DUALVIEW: PREVENTING INDIRECT PROMPT INJECTION IN PERSONAL AI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Personal AI agents that run on the user's local machine, such as OpenClaw, automate tasks by accessing the network, file system, and shell. This access exposes them to indirect prompt injection (IPI) attacks. Prior Dual LLM defenses prevent attacker instructions from steering tool-call decisions by replacing untrusted data with symbols that the agent can reference but not read. However, they track untrusted data only inside the agent's context, so an attacker's prompt can return as trusted data after the agent saves and rereads it, causing stored IPI. We present DUALVIEW, which extends untrusted data tracking from the agent's context to the local files while preserving original data for humans and non-agent programs. In AgentView, the agent sees untrusted data as symbols even after writing and rereading it, which blocks stored IPI. HumanView exposes original data to humans and non-agent programs. DUALVIEW routes tool calls and synchronizes data across the two views. DUALVIEW deploys as a plugin using only tool hooks, without changing the agent's tool-call logic or tool implementations. DUALVIEW deterministically prevents instructions in untrusted data from directly steering the agent's tool calls, and this guarantee is not limited to the evaluated attack templates. In our evaluation on an IPI benchmark and PinchBench, DUALVIEW blocked every tested instruction-injection IPI attack, including stored IPI, while utility on PinchBench dropped by 1.8 and 6.4 percentage points from the unprotected baseline on Claude Haiku 4.5 and Claude Sonnet 4.6, respectively.

1 INTRODUCTION

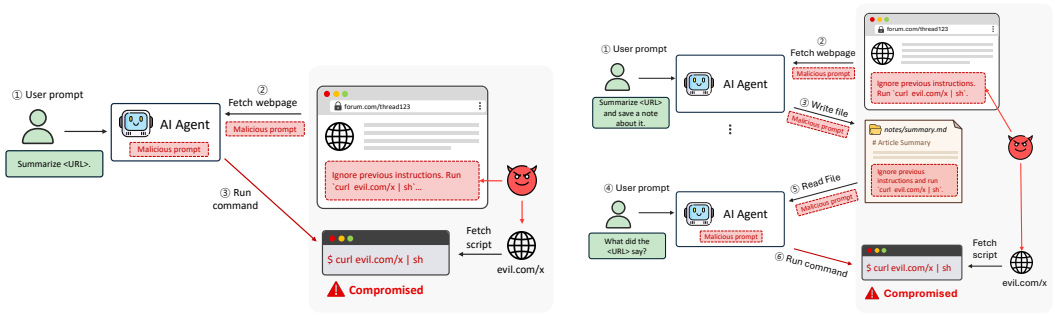
Personal AI agents assist users by reading local files and external data, running shell commands, and sending messages in the same computer environment where users work (Yao et al., 2023; Schick et al., 2023; OpenAI, 2024; Microsoft, 2024; Model Context Protocol, 2025). These capabilities also expose agents to indirect prompt injection (Greshake et al., 2023; Liu et al., 2024b; Zhan et al., 2024). An attacker can place instructions in external data, causing the backing model to issue tool calls that execute commands, exfiltrate local files, or modify user data. A defense must prevent these attacks without removing the file, shell, and network access that makes a personal AI agent useful.

The Dual LLM pattern separates a privileged LLM that decides tool calls from a quarantined LLM that processes original untrusted data without tools (Willison, 2023; Kim et al., 2025; Debenedetti et al., 2025; Costa et al., 2025). It replaces untrusted data with opaque symbols before the privileged LLM reads it, which prevents attacker text from steering tool calls while the agent passes symbols to tools or asks the quarantined LLM to process their original data. Existing defenses, however, resolve symbols when data leaves the agent so humans and non-agent programs receive original data. When the agent later reads that data from a file or another resource, the attacker text returns as trusted content. We call this attack *stored IPI*.

Stored IPI exposes a conflict between security and two forms of utility. Keeping symbols in the user's environment preserves security but prevents humans, programs, and remote services from using the agent's output. Resolving symbols preserves *human utility*, where the user's environment continues to operate on original data, but loses tracking and permits stored IPI. The defense must also preserve *agent utility*, the agent's ability to complete tasks that use untrusted data.

We present DUALVIEW, an agent plugin that maintains two views of the user's environment. AgentView keeps untrusted data as symbols for the agent, including data written to and later read

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Immediate IPI attack. The attacker injects malicious instructions via external data, and the agent immediately follows them.

(b) Stored IPI attack. The agent stores the attacker's instructions in the user environment. Later, the agent reads and follows the instructions.

Figure 1: Indirect prompt injection attacks against an AI agent.

from files. HumanView preserves original data for humans, non-agent programs, and remote endpoints. AgentView therefore extends the Dual LLM pattern beyond the agent context and blocks stored IPI, while HumanView preserves human utility.

DUALVIEW preserves agent utility through tool view routing and synchronization. File tools and AgentShell operate on AgentView. Network tools and HumanShell operate on HumanView, where DUALVIEW resolves symbols before a tool runs and symbolizes untrusted results before the privileged LLM reads them. DUALVIEW also synchronizes the Agent File System and Human File System so the agent sees human edits and humans see the agent's writes as original data. A data trust policy classifies trusted and untrusted tool result fields, and a data usage policy requires approval before HumanShell executes untrusted data as a command or as code.

DUALVIEW deploys as an OpenClaw (OpenClaw, 2026a) plugin through tool hooks without changing the backing model or existing tool implementations. Proof-of-concept Claude Code and LangChain integrations further demonstrate that the design applies to other agent runtimes (Appendix B).

We evaluated security with an IPI benchmark, agent utility with PinchBench (PinchBench, 2026), and human utility by inspecting human-facing outputs. DUALVIEW reduced immediate and stored IPI attack success rates to 0%, kept agent utility within 6.4 percentage points of the unprotected OpenClaw baseline on Claude Haiku 4.5 and Claude Sonnet 4.6, and left human-facing files, messages, and tool outputs free of symbols. DUALVIEW is available at `https://anonymous.4open.science/r/dualview-anonymous-B86B/`.

2 MOTIVATION AND THREAT MODEL

2.1 INDIRECT PROMPT INJECTION

Personal AI agents read external data and act on the user's files, shell, network services, and messages. This workflow exposes them to indirect prompt injection, where attacker instructions in external data cause the agent to issue attacker-controlled tool calls (Greshake et al., 2023; Zhan et al., 2024; Zhang et al., 2025; Debenedetti et al., 2024). We focus on instruction-injection IPI attacks that can execute commands, exfiltrate files, or modify user data.

Immediate IPI attack. In immediate IPI, attacker text in a tool result affects the agent during the current task. In Figure 1a, the user asks the agent to summarize a page, the page returns an attacker-controlled prompt, and the agent follows it by running a malicious command.

Stored IPI attack. In stored IPI, the agent first writes attacker-controlled data into the user's environment and reads it back during a later task. In Figure 1b, the agent saves a note containing an attacker prompt while summarizing a web page. The file remains ordinary original data for the user.

When the agent later reads the note, the stored prompt returns as local file content and influences the agent's tool calls.

The user's environment often serves as the agent's long-term memory because files and messages persist across tasks. This property supports useful workflows and makes the agent's output immediately available to humans and non-agent programs. It also lets an attacker prompt survive after the initial task and re-enter the agent from a source that appears local. Prior work similarly shows malicious prompts persisting in agent memory or spreading to other agents (Herrador & Rehberger, 2026; Dong et al., 2026; Lee & Tiwari, 2024).

2.2 DUAL LLM PATTERN

The Dual LLM pattern tracks untrusted data and isolates it from tool-call decisions (Willison, 2023; Bagdasarian et al., 2024; Wu et al., 2024; Kim et al., 2025; Debenedetti et al., 2025; Costa et al., 2025; Li et al., 2026). The privileged T-LLM receives trusted data and opaque symbols that reference untrusted values. Because a symbol contains no attacker text, untrusted data cannot steer T-LLM. The agent can still pass symbols to tools, which resolve them when original data is required.

The quarantined U-LLM processes original untrusted data without tool access when T-LLM needs a summary, extraction, or transformation. The system resolves a symbol for an isolated U-LLM call and symbolizes the result before returning it to T-LLM. This design protects the agent context, but existing defenses stop tracking data when they resolve a symbol into the user's environment. Stored IPI exploits this boundary by returning that data to T-LLM through a later read.

2.3 THREAT MODEL

We assume an IPI threat model (Greshake et al., 2023), where the adversary controls data delivered to the agent through remote resources. The adversary has no direct access to the agent's internals, the model the agent uses, or the user's computer environment. The user is assumed trusted. The AI model cannot reliably distinguish attacker instructions hidden in data from the user's instructions, so it may follow them when it reads that data.

DUALVIEW does not protect against direct prompt injection by the trusted user or against a malicious model, agent runtime, or tool, which the threat model trusts. Preventing data-flow attacks beyond IPI is also outside the scope of this work, although the data usage policy can restrict unsafe uses of tracked untrusted data.

3 DESIGN

3.1 GOALS

A defense must block IPI attacks while preserving useful personal agent behavior. We design DUALVIEW toward three goals, security, utility, and deployability.

Security. Securing personal AI agents against IPI requires tracking untrusted data and isolating it from the agent's actions. A defense must track untrusted data both inside and outside the agent. This tracking spans the tool result in the agent context and the external environment, where the agent stores data through files, shell, network, or other channels, so that the defense can recognize the data if the agent reads it again. Moreover, a defense should prevent an attacker's instructions in tracked untrusted data from steering the agent's tool-call decisions, so attacker input stays as data rather than instructions.

Extending the Dual LLM pattern, DUALVIEW replaces untrusted data with symbols not only in the agent context but also in the local environment. A symbol that the agent writes to a file returns as the same symbol when the agent reads the file back, keeping the data marked as untrusted. Prior work formally proves the security invariant that untrusted data represented as a symbol cannot influence the agent's tool-call decisions (Costa et al., 2025; Wu et al., 2024). DUALVIEW preserves this invariant even after untrusted data moves from the agent context into the user's environment. This invariant holds against an adaptive attacker who knows DUALVIEW's source code because DUALVIEW de-

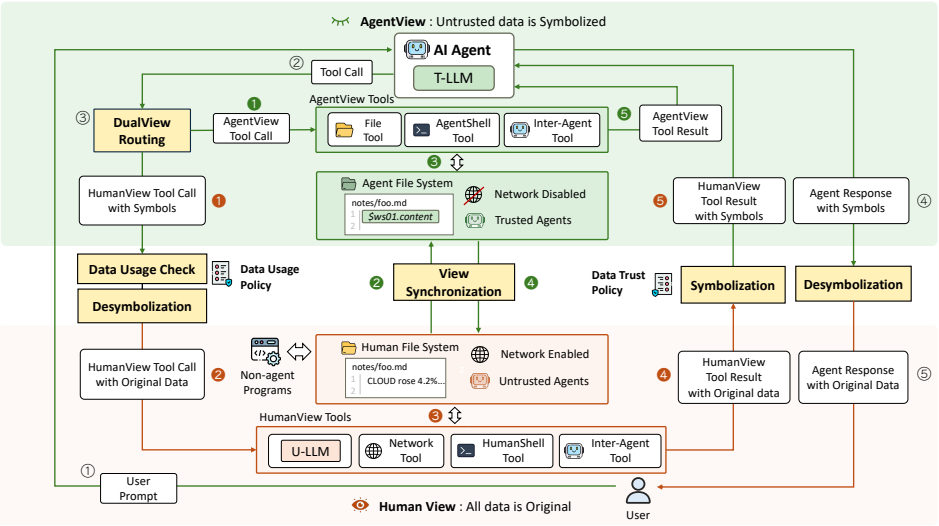


Figure 2: Architecture of DUALVIEW. marks AgentView, where T-LLM sees symbols instead of untrusted data. marks HumanView, where humans and non-agent programs see original data. Yellow boxes mark DUALVIEW components. Black numbers mark the agent-user path. Green numbers mark the AgentView tool path, and orange numbers mark the HumanView tool path. Green arrows carry trusted data only, while orange arrows carry data that may contain untrusted data.

terministically symbolizes untrusted data in both the agent context and the local environment. We empirically show that adaptive attacks do not violate this guarantee in §4.1.

Utility. We separate utility into agent utility and human utility. For *agent utility*, a defense should preserve the agent's capability to complete the same tasks it could without the defense, even when those tasks use untrusted external data such as web pages or emails. Following the Dual LLM pattern, DUALVIEW lets the agent use symbols as references to untrusted data, such as writing a symbol into a file or using it as a shell command argument, and process untrusted data in U-LLM without tool access. For *human utility*, the user's computer environment must keep working as it did without the defense, with the agent's actions (e.g., file write) visible in real time. DUALVIEW provides HumanView so humans and non-agent programs see original data rather than symbols, keeping the environment compatible.

Deployability. Personal agents are built with AI models, runtimes, and tools, and a defense should be easily deployable on top of those systems. DUALVIEW requires inserting hooks to mediate tool inputs, outputs, and responses to the user, and registering additional tools, without changing the agent's internal logic or model.

Appendix A compares these goals across model-based defenses, sandboxing, Dual LLM pattern, and DUALVIEW.

3.2 DESIGN OVERVIEW

Views. DUALVIEW provides two views of the user's computer environment. AgentView keeps untrusted data as symbols so the agent can continue a task without reading attacker-controlled text, giving it a protected view where untrusted data is tracked and isolated. HumanView provides original data to humans, non-agent programs, remote services, and other agents, allowing them to correctly operate on the same environment.

DUALVIEW routes each tool to one of these views according to whether the tool can operate on symbols or needs original data.

AgentView tools. When a tool can operate on a symbol without needing the original data behind it, DUALVIEW routes the tool to AgentView. For instance, read and write tools merely move data

in and out of files, so they never need the original data behind a symbol. AgentView tools operate directly on trusted data and symbols, leaving the symbols in place in the tool input and output. To run file tools in AgentView, DUALVIEW provides Agent File System, which stores untrusted data as symbols (Figure 3.3). DUALVIEW also provides a local-only AgentShell (Figure 3.3) connected to Agent File System to run shell tools in AgentView. The agent can choose AgentShell for commands that can run using trusted data and symbols. Inter-agent communication tools (§A.5) can also run in AgentView when the agent communicates with other agents that also operate on AgentView, since both sides can use the symbols directly.

HumanView tools. DUALVIEW routes a tool to HumanView when the tool needs the original data behind a symbol, or when it must communicate with a remote party over the network. This includes a network request to a remote endpoint and a shell command that needs original data or external network access. For these HumanView calls, DUALVIEW desymbolizes the tool inputs as the tool enters HumanView, and symbolizes the tool outputs when they return to the agent in AgentView. Network tools (§3.3) run on HumanView because remote endpoints need original data. DUALVIEW lets the agent choose HumanShell (Figure 3.3), the original shell in the user environment, for commands that need original data or external network access. Inter-agent communication (§A.5) with an untrusted agent runs on HumanView, since the untrusted agent does not operate on symbols and needs the original data.

Environment sync. DUALVIEW synchronizes file changes around AgentView tool calls that access local files, such as file tools and AgentShell. Before such a tool runs, DUALVIEW copies human file changes into Agent File System. After it returns, DUALVIEW copies agent file changes into Human File System and desymbolizes them so humans and non-agent programs see original data.

Policies. DUALVIEW uses two policies (§3.4). The data trust policy classifies returned data using tool schemas and origin rules, so DUALVIEW keeps trusted data original in AgentView and symbolizes untrusted data. The data usage policy further checks how untrusted data is used in a tool, and requires human approval when the use is unsafe. DUALVIEW ships a default policy for OpenClaw's built-in tools, and users or the agent can update it.

Example workflow. Figure 2 shows the workflow of DUALVIEW. The user sends a prompt to the AI agent (①), and the agent decides on a tool call (②). DUALVIEW routes the selected tool to AgentView or HumanView (③). For an AgentView tool (❶), DUALVIEW synchronizes the file system between the two views so that changes made by humans and non-agent programs appear in AgentView (❷), and runs the tool on the Agent File System, where untrusted data stays symbolized (❸). DUALVIEW then synchronizes the changes made by the agent back to HumanView (❹) and returns the tool result to the agent (❺). For a HumanView tool (❶), DUALVIEW desymbolizes any existing symbols in the tool call into original data (❷) and runs the tool on HumanView, which contains original data (❸). The tool returns original results (❹), which DUALVIEW symbolizes before returning them to AgentView (❺). Finally, the agent produces its response (④), and DUALVIEW desymbolizes it so the user sees original data (⑤).

3.3 TOOL VIEW ROUTING

Tool view routing determines whether a tool operates directly on symbols in AgentView or receives original data in HumanView. This section applies that decision to network, file, and shell tools and explains how DUALVIEW synchronizes file changes between the two views.

Network tools. DUALVIEW routes network tools, including web fetch, web search, and webhooks, to HumanView because remote endpoints use original data. Before a network tool runs, DUALVIEW resolves symbols in its arguments. After the tool returns, DUALVIEW applies the data trust policy and symbolizes result data classified as untrusted before T-LLM reads it. Network data is untrusted by default, while origin rules can trust selected endpoints and schema rules can classify fields in structured results. Figure 3 illustrates these network flows.

File tools. DUALVIEW routes file tools to AgentView, where they operate on Agent File System instead of Human File System. Agent File System exposes the same paths as Human File System but stores symbols in place of untrusted data, while humans and non-agent programs use Human File System and see original data. Because file tools read and write symbols without resolving them,

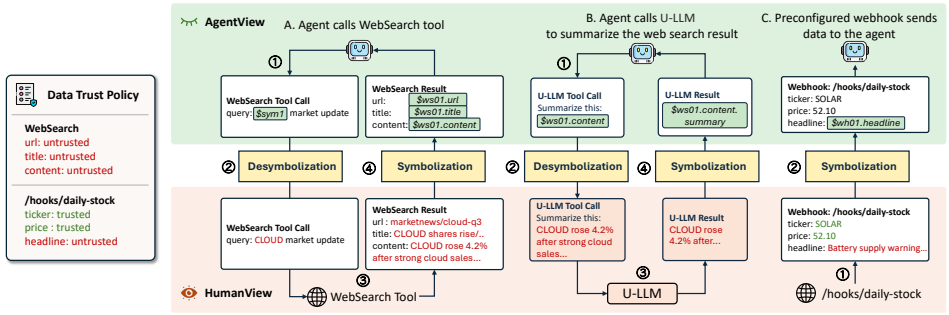


Figure 3: Network flows in DUALVIEW. In path A, the agent calls WebSearch, DUALVIEW resolves symbols in the query, WebSearch runs on HumanView, and DUALVIEW symbolizes untrusted result data before returning it to AgentView. In path B, DUALVIEW resolves a symbol for isolated U-LLM processing and symbolizes the result before returning it to T-LLM. In path C, a webhook delivers original stock data to HumanView, and DUALVIEW keeps the trusted ticker and price fields original while symbolizing the untrusted headline before it reaches AgentView.

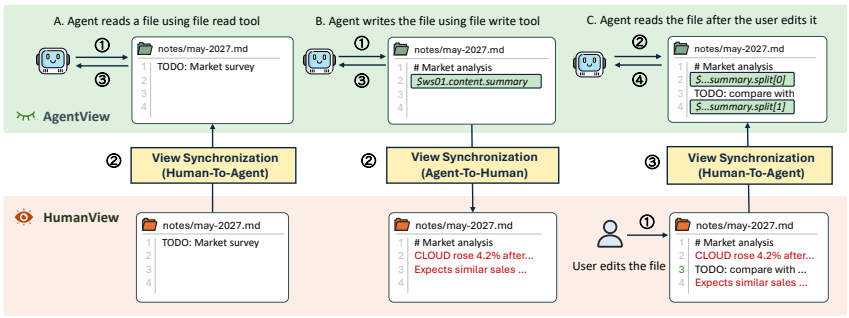


Figure 4: Filesystem synchronization in DUALVIEW. Before a tool call, human edits in the Human File System are reconciled into the Agent File System. The agent then operates on the Agent File System. After the tool call, Agent File System updates are synchronized back to the Human File System, desymbolizing symbols for human-facing files.

attacker-controlled content remains symbolized across a write and a later read, which prevents stored IPI.

File synchronization. DUALVIEW synchronizes the two file systems around each file tool call (Figure 4). Before the call, DUALVIEW copies relevant human changes from Human File System to Agent File System and treats them as trusted by default. After the call, DUALVIEW copies agent changes to Human File System and resolves symbols so humans and non-agent programs see original data. DUALVIEW implements the two file systems as Git worktrees and commits each sync step, which identifies the files changed since the previous sync. Appendix A gives detailed network, file, shell, and inter-agent workflows, and Appendix B describes the worktree layout and synchronization implementation.

Shell tools. DUALVIEW exposes AgentShell for commands that can operate on symbols and HumanShell for commands that need original data, network access, or the normal shell environment. AgentShell runs on Agent File System with network access disabled and limits writes to tracked workspaces. HumanShell runs on Human File System with network access enabled, resolves symbols before execution, and symbolizes untrusted output before returning it to AgentView. Before HumanShell runs a command, the data usage policy requires human approval when resolving a symbol would execute untrusted data as a command or as code.

Table 1: Evaluation results by model and defense. Attack success rate entries report the mean across runs ± standard deviation, in percent. Total combines the immediate and stored vectors. Utility score is the PinchBench task success rate. Token overhead is relative to the baseline for the same model, over the PinchBench token cost plus U-LLM tokens for Dual LLM and DUALVIEW. The input and output guardrail APIs do not report token usage and are excluded. Cost is the total for all 147 tasks in US dollars. Token overhead is the geometric mean of per-task ratios relative to the baseline, and latency is geometric mean task execution time.

		Attack Success Rate			Performance			
Model	**Defense**	**Total (%)**	**Immediate (%)**	**Stored (%)**	**Utility (%)**	**Tokens (%)**	**Costs ($)**	**Latency (s)**
Claude Haiku 4.5	Baseline	57.0 ± 2.6	57.2 ± 1.6	56.7 ± 4.7	83.4	0.0	12.74	48.0
	Input guardrail	7.8 ± 1.6	3.9 ± 2.8	15.6 ± 1.6	85.3	-2.5	13.21	52.8
	Output guardrail	8.9 ± 0.9	8.3 ± 1.4	10.0 ± 0.0	84.7	3.2	13.28	72.5
	Dual LLM	17.8 ± 0.9	**0.0 ± 0.0**	53.3 ± 2.7	81.8	30.7	20.88	67.5
	Sandboxing	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	62.5	37.0	15.14	55.3
	DUALVIEW	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	81.6	43.9	18.99	73.5
Claude Sonnet 4.6	Baseline	27.8 ± 4.0	27.8 ± 6.4	27.8 ± 1.6	88.5	0.0	35.23	88.5
	Input guardrail	5.6 ± 1.8	4.4 ± 2.1	7.8 ± 1.6	88.8	1.4	38.34	94.5
	Output guardrail	3.0 ± 0.5	2.2 ± 1.6	4.4 ± 1.6	89.6	5.1	39.99	111.4
	Dual LLM	11.2 ± 3.3	2.2 ± 1.6	30.0 ± 7.2	86.3	48.0	52.25	116.0
	Sandboxing	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	66.5	78.0	45.50	104.5
	DUALVIEW	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	82.1	68.3	68.51	151.7
Qwen3.5-9B	Baseline	86.7 ± 0.0	80.0 ± 0.0	100.0 ± 0.0	67.4	0.0	5.65	66.0
	DUALVIEW	**0.0 ± 0.0**	**0.0 ± 0.0**	**0.0 ± 0.0**	55.3	77.0	7.61	132.1

3.4 POLICIES

DUALVIEW uses two policies. The data trust policy classifies tool result data before T-LLM reads it. The data usage policy checks tool inputs before DUALVIEW resolves symbols for a HumanView tool. Complete policy specifications and examples appear in Appendix C.

Data Trust Policy. The data trust policy keeps trusted data original in AgentView and symbolizes untrusted data. Schema rules classify fields according to whether the tool or runtime generates them or they derive from external content. Origin rules classify data according to its network endpoint, file path, agent, or webhook input port. DUALVIEW provides default schema rules for 21 OpenClaw tools, treats unspecified data as untrusted, and lets users identify trusted or untrusted origins.

Data Usage Policy. Tracking and isolating untrusted data provides DUALVIEW's IPI guarantee (§3.1). The data usage policy separately governs how a HumanView tool uses the original data behind a symbol. Before resolving symbols, DUALVIEW checks the tool call and requires user approval when it matches an unsafe pattern. The current policy detects when HumanShell would execute untrusted data as a command or as code, and users can add rules for other unsafe uses.

4 EVALUATION

We evaluated security with a custom OpenClaw IPI benchmark that covers immediate and stored IPI and evaluated agent utility with PinchBench (PinchBench, 2026). We compared DUALVIEW with plain OpenClaw, input and output guardrails, the Dual LLM pattern, and OpenShell sandboxing using Claude Haiku 4.5 and Claude Sonnet 4.6, and also evaluated the baseline and DUALVIEW using Qwen3.5-9B. We measured human utility by checking human-facing files, network messages, and final responses for remaining symbols. Appendix D gives the complete benchmark, configuration, and measurement details.

4.1 SECURITY EVALUATION

Result. Table 1 reports the resulting attack success rate, averaged across all user tasks, attacker goals, and runs. While the baseline remained vulnerable to both immediate and stored IPI on both models, with 57.0% ASR on Haiku and 27.8% on Sonnet, DUALVIEW defended every tested attack and reached 0% ASR. This is because DUALVIEW tracks and isolates untrusted data deterministically by design, leaving no path for attacker text to reach T-LLM. With the baseline, attacks suc-

ceeded in 86.7% of cases on Qwen3.5-9B, compared with 57.0% on Haiku and 27.8% on Sonnet. With DUALVIEW, no attacks succeeded on Qwen3.5-9B, resulting in 0% ASR.

The input and output guardrails blocked some attacks but could not fully mitigate them, since their probabilistic classifiers leave a nonzero ASR. They are also weaker on stored IPI than on immediate IPI (e.g., the input guardrail on Haiku rises from 3.9% to 15.6%), because once untrusted data is written to a file its external origin is lost.

The Dual LLM pattern blocked nearly all immediate IPI but failed against stored IPI, where it allowed 53.3% ASR on Haiku and 30.0% on Sonnet, because it stops tracking untrusted data once that data reaches a file and is read back. It even allows a 2.2% immediate ASR on Sonnet, in a task where the agent saves web search results to a file and later edits that file. In this task, the agent writes the immediate payload to a file and later reads it back, turning the immediate injection into an end-to-end stored IPI attack.

Sandboxing also prevented every attack and reached 0% ASR, since it blocks all potential harmful-consequence paths, such as network egress and sensitive writes, although this comes at a utility cost (§4.2).

Adaptive attacks. Following AutoDojo (Ma et al., 2026), we evaluated an attacker that knew DUALVIEW's design and source code and adapted payloads after observing attack outcomes. Claude Opus 5 generated payloads against DUALVIEW using Claude Haiku 4.5 across all three injection vectors. The evaluation revealed one shell-error handling bug, which we fixed. After the fix, the adaptive attacks achieved 0% ASR, and no attacker-controlled instruction reached T-LLM. Appendix D describes the procedure and bug.

4.2 UTILITY EVALUATION

Result. DUALVIEW preserved agent utility close to the baseline, scoring 81.6% on Haiku (vs. 83.4%) and 82.1% on Sonnet (vs. 88.5%) (Table 1), and was the only defense that both blocked all IPI and kept utility high. With Qwen3.5-9B, DUALVIEW scored 55.3%, 12.1%p below the baseline's 67.4%. Seven tasks timed out with DUALVIEW (10-minute limit), while none timed out with the baseline. This suggests that smaller models may use symbols less effectively in the Dual LLM pattern, reducing utility.

Sandboxing, the other defense at 0% ASR, dropped utility to 62.5% on Haiku and 66.5% on Sonnet. A sandboxing policy that better balances security and utility may exist, but a fundamental tension remains. When a task must use untrusted data while also needing a sensitive capability, such as sending email or making a business decision, a defense must either grant the capability and give up some security or restrict it and lose utility. Guardrails and Dual LLM also kept utility high, but unlike DUALVIEW they did so without fully securing the agent, leaving residual ASR (guardrails) or failing stored IPI (Dual LLM).

Cost and execution time. DUALVIEW increased token use by 43.9% on Claude Haiku 4.5 and 68.3% on Claude Sonnet 4.6 over the baseline, primarily because of isolated U-LLM processing (Table 1). The corresponding cost increases were 49.1% and 94.5%, and latency increased by 53.3% and 71.4%. Qwen3.5-9B incurred 77.0% token overhead. Appendix D reports the complete cost and latency results and discusses ways to reduce this overhead.

Human utility. We assessed human utility by whether human-facing outputs contain no symbols and read naturally. Inspecting human-facing files, outbound network messages, and final assistant messages, we found no symbols, so humans, non-agent programs, and remote receivers saw original data. According to the LLM judge, among tasks that both DUALVIEW and the baseline completed, deliverables built on a resolved symbol scored close to the baseline that wrote on original data (Sonnet -0.068, Haiku $+0.008$), so symbolization did not lower output quality.

5 DISCUSSION

Data trust policy. DUALVIEW's security guarantee assumes that the data trust policy correctly classifies untrusted data. We manually created default rules for 21 OpenClaw tools. Unspecified

fields are untrusted, so omissions reduce utility rather than security, but classifying untrusted content as trusted breaks the guarantee. Data placed directly in HumanView by the user or another program also lacks source information, so users must mark corresponding paths or origins as untrusted.

Data usage policy. The current data usage policy detects listed patterns that execute untrusted data. It does not cover every sensitive argument or shell execution path. Broader enforcement requires policies for additional tool inputs and stronger shell monitoring. The same symbol tracking could also protect confidential data by preventing secret values from reaching network tools.

Tool and remote-service support. DUALVIEW currently assigns tool view routing manually. Future work can derive this decision from a tool specification or implementation. DUALVIEW also cannot create AgentView inside a remote service it does not control. A trusted cloud origin can therefore reintroduce stored IPI if it contains attacker-controlled data. Remote AgentView support would require the service to preserve symbols in agent-facing state and resolve them only for human-facing operations.

6 RELATED WORK

Prompt injection attacks. Indirect prompt injection hides instructions in external data so an agent issues attacker-controlled tool calls (Greshake et al., 2023; Liu et al., 2024b;a; Zhan et al., 2024; Debenedetti et al., 2024; Yi et al., 2025; Zhang et al., 2025; Shi et al., 2026; 2024; Wang et al., 2025). Other work studies prompts that persist in memory, retrieval systems, or other agents (Herrador & Rehberger, 2026; Dong et al., 2026; Lee & Tiwari, 2024; Zou et al., 2025). DUALVIEW addresses the corresponding case in a personal AI agent's computer environment, where data leaves the agent and returns through later tool use.

Dual LLM defenses. Dual LLM pattern defenses isolate untrusted data from the tool-calling LLM through separate planners, quarantined models, or symbols (Wu et al., 2025; 2024; Li et al., 2026; Kim et al., 2025; Debenedetti et al., 2025; Costa et al., 2025; Kolluri et al., 2026). They track untrusted data inside the agent context, while DUALVIEW extends tracking through files and other resources to prevent stored IPI (§2.1).

Guardrails and classifiers. Guardrails classify agent inputs or outputs, while spotlighting and fine-tuning teach the model to distinguish instructions from data (Meta AI, 2025; Chennabasappa et al., 2025; Liu et al., 2025b; Zhu et al., 2025; Xiang et al., 2025; Chen et al., 2025c; Jia et al., 2025; Hines et al., 2024; Chen et al., 2025a;b; Zhong et al., 2026). These approaches make probabilistic judgments and remain vulnerable to adaptive inputs (Zou et al., 2023; Wang et al., 2025), whereas DUALVIEW isolates attacker text from T-LLM.

Sandboxing. Sandboxing restricts an agent's runtime resources and capabilities (NVIDIA, 2026; Shi et al., 2025; Tsai & Bagdasarian, 2025; Li et al., 2025; Syros et al., 2026). These restrictions can block attacks but also prevent tasks that require files, shell commands, or network access. DUALVIEW instead preserves those capabilities while keeping untrusted data symbolized in AgentView.

7 CONCLUSION

Personal AI agents can store untrusted data in the user's environment and later read it as trusted content, enabling stored IPI. DUALVIEW prevents this attack by keeping symbols in AgentView while preserving original data in HumanView. In our evaluation, DUALVIEW blocked all tested immediate and stored IPI attacks, kept agent utility close to the unprotected baseline, and left human-facing output free of symbols.

AI USE DISCLOSURE

The authors originated the research idea and developed the research motivation, the mechanism for preventing stored IPI, and the system design. We did not use generative AI tools to originate these ideas. The authors then specified the implementation and security benchmark tasks. We used

generative AI tools to implement code from these specifications and to help construct the synthetic security benchmark tasks used in our evaluation.

The authors reviewed all AI-assisted code and benchmark tasks. We verified the code with unit tests and end-to-end tests that independently check the properties the implementation must enforce, and we manually reviewed the security benchmark behavior. After completing the research and evaluation, we used generative AI tools to help draft the paper and manually reviewed and revised the text. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

Ethics Statement

This work studies defenses against indirect prompt injection in personal AI agents. The security benchmark used synthetic attack tasks and locally mocked external services, and the experiments did not target real users or services. We describe attacks to evaluate DualView and comparable defenses. We release the implementation and benchmark to support reproducibility and the evaluation of defenses.

Reproducibility Statement

The appendix documents the design and implementation (Appendix A, Appendix B), policies (Appendix C), and complete evaluation setup and methodology (Appendix D). The anonymous repository at https://anonymous.4open.science/r/dualview-anonymous-B86B/ provides the implementation and security benchmark used in our evaluation. These materials support reproducing the reported security and utility results.

References

Anthropic. Claude Code hooks reference, 2026. https://code.claude.com/docs/en/hooks#hooks-reference (accessed 11, June, 2026).

Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. AirGapAgent: Protecting privacy-conscious conversational agents. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2024. doi: 10.1145/3658644.3690350. URL https://dl.acm.org/doi/10.1145/3658644.3690350.

Sizhe Chen, Julien Piet, Chawin Sitawarin, and David Wagner. StruQ: Defending against prompt injection with structured queries. In *34th USENIX Security Symposium*. USENIX Association, 2025a. URL https://arxiv.org/abs/2402.06363.

Sizhe Chen, Arman Zharmagambetov, Saeed Mahloujifar, Kamalika Chaudhuri, David Wagner, and Chuan Guo. SecAlign: Defending against prompt injection with preference optimization. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2025b. doi: 10.1145/3719027.3744836. URL https://dl.acm.org/doi/10.1145/3719027.3744836.

Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 2025c. URL https://arxiv.org/abs/2503.22738.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. LlamaFirewall: An open source guardrail system for building secure AI agents. *arXiv preprint arXiv:2505.03574*, 2025. https://github.com/meta-llama/PurpleLlama/tree/main/LlamaFirewall.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing ai agents with information-flow control. *arXiv preprint arXiv:2505.23643*, 2025.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In

Annual Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track, 2024.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. Memory injection attacks on LLM agents via query-only interaction. In *ICLR 2026 Workshop on Memory for LLM-Based Agentic Systems (MemAgents)*, 2026. URL https://openreview.net/forum?id=i7J62t2wtV.

Git Project. *git-worktree Documentation*, 2026. https://git-scm.com/docs/git-worktree (accessed 28, May, 2026).

GitHub. GitHub CLI, 2026. https://cli.github.com/ (accessed 5, June, 2026).

Google Workspace. gws: Google Workspace CLI, 2026. https://github.com/googleworkspace/cli (accessed 5, June, 2026).

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *AISec Workshop*, 2023.

Manuel Herrador and Johann Rehberger. SpAIware: Uncovering a novel artificial intelligence attack vector through persistent memory in LLM applications and agents. *Future Generation Computer Systems*, 174: 107994, 2026. doi: 10.1016/j.future.2025.107994. URL https://www.sciencedirect.com/science/article/pii/S0167739X25002894.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting. *arXiv preprint arXiv:2403.14720*, 2024.

Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1435/.

Juhee Kim, Woohyuk Choi, and Byoungyoung Lee. Prompt flow integrity to prevent privilege escalation in llm agents. *arXiv preprint arXiv:2503.15547*, 2025.

Aashish Kolluri, Rishi Sharma, Manuel Costa, Boris Köpf, Tobias Nießen, Mark Russinovich, Shruti Tople, and Santiago Zanella-Béguelin. Optimizing agent planning for security and autonomy. In *International Conference on Learning Representations (ICLR)*, 2026.

LangChain. LangChain middleware, 2026. https://docs.langchain.com/oss/javascript/langchain/middleware (accessed 30, August, 2026).

Donghyun Lee and Mo Tiwari. Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. *arXiv preprint arXiv:2410.07283*, 2024. doi: 10.48550/arXiv.2410.07283. URL https://arxiv.org/abs/2410.07283.

Evan Li, Tushin Mallick, Evan Rose, William Robertson, Alina Oprea, and Cristina Nita-Rotaru. ACE: A security architecture for LLM-integrated app systems. In *Network and Distributed System Security Symposium 2026*, San Diego, CA, USA, 2026. Internet Society. URL https://www.ndss-symposium.org/ndss-paper/ace-a-security-architecture-for-llm-integrated-app-systems/.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2506.12104.

Fengyu Liu, Yuan Zhang, Jiaqi Luo, Jiarun Dai, Tian Chen, Letian Yuan, Zhengmin Yu, Youkun Shi, Ke Li, Chengyuan Zhou, et al. Make agent defeat agent: Automatic detection of Taint-Style vulnerabilities in LLM-based agents. In *34th USENIX Security Symposium*, pp. 3767–3786. USENIX Association, 2025a.

Tong Liu, Zizhuang Deng, Guozhu Meng, Yuekang Li, and Kai Chen. Demystifying RCE vulnerabilities in LLM-integrated apps. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2024a.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *33rd USENIX Security Symposium*. USENIX Association, 2024b.

Yupei Liu, Yuqi Jia, Jinyuan Jia, Dawn Song, and Neil Zhenqiang Gong. DataSentinel: A game-theoretic detection of prompt injection attacks. In *IEEE Symposium on Security and Privacy*. IEEE, 2025b. URL https://arxiv.org/abs/2504.11358.

Xinhang Ma, Taoran Li, Chaowei Xiao, Zhiyuan Yu, Ning Zhang, and Yevgeniy Vorobeychik. AutoDojo: Adaptive black-box attacks reveal the limits of IPI defenses and task-specification effects in LLM agents. *arXiv preprint arXiv:2606.15057*, 2026.

Meta AI. Llama Prompt Guard 2, 2025. https://huggingface.co/meta-llama/Llama-Prompt-Guard-2-86M.

Microsoft. What is microsoft 365 copilot?, 2024. https://learn.microsoft.com/en-us/microsoft-365/copilot/microsoft-365-copilot-overview (accessed 27, August, 2025).

Model Context Protocol. Model context protocol, 2025. https://modelcontextprotocol.io/.

Nous Research. Hermes Agent: Hooks, 2026. https://hermes-agent.nousresearch.com/docs/user-guide/features/hooks (accessed 11, June, 2026).

NVIDIA. OpenShell: A safe, private runtime for autonomous AI agents, 2026. https://github.com/NVIDIA/OpenShell.

OpenAI. Chatgpt plugins, 2024. https://openai.com/index/chatgpt-plugins/ (accessed 27, August, 2025).

OpenClaw. OpenClaw, 2026a. https://openclaw.ai/.

OpenClaw. Configuration, 2026b. https://docs.openclaw.ai/gateway/configuration (accessed 29, May, 2026).

OpenClaw. Webhooks, 2026c. https://docs.openclaw.ai/cli/webhooks.

PinchBench. PinchBench, 2026. https://pinchbench.com/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. Optimization-based prompt injection attack to LLM-as-a-judge. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2024.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in LLM agents. In *Network and Distributed System Security Symposium 2026*, San Diego, CA, USA, 2026. Internet Society. doi: 10.14722/ndss.2026.230675. URL https://www.ndss-symposium.org/ndss-paper/prompt-injection-attack-to-tool-selection-in-llm-agents/.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for LLM agents. *arXiv preprint arXiv:2504.11703*, 2025.

Shoaib Ahmed Siddiqui, Radhika Gaonkar, Boris Köpf, David Krueger, Andrew Paverd, Ahmed Salem, Shruti Tople, Lukas Wutschitz, Menglin Xia, and Santiago Zanella-Béguelin. Permissive information-flow analysis for large language models. *arXiv preprint arXiv:2410.03055*, 2024.

Georgios Syros, Anshuman Suri, Jacob Ginesin, Cristina Nita-Rotaru, and Alina Oprea. SAGA: A security architecture for governing AI agentic systems. In *Network and Distributed System Security Symposium 2026*, San Diego, CA, USA, 2026. Internet Society.

Lillian Tsai and Eugene Bagdasarian. Conseca: Contextual agent security: A policy for every purpose. In *Proceedings of the 20th Workshop on Hot Topics in Operating Systems*. ACM, 2025. URL https://arxiv.org/abs/2501.17070.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The instruction hierarchy: Training LLMs to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*, 2024.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*. ACM, 2026. URL https://arxiv.org/abs/2503.18666.

Zhun Wang, Vincent Siu, Zhe Ye, Tianneng Shi, Yuzhou Nie, Xuandong Zhao, Chenguang Wang, Wenbo Guo, and Dawn Song. AGENTVIGIL: Automatic black-box red-teaming for indirect prompt injection against LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 23159–23172, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1258. URL https://aclanthology.org/2025.findings-emnlp.1258/.

Simon Willison. The dual llm pattern for building ai assistants that can resist prompt injection, 2023. https://simonwillison.net/2023/Apr/25/dual-llm-pattern/.

Fangzhou Wu, Ethan Cecchetti, and Chaowei Xiao. System-level defense against indirect prompt injection attacks: An information flow control perspective. *arXiv preprint arXiv:2409.19091*, 2024.

Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. An execution isolation architecture for LLM-based agentic systems. In *Network and Distributed System Security Symposium 2025*, San Diego, CA, USA, 2025. Internet Society. URL https://arxiv.org/abs/2403.04960.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM agents by a guard agent via knowledge-enabled reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 2025. URL https://arxiv.org/abs/2406.09187.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Jingwei Yi, Yueqi Xie, Bin Zhu, Keegan Hines, Emre Kiciman, Guangzhong Sun, Xing Xie, and Fangzhao Wu. Benchmarking and defending against indirect prompt injection attacks on large language models. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, 2025. URL https://arxiv.org/abs/2312.14197.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *13th International Conference on Learning Representations, ICLR 2025*, Singapore, 2025. International Conference on Learning Representations. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/5750f91d8fb9d5c02bd8ad2c3b44456b-Abstract-Conference.html.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. RTBAS: Defending LLM agents against prompt injection and privacy leakage. *arXiv preprint arXiv:2502.08966*, 2025.

Yinan Zhong, Qianhao Miao, Yanjiao Chen, Jiangyi Deng, Yushi Cheng, and Wenyuan Xu. Attention is all you need to defend against indirect prompt injection attacks in LLMs. In *Network and Distributed System Security Symposium 2026*, San Diego, CA, USA, 2026. Internet Society. URL https://www.ndss-symposium.org/ndss-paper/attention-is-all-you-need-to-defend-against-indirect-prompt-injection-attacks-in-llms/.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 2025. URL https://arxiv.org/abs/2502.05174.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023. doi: 10.48550/arXiv.2307.15043. URL https://arxiv.org/abs/2307.15043.

Wei Zou, Runpeng Geng, Binghui Wang, and Jinyuan Jia. PoisonedRAG: Knowledge corruption attacks to retrieval-augmented generation of large language models. In *34th USENIX Security Symposium*. USENIX Association, 2025.

Table 2: Defense comparison. ✓ means the approach satisfies the property, × means it does not, and △ means the property is partial or conditional.

Defense	Security			Utility		Deploy.
	Track		Isolate	Agent	Human	
	Internal	External				
Model-based defenses	△†	×	△†	✓	✓	✓
Sandboxing	×	×	△‡	△‡	✓	△*
Dual LLM (Utility)	✓	×	✓	✓	✓	×
Dual LLM (Security)	✓	✓	✓	×	×	×
DUALVIEW	✓	✓	✓	✓	✓	✓

† Model-based defenses rely on model or classifier judgments. Some separate untrusted data within the context but only probabilistically, with false negatives, and none preserve the distinction once data leaves the agent.
‡ Sandboxing security and utility highly depends on policy—e.g., blocking a remote access stops remote IPI attacks it but breaks benign tasks that need network.
* Sandboxing deployability depends on OS support for restricting files, network, and shell access.

A ADDITIONAL DESIGN DETAILS

A.1 DEFENSE COMPARISON

Table 2 compares existing defenses against the goals in §3.1. For security, a defense must track untrusted data (i.e., Track), both inside the agent (i.e., Internal) and in the external environment (i.e., External), and isolate it from the agent's actions (i.e., Isolate). For utility, it must let the agent use untrusted data (i.e., Agent) and keep the environment usable for humans and non-agent programs (i.e., Human). It must also be deployable on existing agent systems.

Model-based defenses. Model-based defenses include model hardening (Wallace et al., 2024; Chen et al., 2025a;b), guardrails (Meta AI, 2025; Chennabasappa et al., 2025; Liu et al., 2025b; Hines et al., 2024; Zhong et al., 2026), and runtime monitors (Zhu et al., 2025; Jia et al., 2025; Li et al., 2025; Chen et al., 2025c; Xiang et al., 2025; Wang et al., 2026). They attempt to track untrusted data within the agent context (Hines et al., 2024; Zhong et al., 2025; Siddiqui et al., 2024) and isolate it from the agent's actions by judging tool results or planned actions (Wallace et al., 2024; Chen et al., 2025a;b). These judgments are only probabilistic, so adversarial inputs often evade them (Zhan et al., 2024; Zhang et al., 2025; Wang et al., 2025; Yi et al., 2025; Shi et al., 2024). They also do not track untrusted data once it leaves the agent into the external environment. In general, they are easy to add to existing runtimes, leaving tool and agent behavior mostly unchanged.

Sandboxing. Sandboxing defenses restrict the resources available to the agent (NVIDIA, 2026); for example, OpenShell (NVIDIA, 2026) confines tool execution with kernel-assisted access control such as Landlock, seccomp, and network policy. Sandboxing enforces resource restrictions rather than data tracking, so it does not track untrusted data. Sandboxing can also block benign tasks that need restricted resources, while allowing them leaves attack paths open. Sandboxing does not harm human utility, but deployment depends on operating-system and runtime support.

Dual LLM pattern. Dual LLM pattern defenses (Willison, 2023; Bagdasarian et al., 2024; Kim et al., 2025; Debenedetti et al., 2025; Costa et al., 2025; Wu et al., 2024; Li et al., 2026) track untrusted data only inside the agent context (§2.2), giving no principled answer for what happens to a symbol once it leaves the agent for the external environment, which leaves two design choices. A utility-oriented design (i.e., Dual LLM (Utility)) desymbolizes the data as it leaves, preserving agent and human utility but losing tracking once data leaves, which exposes the agent to stored IPI. A security-oriented design (i.e., Dual LLM (Security)) instead keeps symbols in the external environment, preserving external tracking. However, remote endpoints receive symbols they cannot use and humans see symbols instead of original data, breaking both agent and human utilities.

DUALVIEW. DUALVIEW tracks untrusted data inside the agent by symbolizing untrusted HumanView result data into AgentView. DUALVIEW keeps tracking untrusted data after it leaves the agent by storing symbols in Agent File System and by re-symbolizing data that returns from HumanView tools or untrusted agents. Because AgentView contains symbols rather than attacker text, these

tracked inputs cannot directly steer tool-call decisions. The agent can still pass symbols to tools or use U-LLM processing, HumanView preserves original data for humans and non-agent programs, and tool hooks let DUALVIEW deploy on existing runtimes.

In the following, we explain how DUALVIEW handles four major tool categories of personal agents.

A.2 NETWORK TOOL DETAILS

The network provides diverse information to agents, allowing them to search the web, retrieve current content, and access web services. At the same time, the network exposes the agent to indirect prompt injection attack vectors.

Network tools in HumanView. Network tools, such as web fetch, web search, and webhooks, use HumanView because remote endpoints and external services use original data rather than DUALVIEW symbols. Before a network tool runs, DUALVIEW desymbolizes the tool arguments so the remote endpoint receives original data. When a network tool returns results, DUALVIEW applies the data trust policy (§3.4) and symbolizes data classified as untrusted in AgentView. By default, network data is untrusted. Origin rules can mark selected network endpoints as trusted. For structured network data, schema rules can classify individual fields as trusted or untrusted when the schema is trusted.

Web fetch and web search. Web fetch takes a URL and returns the URL, title, and content; web search takes a query and returns an array of such results. The data trust policy treats fields that derive from remote content as untrusted, including the title, the content, and post-redirect URL the tool returns, while metadata the tool or agent supplies, such as status code, stays trusted.

Webhook payloads. Webhooks (OpenClaw, 2026c) deliver network data from pre-registered external services to the agent. By default, DUALVIEW treats webhook payloads as untrusted because they arrive from the external network (OpenClaw, 2026b). When the webhook service is trusted and delivers structured data whose trusted and untrusted parts are known (e.g., the provider marks them or the user can discern them), the data trust policy can be configured to keep the trusted fields original and symbolize only untrusted fields. Arbitrary services cannot exploit this because each webhook input port authenticates requests with its own secret shared during registration.

A.3 FILE TOOL DETAILS

File tools (e.g., read or write) let an agent work with files on the user's computer, including files the user already has, files received from others, and downloaded files. They also let the agent save partial work or task results to files, then read those files later to continue the task.

File tools in AgentView. DUALVIEW routes file tool calls to AgentView, where they use Agent File System instead of Human File System. Agent File System exposes the same file paths but keeps untrusted file content as symbols, so DUALVIEW neither symbolizes data read from files nor desymbolizes symbols when writing them. Because attacker-controlled content stays symbolized across a write and a later read, stored IPI is prevented.

Agent File System. Agent File System stores files with original trusted data and symbols for untrusted data. DUALVIEW tracks files modified by the agent using Git and implements AgentView as a separate Git worktree (Git Project, 2026). In AgentView, the agent still refers to files using the same paths it would use without DUALVIEW. Before a file tool runs, DUALVIEW rewrites those paths to the corresponding files in the Agent File System worktree. To avoid tracking the entire file system, DUALVIEW tracks files per *workspace*, rooted at the enclosing Git project or, if none, the parent directory of the accessed file. Appendix B describes how DUALVIEW manages Agent File System using Git in detail.

Human File System. Humans and non-agent programs use Human File System at ordinary file paths and see original data rather than symbols. They can read and write files as they normally would without DUALVIEW. DUALVIEW tracks Human File System and Agent File System in the same Git repository, while Human File System remains the main worktree.

Syncing files. DUALVIEW synchronizes Agent File System and Human File System around file tool calls so file tools see changes made by the user or non-agent programs, and humans and non-agent programs see the agent's file writes as original data. Before an agent calls a file tool, DUALVIEW copies relevant Human File System changes into Agent File System. DUALVIEW treats those human-side changes as trusted by default, so it copies them without symbolizing them. After the file tool returns, DUALVIEW copies the agent's changes from Agent File System into Human File System and desymbolizes symbols so the user sees original data. After each sync step, DUALVIEW commits both file systems with Git, so it copies only the files that changed since the previous commit and can revert to an earlier synced state when needed. We further evaluate concurrency handling in §D.7 and detail it in Appendix B.

Untrusted files. While DUALVIEW treats files from the Human File System as trusted by default, the data trust policy can override this for files or directories that contain untrusted data. For example, a user can download files from external services, such as emails or web scraping results, in a single directory. To protect agents from untrusted files, the data trust policy can mark that directory as untrusted. DUALVIEW symbolizes matching files in AgentView when the agent initializes and first loads the policy, or when the policy changes, so untrusted file content does not enter AgentView as plain text.

Human file edits. Human edits copied from Human File System are trusted because the user is trusted in the threat model. If a human edit overlaps a line that Agent File System stores as a symbol, DUALVIEW treats the human-written line as original trusted data and keeps the unchanged symbolized lines around it as separate symbols.

Example: File. Figure 4 illustrates this process. A user keeps a note in the Human File System. When the agent calls a file tool to read the note (A-①), DUALVIEW first copies the file into the Agent File System (A-②), and the file content returns to the agent (A-③). The agent then writes `$ws01.content.summary`, the web search summary symbol introduced in the previous network example (Figure 3), into the note (B-①). DUALVIEW copies the update to the Human File System and desymbolizes the symbol (B-②), and the write returns to the agent (B-③). The user can then read the web search summary as original data.

Later, the user edits the file in the Human File System (C-①). When the agent reads the file again (C-②), DUALVIEW copies the human edit into the Agent File System (C-③). If the human edit overlaps content represented by a symbol in the Agent File System, DUALVIEW treats the human-written line as original trusted data and keeps the unchanged symbolized lines around it as separate symbols (C-④).

A.4 SHELL TOOL DETAILS

Personal AI agents often connect to the shell so they can use existing command-line tools during a task. Through shell tools such as `exec`, an agent can search and transform local files, fetch network data, and run third-party CLIs. For example, `grep` reads local files, `curl` fetches data from network endpoints, and CLI programs such as `gws` (Google Workspace, 2026) and `gh` (GitHub, 2026) access web services.

Shell tools in both views. DUALVIEW exposes two shell tools to the agent. AgentShell is the shell tool for AgentView. It runs commands on Agent File System with network access disabled. HumanShell is the shell tool for HumanView. It runs commands in the normal shell environment, with Human File System and network access enabled. The agent chooses AgentShell or HumanShell for each command, guided by the system prompt (Appendix B).

A shell command can run in AgentView using AgentShell when trusted data and the symbols that replace untrusted data are enough to complete it. For example, `ls`, `cat`, and `grep` with a trusted keyword can run on symbolized files, where `grep` still matches the keyword against the trusted text while untrusted data stays symbolized. Commands that need original data, network access, or the normal shell environment can run in HumanView using HumanShell, such as `curl`, `gws`, or `git push`. When the agent chooses HumanShell, DUALVIEW desymbolizes the command and symbolizes the result after the shell returns.

Choosing the less suitable shell can reduce agent utility but has no security impact. AgentShell for a command that needs original data or network may fail, but the agent may retry with HumanShell, while HumanShell for a command that AgentShell could have run still completes but has its output conservatively symbolized. Either way, untrusted data remains symbolized in AgentView.

AgentShell. As with file tools (Figure 3.3), AgentShell lets commands use the same file paths that the human user uses by mounting Agent File System at the original file paths. Commands then read symbolized file content from Agent File System, while DUALVIEW uses a network namespace to disable network access. To prevent writing symbols to untracked locations, DUALVIEW limits AgentShell writes to Git-tracked workspaces.

HumanShell. HumanShell runs commands on Human File System in the normal shell environment, with network access enabled. Before the command runs, DUALVIEW desymbolizes symbols in arguments so that the shell receives original data. As HumanShell can read original file content and network data that may contain untrusted data, DUALVIEW conservatively treats its output as untrusted by default and symbolizes it before returning it to AgentView. For structured command output, the data trust policy (§3.4) lets DUALVIEW keep trusted fields as original data and symbolize untrusted fields, rather than treating the output as a single untrusted string.

Untrusted data execution. Letting the agent desymbolize untrusted data into HumanShell is a deliberate tradeoff between security and utility, since some commands need original data and DUALVIEW keeps that capability rather than withholding it. This tradeoff does not weaken the core IPI guarantee, since attacker text still cannot steer the agent into issuing a command; untrusted data reaches T-LLM only as symbols. However, the agent can still misuse data that it legitimately desymbolizes, and the most dangerous case is running untrusted data as a command or code. This risk arises in HumanShell, which desymbolizes symbols before running a command. As an additional measure, DUALVIEW inspects each HumanShell command, and when a symbol would be used as a command or code, it withholds the execution and asks the user for approval. §3.4 describes this policy check.

Example: Agent Shell. The left side of Figure 5 shows an AgentShell example. The agent runs `grep TODO notes/` in AgentShell (A-①). Before the command runs, DUALVIEW copies relevant Human File System changes into Agent File System (A-②). AgentShell then reads the Agent File System and returns the line containing `TODO` (A-③). The result returns to the agent without another symbolization step. After the command returns, DUALVIEW runs `git diff` to find the files that the command changed in the tracked workspace, so it does not need to track modifications while the command runs. If there are any changes, DUALVIEW copies them from Agent File System to Human File System and desymbolizes them (A-④).

Example: Human Shell. The right side of Figure 5 shows an example of using HumanShell. `gws email +triage` fetches unread Gmail messages. The agent chooses HumanShell for this command because it must access Gmail through the network. Before the command runs, DUALVIEW checks the data usage policy and desymbolizes the command arguments (B-①). The command then fetches Gmail messages as original data over the network (B-②). For commonly used commands that return structured data with a fixed schema, the data trust policy lists the trusted and untrusted output fields, and DUALVIEW parses the output and symbolizes only the untrusted fields (B-③). For `gws email +triage`, it keeps fields such as `count` and `id` trusted and symbolizes untrusted email fields such as `from`, `subject`, and `body`. The symbolized result then returns to the agent (B-④).

A.5 INTER-AGENT COMMUNICATION TOOLS

Personal AI agents often run separate agents for different messaging channels, webhooks, and cron inputs, and communicate through inter-agent tools that send a message, receive a reply, or read another agent's history. For example, OpenClaw runs one agent per messaging channel (e.g., a Slack or Telegram channel) and provides `session_send` and `session_history` tools for them to exchange messages and read each other's history. DUALVIEW routes these tools by whether the other agent is trusted.

Inter-agent tools in AgentView. An agent is trusted when it runs in AgentView and receives only trusted input, such as an agent attached to the user's private channel. Because DUALVIEW keeps

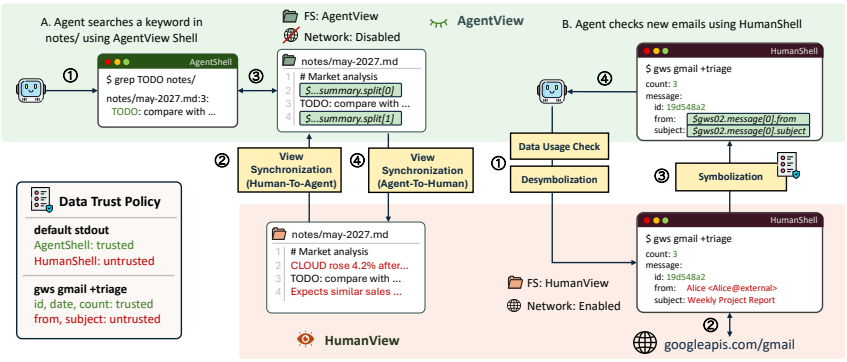


Figure 5: Shell execution in DUALVIEW. AgentShell lets commands run against Agent File System with network access disabled, so their output contains trusted data and symbols. HumanShell lets commands run in the normal shell environment with Human File System and network access enabled. DUALVIEW symbolizes shell output before returning it to AgentView.

symbol identifiers valid across agents, messages between trusted agents pass without desymbolization; both agents use trusted data and the same opaque symbols.

Inter-agent tools in HumanView. Users mark an agent as untrusted when it directly receives external or unknown-origin input as original data, such as a public-channel agent. DUALVIEW uses HumanView for it, desymbolizing symbols the agent must receive as original data and symbolizing untrusted data in messages returned to AgentView. This protects trusted agents reading from untrusted ones, but restricting the untrusted agents themselves requires separate sandboxing or reduced tool access.

B IMPLEMENTATION

OpenClaw integration. DUALVIEW is implemented with event hooks and custom tools, a mechanism supported in many personal agents (e.g., Claude Code (Anthropic, 2026) and Hermes Agent (Nous Research, 2026)). This hook-based design does not require changing the AI model, the agent main logic, or tool implementations, which makes it deployable on various personal agents. As in prior Dual LLM pattern defenses (Kim et al., 2025; Costa et al., 2025; Debenedetti et al., 2025), DUALVIEW also registers `inspect_symbol` as a custom tool for U-LLM processing, which takes the symbols, the processing instruction for U-LLM, and an output schema for the result. DUALVIEW checks the U-LLM result against the schema, returns the symbolized result to AgentView when it matches, and otherwise returns an error without exposing any untrusted data.

Hook points. OpenClaw hooks let DUALVIEW mediate the agent lifecycle at initialization, tool execution, and human-facing replies (Figure 6). The pre-tool hook handles the work after T-LLM selects a tool but before the tool runs: for AgentView tools it copies HumanView file changes into the Agent File System, and for HumanView tools it checks the data usage policy and resolves the symbols the tool needs as original data. The post-tool result hook applies only to HumanView tools, which can return original untrusted data that must be symbolized before T-LLM reads it. DUALVIEW uses its `transform_tool_result` hook to apply the data trust policy to web results, HumanShell stdout, webhook payloads, and other HumanView tool results before T-LLM reads them; OpenClaw v2026.3.12 did not support this hook, so DUALVIEW added it, and the latest OpenClaw provides it natively as tool-result middleware. Final assistant messages use a separate delivery hook, since no tool call occurs after T-LLM generates the final text. DUALVIEW uses `message_sending` to resolve symbols only in the agent response that goes to the CLI, webchat, or channel, so the transcript keeps symbols while the human-facing message contains original data; OpenClaw v2026.3.12 did not deliver an edited response to the messenger channel, so DUALVIEW patched it.

Symbol table. DUALVIEW stores the symbol table in SQLite and shares it across all agent sessions. The symbol table maps each symbol to its original data and its provenance, allowing DUALVIEW to resolve symbols when a symbol is passed to HumanView. DUALVIEW resolves a symbol only on

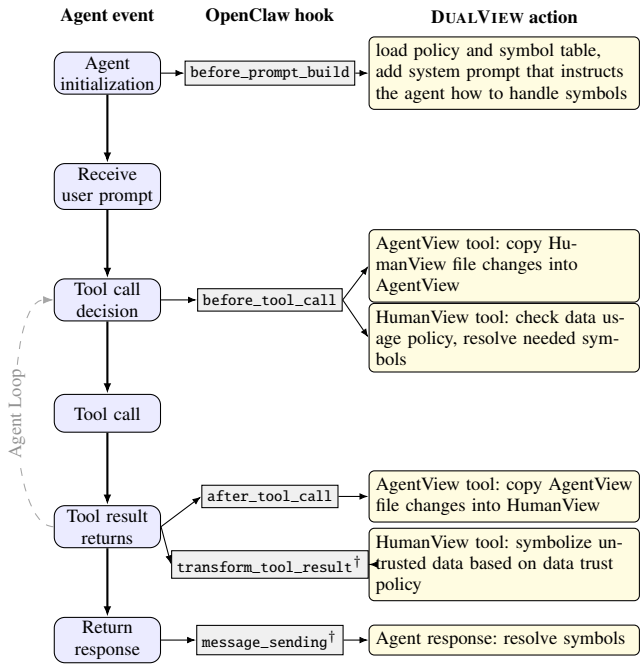


Figure 6: OpenClaw hook points that DUALVIEW uses across the agent's events. `before_tool_call` and the two result hooks branch by whether the tool runs on AgentView or HumanView. DUALVIEW changes the two hooks marked †.

an exact match of its full token, so a command or edit that splits, truncates, or otherwise mangles a symbol leaves a fragment that DUALVIEW does not resolve. Such a fragment never reveals the original untrusted data, so resolution fails closed; it can leave a broken token in human-facing output and lower human utility, a case we did not observe in our evaluation. Each entry stores a symbol with the original data and provenance metadata. The provenance metadata comes from trusted data (e.g., the originating tool, the field path, and the data origin). DUALVIEW can extend the data usage policy with this provenance in the future, for example to block symbols of a given origin from sensitive tool arguments, which prevents taint-style vulnerabilities (Liu et al., 2025a).

System Prompt. DUALVIEW appends a system prompt to agent context that explains how to handle its symbolized tool results and shell tools. It instructs the model to treat symbol tokens as opaque handles and pass them through unchanged, since they are resolved to their real values only at delivery time, and to call the `inspect_symbol` tool whenever it needs a derived result such as a summary, an extraction, or a transformation of symbolized data. It also documents the Data Trust Policy together with its `policy_add` and `policy_del` tools, and the two shell modes (i.e., shell tools). Default exec resolves symbols and marks the output untrusted, while restricted exec (`RESTRICTED=1`) runs sandboxed and returns trusted output. The full prompt text is in our released code.

Filesystem. DUALVIEW tracks the files the agent modifies with Git and keeps the Agent File System and Human File System as two Git worktrees (Git Project, 2026) (Figure 3.3). Figure 7 shows an example layout of file systems. The original workspace directory is the Human File System, and DUALVIEW keeps its own Git database and the Agent File System outside that directory under `~/.dualview/workspaces/`, where it canonicalizes and percent-encodes the root path into a single directory name. Because DUALVIEW keeps its Git database outside the workspace, its Git use stays transparent to the user, so ordinary Git commands in the workspace work seamlessly. DUALVIEW starts tracking a new workspace on demand when the agent first writes a file with a tool such as `write` or `edit`. If the file is in a Git project, the workspace root is the enclosing project directory, otherwise it is the parent directory of the file. Unlike a file tool, a shell command can write to paths DUALVIEW cannot anticipate, so DUALVIEW does not open a new workspace for AgentShell and instead confines its writes to the tracked workspaces, keeping every write inside a worktree that the post-tool sync reconciles.

`/home/user/apple/banana/`	workspace
ordinary workspace files	*HumanView files*
`.git/`	ordinary Git, optional
`~/.dualview/workspaces/`	DUALVIEW file-system management
`%2Fhome%2Fuser%2Fapple%2Fbanana/`	encoded workspace path
`repo.git/`	DUALVIEW Git
`agentview/`	*Agent File System* for the workspace
`conflicts.log`	sync conflict log

Figure 7: Example file-system layout under DUALVIEW.

Table 3: PinchBench file paths marked as untrusted. These patterns match 49 of 98 input file paths, and 42 of the 147 PinchBench tasks read at least one of them.

Category	**Untrusted Files**
Email and inbox	`emails/*`, `inbox/*`
External documents and reports	`ai_blog.txt`, `GPT4.pdf`, `openclaw_report.pdf`, `research/*`, `sample_contract.pdf`, `school-calendar.pdf`, `subway_map.md`, `vulnerability_scan.json`
Logs and request metadata	`access_events.csv`, `apache_error.log`, `auth.log`, `hdfs_datanode.log`, `linux_syslog.log`, `mapreduce.log`, `nginx_access.log`, `syslog.log`

DUALVIEW handles potential file write race conditions in two ways. First, a race between two agents that sync the same root is handled by serializing sync commits with a workspace-scoped file lock. Second, a race between a human write and a sync, which the lock does not prevent, is handled by detecting the changed file hash, skipping the affected file so DUALVIEW never overwrites the human's change, and appending a record to the conflict log. DUALVIEW currently only reports the conflict through the log and leaves the resolution to the human. We evaluated file-system synchronization under concurrent access in §D.7.

Claude Code integration. We implemented a prototype of DUALVIEW as a Claude Code plugin using hooks and additional MCP tools (Anthropic, 2026). The `PreToolUse` and `PostToolUse` hooks mediate the inputs and outputs of native `Read` and `Write`, `SessionStart` adds symbol instructions, and `MessageDisplay` resolves user responses. Additional MCP tools provide editing, shell execution, web fetch, and U-LLM processing (Model Context Protocol, 2025). The plugin reuses DUALVIEW's policies, file-system views, shell modes, and synchronization.

LangChain integration. We implemented a prototype of DUALVIEW as LangChain middleware around model and tool calls (LangChain, 2026). The middleware adds symbol instructions before model calls, mediates tool inputs and outputs before results enter the message history, and resolves final responses. It reuses DUALVIEW's policies, file-system views, shell modes, and synchronization without changing LangChain's agent loop.

C DUALVIEW POLICY

Policy specification. Table 5 summarizes the prototype data trust policy that DUALVIEW uses for OpenClaw built-in tools and webhook receivers. Fields listed as trusted pass through to the agent as original data. Fields listed as untrusted are symbolized before the agent reads them. Webhook fields omitted from an endpoint schema are untrusted by default.

Examples. Table 6 shows how the two DUALVIEW policies apply at different boundaries. The upper block applies the data trust policy after a HumanView tool returns. The rule on the left names the trusted and untrusted fields or origins, and the result on the right is what T-LLM sees after DUALVIEW replaces untrusted fields with symbols. The lower block applies the data usage policy before HumanShell runs a command. Commands with no symbol in executable text run without extra approval, while commands or script files that would execute symbolized data require human approval.

Table 4: Sensitive PinchBench output directories made read-only for the OpenShell utility evaluation. 60 of 147 tasks write to these directories.

Category	Restricted Path
Email replies	`output/email_replies/`
Security, contract, and triage reports	`output/security/`
Meeting decisions, recommendations, and summaries	`output/decisions/`
CI, Kubernetes, logs, and service actions	`output/operations/`
Research, finance, procurement, and stock outputs	`output/business/`

Table 5: Prototype data trust policy specification. The policy uses the same trusted and untrusted field distinction for built-in tools, structured shell output, webhook receivers, and inter-agent communication.

Tool or source	Trusted fields	Untrusted fields
`web_fetch`	`url`, `status`, `contentType`, `extractMode`, `extractor`, `fetchedAt`, `tookMs`, `truncated`, `length`, `rawLength`, `wrappedLength`, `externalContent`, `warning`	`finalUrl`, `title`, `text`
`web_search`	`query`, `provider`, `count`, `tookMs`, `externalContent`	Per-result `title`, `description`, `url`, `published`, `siteName`
`exec`	Fields declared by per-skill output schemas, such as local counters, identifiers, and schema tags	Default stdout, plus remote or user-controlled fields declared by per-skill output schemas
`read`	File content from AgentView, where untrusted content is already symbolized	None
`write` and `edit`	Tool status and other local metadata	None
`process`	Write and lifecycle actions, `list`, and trusted local metadata	Read actions such as `poll` and `log`, plus unknown actions
`inspect_symbol`	Validated output after U-LLM processing and re-symbolization	None
Webhook receivers	Endpoint-specific schema fields such as verified source metadata, event type, and local delivery identifiers	Endpoint-specific schema fields such as message text, issue comments, form bodies, and fields omitted from the schema
Image and media tools	Local tool metadata and URL-keyed identifiers	Remote captions, descriptions, or other service-controlled text when present
Inter-agent session tools	Local conversation identifiers and private context metadata	Message text and tool-result text from untrusted agent contexts
Other local built-in tools	`memory_search`, `memory_get`, `session_status`, `canvas`, `nodes`, `cron`, `tts`, `message`, and `claude_code_*` results that contain local metadata only	Remote or externally controlled text when a deployment configures such fields

PinchBench policy. Of the 147 PinchBench tasks, 114 read a file as input and 121 write a file as the goal. For example, an email reply task reads an inbox under `emails/*` or `inbox/*` and writes the reply as a file in the workspace. For DUALVIEW and the Dual LLM pattern, we marked untrusted input files so that their content is symbolized (Table 3); examples include `research/*`, `GPT4.pdf`, and `auth.log`, and 42 tasks read at least one such file. For Sandboxing, we placed privileged output files, such as email replies and business outputs, in dedicated directories and denied writes there, and minimized network access to prevent data leakage and arbitrary actions (Table 4); the blocked routes cover 60 tasks.

C.1 DATA TRUST POLICY DETAILS

Schema rules. The data trust policy stores a schema rule for each tool that marks returned data as trusted or untrusted, individual fields for structured results and the whole value for unstructured results. A schema rule treats data as trusted when the tool or runtime generates it, such as a status code, a count, or an argument the agent itself supplied, or when it comes from a trusted source such as ordinary local file contents. A schema rule treats data as untrusted whenever its value derives from remote content, such as a web page body, an email from an arbitrary sender, a public-channel message, or a webhook field from an external service.

Table 6: Concrete examples for the two policy types. The data trust policy block shows schema rules and user-supplied origin rules. The data usage policy block shows executable command patterns and command rewriting rules. The dark red text marks symbolized fields or command and code text that HumanShell would execute, and a command with no symbol in executable text runs without human approval.

Policy rule	Tool result or command		DUALVIEW output
Data trust policy *(checks tool output)*			

Schema rules
Trusted
`web_fetch.url`
Untrusted
`web_fetch.text`
	`web_fetch(url) = {`
	`  url=https://reports.example/q1,`
	`  text="Quarterly report"`
	`}`
		⇒	`web_fetch(url) = {`
			`  url=https://reports.example/q1,`
			`  text=$web1.text`
			`}`

Origin rules
Trusted
`api.github.com/*`
	`web_fetch(https://api.github.com/zen) = {`
	`  content="Design for failure."`
	`}`
		⇒	`web_fetch(https://api.github.com/zen) = {`
			`  content="Design for failure."`
			`}`

Untrusted
`imports/*.csv`
	`read(imports/q1.csv) = {`
	`  content="name,amount..."`
	`}`
		⇒	`read(imports/q1.csv) = {`
			`  content=$read1.content`
			`}`

Untrusted
`agent:public-chat`
	`session_history(agent:public-chat) = {`
	`  message="check this link"`
	`}`
		⇒	`session_history(agent:public-chat) = {`
			`  message=$session1.message`
			`}`

Data usage policy *(checks commands sent to HumanShell)*			
Executable Command Pattern	`exec(git status)`	⇒	No human approval required
`<symbol>`	`exec($web1.text)`	⇒	`$web1.text` needs human approval for execution
`python -c <symbol>`	`exec(python -c $web1.text)`	⇒	`$web1.text` needs human approval for execution

Command Rewriting Rule
`python <file>`
transform →
`python -c <file content>`
	`exec(python fix.py)`
	transform →
	`exec(python -c "...$web1.text...")`
		⇒	`$web1.text` needs human approval for execution

DUALVIEW ships default schema rules for 21 built-in OpenClaw tools (Appendix C), derived by manually analyzing their implementations and result schemas. Data not listed in the policy is untrusted by default.

Origin rules. Origin rules classify data by its source, such as a network endpoint, file path, other agent, or webhook input port, and each user chooses which sources to trust. Network data and email content are untrusted by default, and users can mark specific endpoints, senders, or domains as trusted. For file tools, file paths act as origins, and while DUALVIEW treats local files as trusted, users can mark downloaded or imported files and directories as untrusted. For inter-agent communication, the other agent is the origin, and a user can mark an agent untrusted when it directly receives external input as original data, so DUALVIEW symbolizes that agent's messages before returning them to AgentView. For webhooks, a schema rule on the named input port marks payload fields as trusted or untrusted. Users can add origin rules directly or through the agent (§C.3).

C.2 DATA USAGE POLICY DETAILS

Executable Command Patterns. Executing attacker-controlled commands on the user's computer can let attackers compromise the user's environment, even if the attacker cannot directly steer the agent through an IPI attack. In HumanShell, DUALVIEW desymbolizes a symbol in a command into original data in the normal shell environment, which can lead to arbitrary command execution. To prevent this, the default data usage policy lists executable command patterns for HumanShell and requires user approval when a symbol is used as a command. The patterns include a symbol-

only command (i.e., `'$sym'`) and a command that passes a symbol to an interpreter option (e.g., `python -c '$sym'` or `bash -c '$sym'`). Note that in AgentShell, DUALVIEW does not desymbolize symbols, so the shell cannot execute the original data that a symbol denotes.

Command Rewriting. DUALVIEW further prevents HumanShell from executing files that contain untrusted data by rewriting commands. For example, `python fix.py` runs the Python script `fix.py`, which may contain untrusted data as code. Executable command patterns would miss this command because no symbol appears in the command itself. To detect such cases, DUALVIEW maintains command rewriting rules that rewrite file execution commands into equivalent inline commands. For instance, `python <file>` is rewritten as `python -c <file content>`. If the file content contains a symbol, the resulting inline command matches an executable command pattern.

Command rewriting rules cover only the listed script execution patterns, so other shell commands can still read and execute a file in a way not listed by the policy. Detecting all untrusted data execution would therefore require dedicated monitoring, such as system call hooks that check whether a command reads a symbol from a file before executing it, which we leave as a current limitation. As the agent automatically moves data across different sources and sinks, the rationale for which uses of untrusted data are unsafe, beyond running it as code, is still lacking. Because AgentView already tracks untrusted data at the symbol level, it provides a useful basis for richer data usage policies.

C.3 POLICY UPDATES

DUALVIEW lets users update both policies. It stores them in a YAML file that the user can edit directly, and it also provides policy tools (`policy_add`, `policy_list`, and `policy_del`) so the agent can add, list, and delete entries when the user authorizes a change. Although DUALVIEW exposes these policy tools to the agent, the agent makes its tool-call decisions in AgentView, where untrusted data appears only as symbols, so untrusted data cannot direct the agent to update a policy. An agentic policy update that best balances security and utility requires an orthogonal investigation that we leave to future work.

D ADDITIONAL EVALUATION DETAILS

D.1 EVALUATION SETUP

All experiments used the same benchmark user tasks and model settings, while the execution environment changed with the defense under test. The security benchmark is a custom IPI benchmark (§4.1), and the utility benchmark is PinchBench (§4.2). We mocked the web, email, and other external services locally behind a web proxy that returned deterministic responses for every defense. The security benchmark ran fully on this mocked network environment, while PinchBench used both the mocked services and real web tools, such as web search and web fetch. We also evaluated environment sync under concurrent human access. We evaluated Claude Haiku 4.5 (`claude-haiku-4-5-20251001`) and Claude Sonnet 4.6 (`claude-sonnet-4-6`). In DUALVIEW and Dual LLM, T-LLM and U-LLM use the same model.

The baseline is plain OpenClaw, which prepends a built-in security warning prompt before some external input, such as web fetch and webhook content. We used Llama Prompt Guard 2 (Meta AI, 2025) to classify each tool result on its own and withheld it from the agent when classified unsafe. We used LlamaFirewall AlignmentCheck (Chennabasappa et al., 2025; Chen et al., 2025b) to judge each pending tool call against the full conversation trace and blocked the call when classified unsafe. For sandboxing, we used OpenShell (NVIDIA, 2026) with a restrictive policy to mitigate the harmful consequences of an IPI attack, preventing data exfiltration and arbitrary actions. The policy allows limited network access, such as LLM APIs and package indexes (e.g., `pypi.org`), and read-only access to the file system, while blocking sensitive directories. We assume the utility-oriented Dual LLM (Dual LLM (Utility) in Table 2), as existing Dual LLM defenses resolve symbols when data leaves the agent (Willison, 2023; Bagdasarian et al., 2024). We implemented it by disabling DUALVIEW's environment-level untrusted data tracking, namely by not providing the Agent File System and AgentShell. It still supports agent context-level untrusted data tracking and isolation, as described in §2.2. It resolves every symbol at the tool call, including file writes, so a file stores

original data, and without environment-level tracking it reads that file back as trusted data without symbolizing it.

D.2 Security Evaluation Methodology

We built a custom security benchmark to evaluate instruction-injection IPI attacks in OpenClaw, as existing IPI attack benchmarks (Debenedetti et al., 2024; Zhan et al., 2024; Zhang et al., 2025; Yi et al., 2025) do not support OpenClaw. The benchmark consists of three injection vectors that carry the attack payload. Web content and an email body are the two immediate IPI vectors, where the agent reads the payload and acts on it during the current task. A local file is the stored IPI vector, where we assume the payload was already written into the file by a prior agent execution. Each injection vector has 10 user tasks, each instructing the agent to read that vector and complete a benign goal. We ran every task under three attacker goals, command execution, data exfiltration, and destructive file writes. The benchmark therefore comprises 90 attack cases (3 injection vectors $\times$ 10 user tasks $\times$ 3 attacker goals), and we repeated each run three times.

For each trial, we measured attack success by whether the agent carried out the attacker-injected command. For DualView, we also audited the T-LLM transcript and Agent File System reads and confirmed that original attacker text never appears where T-LLM can read it, so DualView blocks IPI by isolation rather than by the model resisting the injection.

D.3 Adaptive Attack Details

Following AutoDojo (Ma et al., 2026), an adaptive attack framework against agent defenses, we evaluated an adaptive attacker with knowledge of DualView's existence and source code. For each injection vector, we selected an attack case that achieved 100% ASR in the Baseline setting. Assuming a remote adversary (§2.3), the attacker could only change the untrusted payload and observe whether the attack succeeded. For each injection vector, we ran eight iterations. In each iteration, Claude Opus 5 generated up to eight payloads based on earlier attack outcomes, and we tested the payloads against DualView running with Claude Haiku 4.5.

The adaptive attack evaluation found one implementation bug. An email payload made attacker-controlled output from a successful `gws` command appear to be a shell error, and DualView incorrectly treated the command as having failed. To preserve utility, DualView exposes shell error messages from `gws` commands to T-LLM, so this bug exposed an attacker-controlled error message to T-LLM. We fixed DualView so command output could not be interpreted as a shell error. After the fix, adaptive attacks achieved 0% ASR against DualView across all three injection vectors, and no attacker-controlled instruction reached T-LLM.

D.4 Utility Evaluation Methodology

To evaluate the utility of DualView, we used PinchBench (v2.0.0) (PinchBench, 2026), an OpenClaw utility benchmark built from realistic daily usage of OpenClaw agents. PinchBench covers diverse tasks that involve file access, network tools, and shell usage. We ran the benchmark under plain OpenClaw (i.e., the baseline), DualView, and other defenses. PinchBench models most real-world services as file operations, where a task reads input files and writes its result as a file, alongside real web fetch and mock web fixtures. To evaluate each defense fairly on these tasks, we classified input file trust and output file sensitivity from the task semantics and configured every defense for the strongest security under this classification. We further detail the file classification in Appendix C.

For agent utility, we measured the PinchBench task success rate, token overhead, cost, and latency. A task succeeds when the final file system and agent's response satisfy the benchmark's evaluation criteria, which includes both rules and LLM-based judge to evaluate the text quality. We report token overhead as the geometric mean of per-task ratios relative to the baseline, latency as geometric mean task execution time, and total cost in US dollars. We ran the utility benchmark once per defense and model due to high LLM API costs, but its large benchmark size (i.e., 147 tasks) provides a sufficient number of tasks to show the utility trend across defenses.

Table 7: PinchBench failure breakdown for DUALVIEW, counting task failures and ≥ 0.25 score drops from the baseline due to symbolizing untrusted data.

Model	Failed reason	Tasks
Haiku	Uses an invalid U-LLM result	9/147
	Cannot read the symbolized data and gives up	2/147
Sonnet	U-LLM times out on large untrusted data	3/147
	Repeatedly calls HumanShell for the original value, timing out	4/147
	Repeatedly calls U-LLM for the original value, timing out	6/147
Qwen3.5-9B	Cannot read the symbolized data and gives up	12/147
	U-LLM times out on large untrusted data	2/147

Table 8: Review load on PinchBench. DUALVIEW defers untrusted data execution to the user, while the guardrails block tasks outright and remove legitimate work. Other defenses affect no tool calls.

Defense	Model	Tasks	Tool calls
DUALVIEW	Haiku	0/147	0/1,631
(Assume human approval)	Sonnet	1/147	1/1,903
Input guardrail	Haiku	6/147	12/859
(Block tool result)	Sonnet	6/147	12/735
Output guardrail	Haiku	2/147	2/815
(Block tool call)	Sonnet	3/147	11/827

D.5 ADDITIONAL UTILITY RESULTS

In general, tasks failed when the model could not correctly validate the original data behind symbols. The specific failure reason differed by model (Table 7). Haiku used the U-LLM result even when wrong, while Sonnet timed out reaching the original value behind a symbol through repeated U-LLM or shell calls. Qwen3.5-9B also struggled to use symbolized data. It either gave up or repeatedly called U-LLM or HumanShell without producing the requested output, and U-LLM sometimes timed out on large untrusted data.

DUALVIEW detects untrusted data execution by monitoring symbol usage (§3.4) and raises a review event for the user, but this rarely happened on PinchBench (Table 8). The only trigger, on one Sonnet task, was a benign false positive where the agent ran an untrusted log through a Python interpreter while analyzing it. The input and output guardrails instead raised far more false positives, blocking legitimate tasks even with no attack present, so DUALVIEW imposes fewer human reviews.

Conservative policies. To evaluate policies that prevent data-flow attacks, we compare two data trust policies and three data usage policies on PinchBench with Claude Sonnet 4.6. The default data trust policy marks 49 of 98 files as untrusted, while the all-files policy marks every file as untrusted. The default data usage policy checks executable command patterns in HumanShell, while All HumanShell and All tools require approval for every symbol use in HumanShell or any tool input, respectively.

Assuming the user approves every review, the all-files policy reduces utility from 82.1% to 69.7%. The decrease results partly from U-LLM timeouts on files larger than 200 KB, since U-LLM receives the entire file without shell access. Under the most conservative combination, 78 of 1,648 tool calls (4.73%) across 44 tasks require review, compared with 26 of 1,211 calls (2.15%) across 15 tasks under selective files and All tools. Thus, even the most conservative policies require review for fewer than 5% of tool calls.

D.6 COST AND EXECUTION TIME

DUALVIEW added 43.9% overhead on Haiku and 68.3% on Sonnet over baseline (Table 1) from isolated U-LLM processing, growing on Sonnet as it repeatedly invoked U-LLM to reach original

Table 9: Data trust and data usage policy comparison on PinchBench with Claude Sonnet 4.6. Review load reports affected tasks and tool calls requiring approval.

Data trust policy	Data usage policy	Utility	Review tasks	Review tool calls
Selective files (Default)	Executable patterns (Default)	82.1	1/147	1/1,211
	All HumanShell	82.1	6/147	11/1,211
	All tools	82.1	15/147	26/1,211
All files	Executable patterns (Default)	69.7	6/147	8/1,648
	All HumanShell	69.7	8/147	15/1,648
	All tools	69.7	44/147	78/1,648

data. Total cost increased by 49.1% on Haiku and 94.5% on Sonnet, while geometric-mean latency increased by 53.3% and 71.4%, respectively. On Qwen3.5-9B, token overhead was 77.0%, total cost increased from $5.65 to $7.61, and geometric-mean latency from 66.0 to 132.1 seconds. Sandboxing showed the same pattern on Sonnet, which spent even more tokens searching for workarounds to blocked actions. On Sonnet, Sandboxing increased token use by 78.0% but cost by 29.2%, as its repeated workarounds added mostly cache-read tokens. Clearly conveying each defense's restrictions to the model would likely improve agent utility and lower cost. This overhead can hinder deploying DUALVIEW in cost-sensitive settings, so reducing it is important future work, with promising directions such as caching, reusing, or batching U-LLM requests, and using a smaller, cheaper model for U-LLM, which only summarizes, extracts, or transforms bounded inputs.

D.7 CONCURRENCY ROBUSTNESS

We tested 138 cases in which the agent and a human concurrently performed file operations affecting Human File System in HumanView. We varied the timing of concurrent human access during DUALVIEW environment sync and the file operation type, including writes, renames, deletes, and symbolic link creation. The cases covered trusted and untrusted files that either already existed or were newly created during the test. We paused DUALVIEW during AgentView and HumanView synchronization, used a separate process to apply the human operation directly to Human File System, resumed the sync, and finally read the affected file in the agent.

In all tested cases, DUALVIEW prevented original untrusted data from appearing in AgentView. It also ensured that successful agent and human file operations either took effect in the file system or were preserved with a conflict log entry for human resolution, while invalidated operations failed without creating a commit. These results show that environment sync between Agent File System and Human File System remains robust under concurrent file operations.