did not match the user's preferred tone. In AgentClick's Email UI, the user selected a paragraph for rewriting and provided an explicit reason requesting that the response use emoji and adopt a lighter style. The revised draft was then confirmed and sent. As shown in Fig. 3, when the agent generated a reply for the next email, it automatically adopted the updated stylistic preference without requiring the user to restate the instruction. This walkthrough highlights a central goal of AgentClick: improving the agent's ability to infer and operationalize human intent through structured review. Rather than treating correction as a one-off override, the system keeps the agent attached to the live review context, allowing user-provided rewrite reasons to shape subsequent generation in the interaction loop.

Table 1: Capabilities exercised in the evaluation tasks

Capability	Task 1	Task 2	Task 3
Structured artifact review	✓	✓	✓
Targeted user intervention	✓	✓	✓
Pre-execution modification	✓		✓
Improved intent communication	✓	✓	✓
Preference/style adaptation		✓	
Remote/cross-device review			✓
Terminal-free human oversight	✓	✓	✓

Task 3 — Remote review from a mobile device. To evaluate AgentClick in a remote deployment setting, we ran an agent under OpenClaw on a server and exposed AgentClick through a Cloudflare tunnel. The resulting URL was opened in a mobile browser, where the user reviewed an email draft in the AgentClick UI, edited one paragraph, and confirmed the result. The server-side agent then received the approval and dispatched the email. This walkthrough demonstrates that AgentClick extends human-in-the-loop review to settings where the agent and the user are physically separated and the terminal is not the primary interface. In contrast to terminal-only workflows or chat-based relays such as Discord, the user can inspect and modify the artifact directly from a personal device through a structured browser interface.

3.2 Capability Coverage

Table 1 summarizes the capabilities exercised by the three walkthroughs. Task 1 shows that AgentClick supports pre-execution intervention on long-horizon plans, allowing users to refine downstream behavior before expensive execution begins. Task 2 shows that targeted rewriting with explicit reasons improves alignment between agent output and user intent, reducing the need for repeated restatement. Task 3 shows that the same review model remains usable when agents run on remote or headless infrastructure, supporting cross-device oversight without terminal access. Across all three tasks, the key benefit is not simply that AgentClick inserts a pause before execution, but that it gives users a more effective way to communicate intent and make localized corrections at the level of the artifact under consideration—whether a plan, an email draft, or a remote approval request.

4 Related Work

AgentClick relates to three lines of prior work. **Human-in-the-loop agent systems** study how users can remain involved during agent execution, including through co-planning, action approval, interactive plans, and preference learning [3, 6, 7]. In contrast, AgentClick targets a lighter-weight and more directly deployable review layer for existing terminal-based agents, with explicit preference capture through reason-tagged edits to structured memory files. **Remote access and terminal interfaces** make terminal-based agents accessible through web or mobile interfaces, allowing users to interact with remotely running agents from personal devices [9]. Unlike these systems, AgentClick does not simply expose a chat or terminal view, but provides task-specific interfaces for inspecting, editing, and approving structured artifacts. **Debugging and execution platforms** support agent development and inspection either by running agents inside dedicated infrastructures

or by analyzing their traces after execution [2, 11]. By contrast, AgentClick attaches to existing terminal agents and intervenes before or during execution, enabling structured review rather than post-hoc debugging or sandboxed re-hosting.

5 Conclusion

We presented AgentClick, a lightweight browser-based review layer that improves human–agent collaboration for terminal-based agents without modifying the underlying system. By surfacing structured, artifact-specific interfaces for email, planning, code, memory, and trajectory review, AgentClick addresses the visibility, correction, and accessibility gaps of terminal-first interaction. We hope it demonstrates that the review interfaces are a practical path toward more effective and accessible human–agent co-work.

References

[1] Anthropic. 2024. Claude Code. https://docs.anthropic.com/en/docs/claude-code. Accessed: 2026-03-14.

[2] Will Epperson, Gagan Bansal, Victor Dibia, Adam Fourney, Jack Gerrits, Erkang Zhu, and Saleema Amershi. 2025. Interactive Debugging and Steering of Multi-Agent AI Systems. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*.

[3] K. J. Kevin Feng, Kevin Pu, Matt Latzke, Tal August, Pao Siangliulue, Jonathan Bragg, Daniel S. Weld, Amy X. Zhang, and Joseph Chee Chang. 2026. Cocoa: Co-Planning and Co-Execution with AI Agents. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*.

[4] Sven Gronauer and Klaus Diepold. 2022. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review* 55, 2 (2022), 895–943.

[5] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680* (2024).

[6] Kaiqu Liang, Julia Kruk, Shengyi Qian, Xianjun Yang, Shengjie Bi, Yuanshun Yao, Shaoliang Nie, Mingyang Zhang, Lijuan Liu, Jaime Fernández Fisac, Shuyan Zhou, and Saghar Hosseini. 2026. Learning Personalized Agents from Human Feedback. *arXiv preprint arXiv:2602.16173* (2026).

[7] Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, et al. 2025. Magentic-ui: Towards human-in-the-loop agentic systems. *arXiv preprint arXiv:2507.22358* (2025).

[8] OpenAI. 2025. Codex CLI: A Coding Agent for the Terminal. https://developers.openai.com/codex/cli. Accessed: 2026-03-14.

[9] siteboon. 2026. Cloud CLI (aka Claude Code UI): A desktop and mobile UI for Claude Code, Cursor CLI, Codex, and Gemini-CLI. https://github.com/siteboon/claudecodeui GitHub repository, accessed 2026-03-14.

[10] Peter Steinberger. 2025. OpenClaw: Your Own Personal AI Assistant. https://github.com/openclaw/openclaw. Accessed: 2026-03-14.

[11] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2025. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *International Conference on Learning Representations*.

[12] Hanwen Xing, Haomin Zhuang, Xuandong Zhao, Yue Huang, Zhenheng Tang, and Xiangliang Zhang. 2026. Recipes for Agents: Understanding Skills and Their Open Questions. Preprint, ResearchGate. doi:10.13140/RG.2.2.11421.99045

Appendix

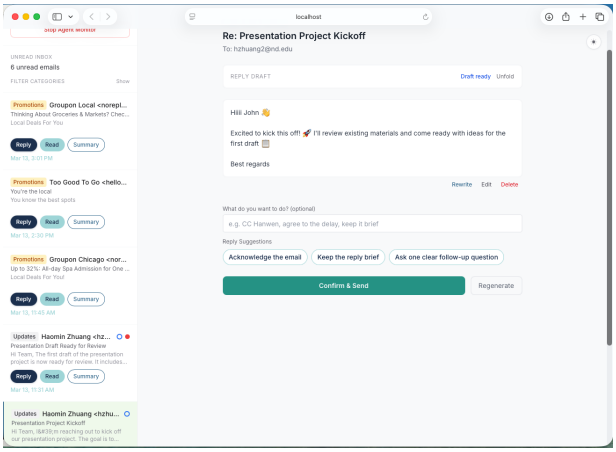
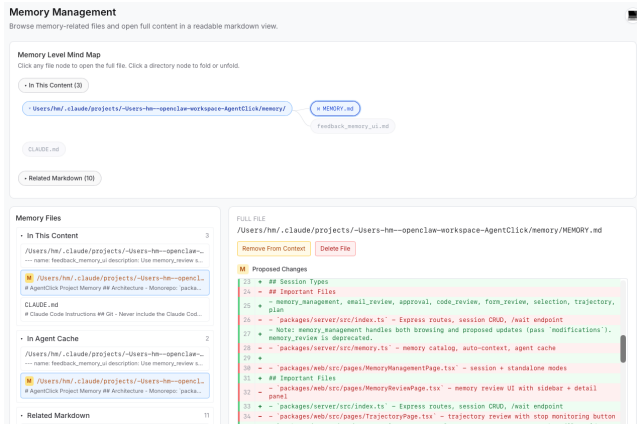
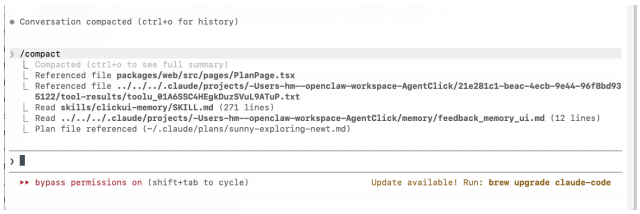


Figure 3: The email reply learns from preference then generate reply with Emoji by default.

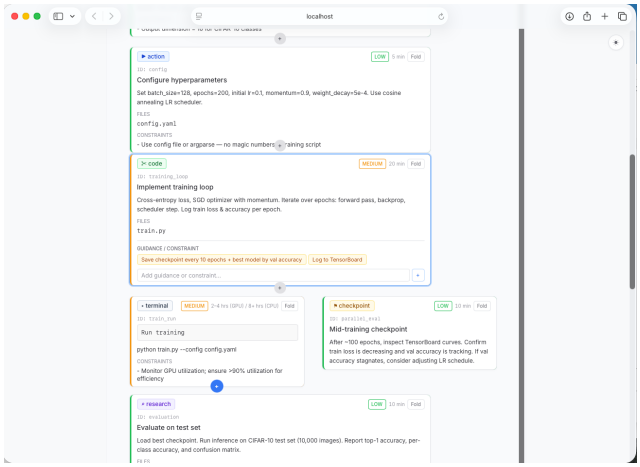


(a) Memory Browser UI

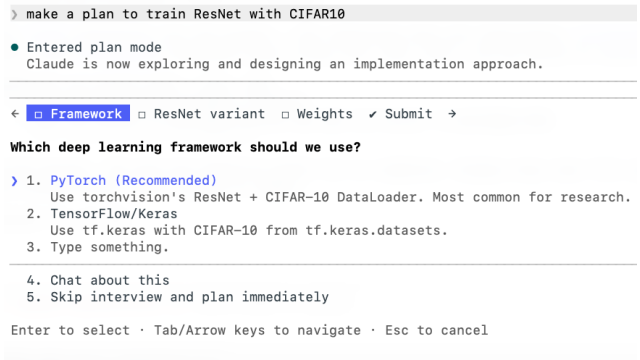


(b) Memory TUI

Figure 4: AgentClick Memory UI. Before context compaction, the interface surfaces all memory entries modified in the current session alongside a generated session summary. Users can review and edit the summary before it is committed, and selectively choose which entries to load into the next session.



(a) Planning Browser UI



(b) Planning TUI

Figure 5: Planning Click UI vs. TUI: Global visibility enables more informed framework selection

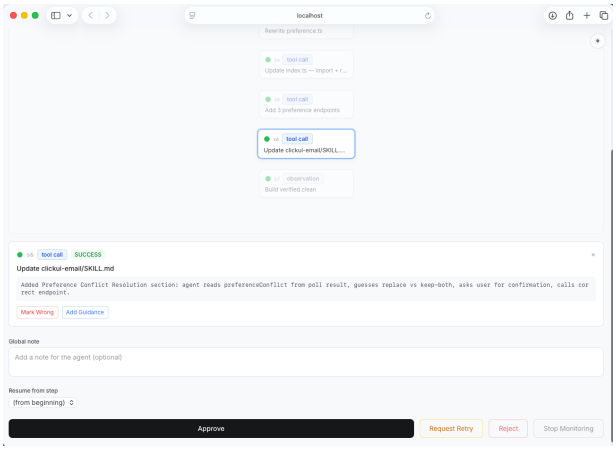


Figure 6: Click UI for trajectory.

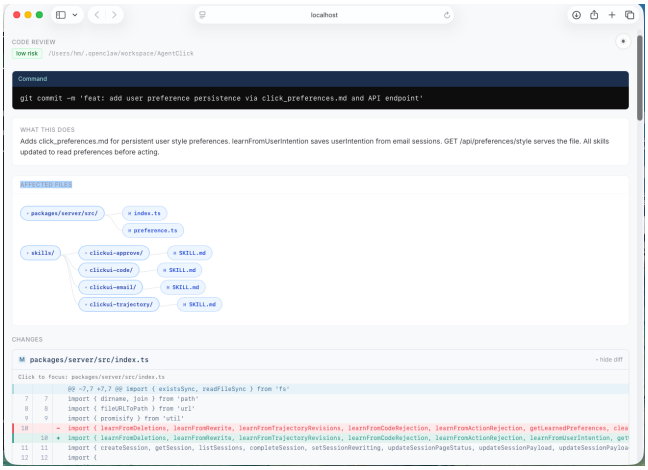


Figure 7: Click UI for code review.