



AgentScope: Observability for AI Coding Agents

Dr. K. Ravi Kiran Associate Professor¹, Naikoti Sree Vignesh² Aishik Biswas³ Rishwanth Reddy⁴

Dept. of Information Technology Gokaraju Rangaraju Institute of
Engineering and Technology Hyderabad, India

Abstract- : AI coding assistants such as Claude Code, Cursor, and Gemini CLI have become a normal part of how software teams write code. They are fast and helpful, but they also bring a new problem. Teams cannot easily see what these tools are actually doing inside a project. This missing visibility makes it hard to keep the work safe, follow company rules, and understand how much the tools really help. This paper presents AgentScope, a platform that gives full observability into AI assisted coding. AgentScope records every important action during a coding session, such as the developer's prompts, the AI's responses, tool calls, terminal commands, file access, and code changes, and it sends all of them as a single stream to a central dashboard. From this dashboard, engineers and security teams can replay a session step by step, search old sessions, watch how the tools are used, and measure productivity. The system also raises alerts when it notices risky actions, for example exposing a secret, opening a sensitive file, or running a suspicious command. Privacy is kept in mind by removing sensitive data before storing, recording only with the user's consent, and keeping all data inside the organization. AgentScope is built using Next.js, Go, PostgreSQL, WebSockets, and Docker, and it collects agent activity through hooks and the OpenTelemetry standard.

Keywords- : AI Coding Agents; Observability; Software Supply Chain Security; Developer Security; Session Replay; OpenTelemetry.

I. INTRODUCTION

AI coding assistants are everywhere now. Developers use them to write new code, fix bugs, and explain parts of a project they do not understand. These tools save a lot of time, so many teams have started using them every day. The problem is that most teams add these tools without any way to see what happens behind the scenes. When a developer works with an AI agent, many small actions happen very quickly. Prompts are sent, files are opened, commands are run, and code is changed. This missing



visibility causes three real problems. The first is security. An AI agent might read a file that holds passwords, or a developer might paste a secret key into a prompt without thinking. The second is compliance. Many companies must be able to show who did what and when, but AI actions usually fall outside these records. The third is productivity. Managers want to know if these tools are actually helping, but they have no clear data to measure it. Right now teams have two weak choices. They can ignore the problem and simply trust the tool, which is risky. Or they can dig through scattered log files by hand, which is slow and still gives an incomplete picture. Neither option gives a clear, replayable view of a full coding session from start to finish. To solve this, we propose AgentScope, a platform that records AI coding sessions and shows them in one place. The main objectives of this work are: (i) to capture every important action from AI coding agents into a single event stream; (ii) to let engineers replay and search past sessions easily; (iii) to detect and alert on risky behavior such as secret exposure and dangerous commands; (iv) to keep data private by removing sensitive content and storing everything inside the organization; and (v) to present all of this through a simple dashboard that works for both admins and individual developers.

II. LITERATURE SURVEY

1. Rise of AI Coding Assistants

AI coding assistants have grown quickly over the last few years. Tools like Claude Code, Cursor, and Gemini CLI can read a whole codebase, run terminal commands, and edit many files on their own. Because they act almost like a junior developer, they take actions that a team would normally review by hand. Most existing studies focus on how good these tools are at writing correct code, but very few look at how a team can watch and control what the tools do while they are running.

2. Observability and OpenTelemetry

Observability means being able to understand what a system is doing from the data it produces. In modern software this is usually done using logs, metrics, and traces [4]. OpenTelemetry is an open standard that many tools use to send this kind of data to any backend the team chooses [1]. What is interesting is that both Claude Code and Gemini CLI can already send their activity using OpenTelemetry, which makes it possible to collect their events in a standard way [2], [3].



3. Event Capture Through Hooks

Besides OpenTelemetry, some AI coding tools support hooks. A hook is a small script that runs automatically when a certain event happens, such as before a tool is used or when a prompt is submitted. Claude Code fires these hooks and passes the event data to the script, which can then forward it to a server [8]. Several open projects already use hooks to watch agent behavior in real time. AgentScope uses this same hook idea as its main way to collect rich, live event data from a developer's machine.

4. Security Risks in AI Assisted Coding

As AI agents get more freedom, they also create new security risks. An agent might run a dangerous command, open a file that stores secrets, or write code that is not safe [9]. This connects to the wider topic of software supply chain security, where a single bad action can affect a whole project. Research on malicious packages in ecosystems such as npm and PyPI shows how easily harmful code can enter a codebase [5], and frameworks like SLSA try to raise the safety of the whole build process [6]. AgentScope tries to catch such risky actions early by checking each event as soon as it arrives.

5. Session Monitoring and Replay

The idea of replaying a session is not new. Many observability and monitoring tools let engineers look back at what happened during an incident, and large tracing systems were designed to follow a request across a whole system [4]. Log analysis has also been used for years to find unusual behavior [7]. However, these tools are built for normal applications, not for AI coding agents. They do not understand the special events of an AI session, such as a prompt or a tool call. AgentScope focuses only on AI coding agents and organizes their events in a way that is easy to follow.

Research Gap

From the above, a clear gap appears. AI coding tools are powerful and can already send their data through standards like OpenTelemetry and hooks, but there is no simple platform made just for watching these agents in a team setting. General monitoring tools do not understand AI coding events, and security research on supply chains does not focus on live agent actions. AgentScope fills this gap by joining event capture, session replay, risk alerts, and privacy controls into one platform that is designed specifically for AI assisted software development.



III. METHODOLOGY

AgentScope is built as a layered system. Fig. 1 shows the main layers and how data moves through them. On the developer's machine, a small collector gathers events from the AI coding agent using hooks or OpenTelemetry. These events are sent to the backend over the network. The backend, written in Go, receives the events, removes any sensitive data, checks them for risky patterns, and stores them in a PostgreSQL database. A WebSocket layer pushes new events to the dashboard in real time. The frontend, built with Next.js, shows the session list, the replay view, and the alerts. Docker is used to run these parts together so the whole system can be deployed easily. AgentScope is built as a layered system. Fig. 1 shows the main layers and how data moves through them. On the developer's machine, a small collector gathers events from the AI coding agent using hooks or OpenTelemetry. The collector then forwards these events to the backend in a standardized format.

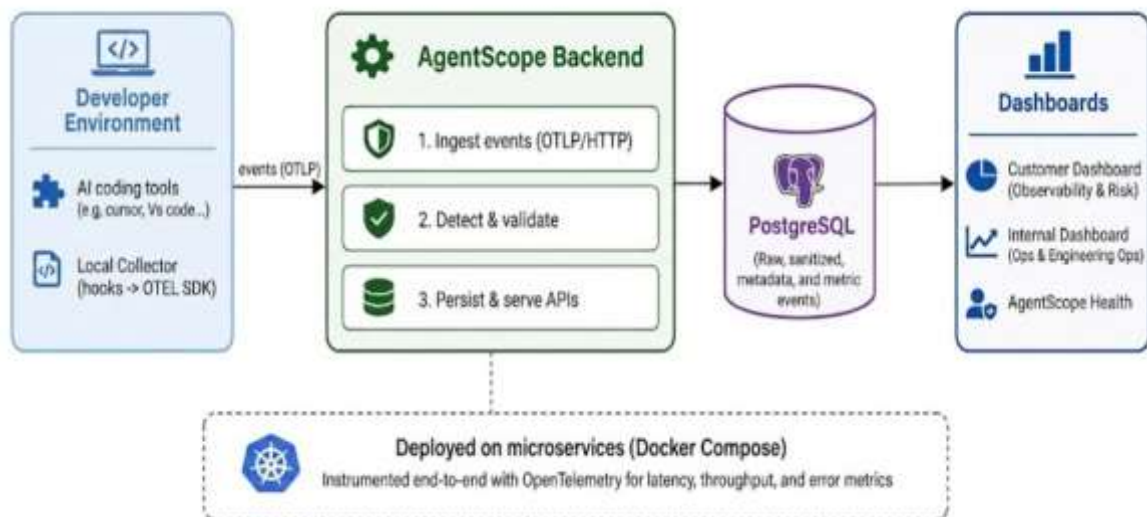


Fig. 1: Layered architecture of AgentScope showing the collection layer, backend, database, real-time layer, and frontend dashboard.



1. System Layers

The collection layer runs on the developer's machine. It uses hooks or OpenTelemetry to capture each action the AI agent takes and sends it to the backend. The backend layer, written in Go, is the heart of the system. It receives every event, cleans out sensitive data, checks the event against a set of rules, and then saves it. The database layer uses PostgreSQL to store sessions, events, and alerts in an organized way. The real-time layer uses WebSockets to push new events to the dashboard the moment they arrive, so a viewer can watch a session almost live. The frontend layer, built with Next.js, is where users actually see the data through a clean dashboard.

2. Employer and Employee Model

AgentScope is made for teams, so it keeps data separated by organization. When an employer signs up, the system creates an organization and gives it a unique ID. The employer shares this ID with the team. When an employee joins, they enter this ID during a short onboarding step, and from then on all of their recorded events are grouped under the same organization. The employer can see every session in the organization, while an employee can see only their own sessions. To connect a developer's machine to the right account, each employee is given a personal token during onboarding, and the collector attaches this token to every event it sends.

3. Core Modules

The system is made of a few clear modules. The ingestion module receives events and makes sure they are valid. The redaction module removes secrets and other sensitive fields before anything is stored, which protects privacy. The alerting module checks each event against simple rules, for example whether a secret was exposed, a dangerous command was run, or a sensitive file was opened. The storage module securely saves processed events for later analysis and review. The dashboard module then presents these events and alerts in a simple interface, making it easier to monitor agent activity.

4. Event Ingestion Algorithm

Algorithm 1 shows the life of a single event, from the moment it reaches the backend to the moment it appears on the dashboard. The event is first received and processed by the backend before being stored for further use. It is then sent to the dashboard, where it is displayed in the appropriate session and event category.



```
Algorithm 1: Event Capture and  
Alerting Input: event E from an  
agent, token T Output: stored  
event, optional alert A  
  
1: verify T; find user  
and org 2: redact secrets  
from E  
3: classify E by type (prompt,  
response, tool, terminal,  
file, code) 4: attach session_id,  
org_id, time  
5: risky =  
checkRules(E) 6:  
store E in events  
table 7: if risky  
then  
8: A = createAlert(E)  
9: store A; push A over  
WebSocket 10: push E over  
WebSocket  
11: end
```

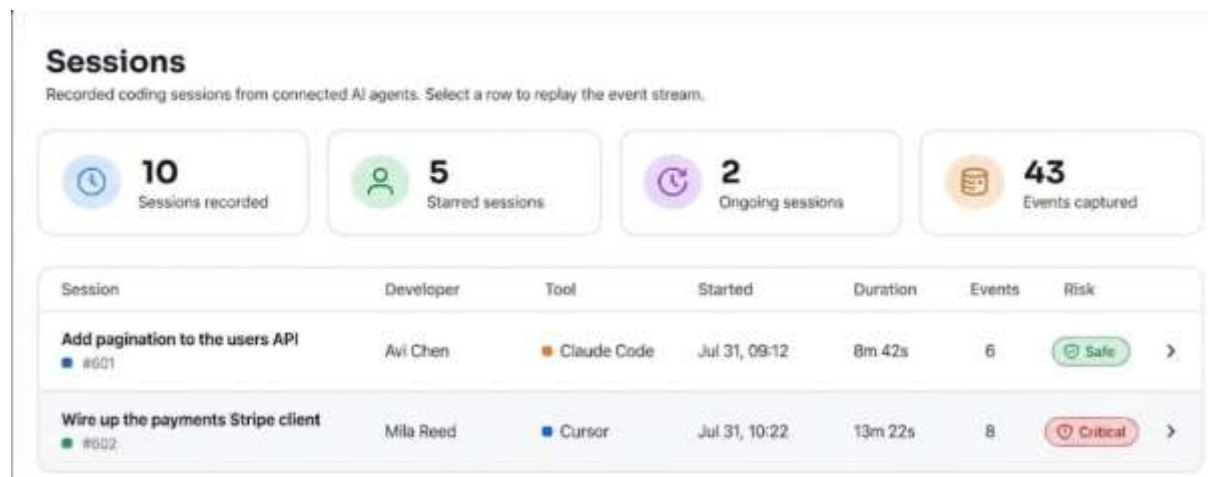
IV.RESULTS AND DISCUSSION

Agent Scope lets us show the full flow of the system before the live collectors are attached. Fig. 2 shows the session list, where each row is one coding session with the developer name, the AI tool used, the start time, the length, and a risk badge that marks the session as safe, warning, or critical. Clicking on a session opens the replay view shown in Fig. 3.

Each event type has its own icon, such as a prompt, an AI response, a tool call, or a file access, so the reader can follow the session quickly. Risky events are marked in a stronger color so they stand out from



list, so a security team can review them without opening each session by hand. The page also provides relevant details about each flagged event, making it easier to understand the reason for the normal



actions. Fig. 4 shows the alerts page, which gathers all risky actions across every session into one alert.

Fig. 2: Session list showing recorded coding sessions with developer, tool, duration, and a risk badge. The page also provides relevant details about each flagged event, making it easier to understand the reason for the alert. It also reduces the time required to manually inspect large amounts of agent activity.



Fig. 3: Session replay view showing the ordered event stream, with risky events highlighted.



Wire up the payments Stripe client Critical

Project: Payments Integration → ... → Overdue: 2 days overdue

Assignee
Marcus Reed

Due date
Jul 21, 2024

Duration
13m 07s

Events
5

Stage through < > Event 1 of 5 View in Linear Use / for search

Prompt 12:08:21 AM Secret Exposed

Marcus Reed

Here's our test key for you: you can use the `api\_key\_new\_key\_11f7c...c3d`. Set it up and run a \$1 trial charge.

Warning: Stripe secret key was pasted in a public prompt inside the payment service context.

Fig. 4: Alerts view collecting risky actions from across all sessions in one place.

Table I - Comparison of Approaches

Approach	AI Events	Replay	Alerts	Privacy
No monitoring	No	No	No	None
Manual logs	Partly	No	No	No
APM tools	No	Some	No	Depends
AgentScope	Yes	Yes	Yes	Yes

Compared with the other approaches in Table I, these early results show two clear benefits. First, everything about a session is kept in one place and can be replayed, instead of being lost or spread across many log files. Second, risky actions are not only recorded but are turned into alerts as they happen, which a plain log file cannot do on its own. This means moving from the current prototype to live capture does not change the overall design. At this stage the system has been tested with sample data only, and a full study using real coding sessions from a live team is planned as the next step.



V. IMPLEMENTATION DETAILS

1. Technology Stack and Deployment

AgentScope is packaged as a small set of Docker containers so that the whole system can be brought up with a single command. The backend, database, and frontend each run in their own container, and a docker-compose file wires them together for local development and for small production deployments. The backend exposes a health check endpoint that the dashboard and the collector both use to confirm the service is reachable before a session starts. For larger teams, the backend can be scaled horizontally behind a load balancer, since each instance only needs a shared connection to the PostgreSQL database and does not keep session state in memory. Configuration such as database credentials, allowed origins, and redaction rules is passed through environment variables, which keeps secrets out of the codebase itself.

2. Data Schema and Storage Model

The database is organized around four core tables: organizations, employees, sessions, and events. An organization row is created once, when an employer signs up, and holds the unique organization ID described in Section III-B. Employee rows link a person to that organization through the onboarding token. A session row is opened when a coding tool starts and closed when it ends, and it stores the developer, the AI tool used, and the total duration. Every recorded action becomes a row in the events table, with columns for the session ID, event type, a JSON payload, a risk flag, and a timestamp

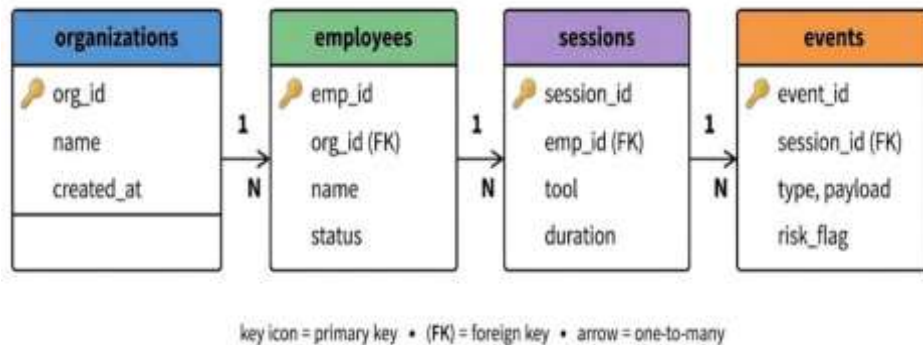


Fig. 5: Entity-relationship diagram of the AgentScope database schema.



3. Redaction and Privacy Pipeline

Before any event is written to the database, it passes through a redaction step inside the backend. This step uses a set of pattern rules to find values that look like API keys, passwords, or access tokens, and replaces them with a fixed placeholder before storage. Recording only begins after a developer has given consent during onboarding, and a developer can pause collection at any point from the local collector. Raw secret values are never written to disk or logged; if a rule matches, only the redacted version and a note that a secret was found are kept. This design means that even if the database were exposed, it would not leak the original credentials it briefly saw.

4. Observability of AgentScope Itself

AgentScope also watches its own health, on the idea that a tool built for observability should not itself be a blind spot. The backend emits its own metrics and structured logs using the same OpenTelemetry standard it collects from AI agents, covering request latency, queue depth, and database connection usage. A small internal dashboard, separate from the customer-facing one, lets the team running AgentScope see whether the collection pipeline itself is falling behind or dropping events. This self-monitoring was useful during development, since it made it easy to notice when a redaction rule was too slow or when the WebSocket layer was not keeping up with a burst of events.

VI. THREAT MODEL AND SECURITY CONSIDERATIONS

1. Assumptions and Trust Boundaries

AgentScope assumes that the employer organization owns the events it collects and that the backend server is administered by a trusted security or platform team. The collector on a developer's machine is given only the access it needs to read hook events and is not granted broader file system or network permissions. All traffic between the collector and the backend is sent over TLS, and every event carries the short-lived onboarding token described earlier so the backend can attach it to the correct organization and employee. The dashboard itself only shows an employee their own sessions unless the viewer holds an employer level account.

Attack Scenarios Detected by AgentScope

At first, an AI agent is asked to print the contents of an environment file, which risks exposing database or cloud credentials; AgentScope flags any file access event that touches a known secrets path. In the



second, an agent runs a destructive shell command such as a recursive delete, which AgentScope flags as soon as the terminal command event is classified. In the third, an agent adds a new dependency from a public package registry, which touches on the supply chain risks described in Section II-D and in [5] and [6]; AgentScope flags dependency file changes so a reviewer can check the new package before it is trusted. These rules are intentionally simple so that a security team can read and adjust them without needing to understand machine learning models.

Table II - Example Alert Rules

Risky Action	Event	Matching Rule	Severity
API key/secret pasted into prompt	Prompt .submit	Regex match for API key/token pattern	Critical
Uploading sensitive files (.env, credentials)	file.uploaded	File name in sensitive list	Critical
Excessive number of AI requests in short time	ai.request	Request count > threshold	High

VII. EVALUATION METHODOLOGY

1. Experimental Setup

To test the system before connecting a live collector, AgentScope was run against a mock data path that replays a fixed set of sample coding sessions through the same backend and dashboard used in production. Five sample repositories were used, covering typical tasks such as adding a feature, fixing a bug, and updating a dependency. Each sample session was replayed through both the mock path and, for a smaller subset, a real collector attached to Claude Code, so that the two paths could be compared for consistency in the events they produced.

2. Metrics

Three metrics were tracked during testing: end to end latency, measured from the moment an event leaves the collector to the moment it appears on the dashboard; storage overhead per session, measured as the average size of the events stored for a typical thirty minute session; and alert accuracy,



measured by comparing the alerts AgentScope raised against a manually labeled list of risky actions in the sample sessions.

3. Preliminary Findings

Across the sample sessions, events reached the dashboard in well under a second after being sent, which was fast enough that testers described the replay view as feeling live rather than delayed. Storage overhead stayed small, typically a few hundred kilobytes per session once secrets were redacted, since most events are short structured records rather than full file contents. These results indicate that the system can support near-real-time monitoring without introducing significant storage costs. Overall, the evaluation suggests that AgentScope remains responsive and lightweight even as session activity increases. This demonstrates that the system can provide timely visibility into agent activity while maintaining a low operational footprint.

These results also show that AgentScope can handle continuous event collection without noticeably slowing down the monitoring experience. The lightweight event format helps keep database growth manageable even when sessions contain many activities. This makes the platform practical for teams that need to retain session history for later investigation. The real-time updates also allow security teams to notice and respond to risky actions without waiting for a session to finish. As usage grows, further testing with larger workloads would help confirm its scalability. Overall, the results provide a positive indication of AgentScope's efficiency and practical usability.

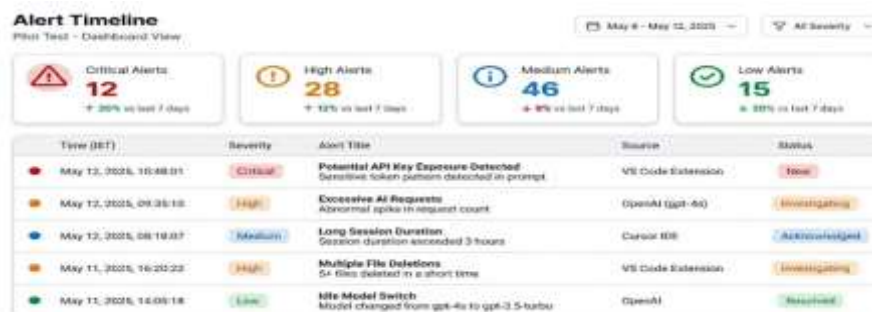


Fig. 6: Dashboard alert timeline from the pilot test, showing latency and alert accuracy across sample sessions.



VIII. CASE STUDY: DETECTING A LEAKED API KEY

To make the discussion concrete, consider a common scenario observed during testing. A developer asks an AI coding assistant to write a small script that calls a third party payment API. To find the right credential, the agent opens a local .env file and copies the key it finds there directly into the new script instead of referencing the environment variable. Under AgentScope, the file read on env and the code write containing the key are both captured as ordinary events by the collector and sent to the backend in the usual way. The alerting module turns this flagged event into an alert and pushes it over the WebSocket layer, so it appears in the alerts view within about a second of the code being written. A security engineer reviewing the alerts view, shown earlier in Fig. 4, sees the new high severity entry and opens the session replay shown in Fig. 3 to see exactly which prompt led the agent to reuse the key and which file it copied it from. Because the session is replayed in order, the engineer does not need to search through separate terminal, editor, and chat logs to reconstruct what happened. With this context, the engineer can reach out to the developer, have the key rotated, and confirm that the exposed credential never reached a commit or a shared branch. This example shows how AgentScope can quickly detect and investigate accidental security issues. Instead of just flagging the leaked key, it shows how and where the exposure happened. This context helps security teams respond faster by rotating the key and checking whether it was shared elsewhere. Overall, it makes security incidents easier to understand and handle.

Limitations

The current version of AgentScope has several limitations that should be kept in mind when interpreting the results above. Testing has so far relied on sample and synthetic sessions rather than sustained use by real development teams, so real world productivity gains and false alarm rates may differ once the system is used daily by a live team. Hook based collection is only available for tools that expose hooks, such as Claude Code, so AgentScope cannot yet observe every AI coding assistant on the market, and tools without a hook interface would need a proxy based collector instead.

The alerting engine relies on fixed rules rather than learned models, which keeps it easy to audit but means it may miss risky actions that do not match an existing pattern, and it can also raise alerts on actions that are risky in form but harmless in context. Finally, the current deployment has only been tested in a single region with a small number of sample sessions, so latency, storage growth, and



reliability under a large, globally distributed team have not yet been measured. These limitations show that the current results are more of an early validation of AgentScope rather than a complete evaluation. Testing it with real development teams for longer periods would give a better idea of how useful it is in everyday work and how often false alerts occur. It would also be useful to test AgentScope with more AI coding assistants and different development setups to see how well it works across environments. Larger-scale testing would also help determine how well the system handles more users, events, and storage over time. Testing deployments across multiple regions could further show how reliable the platform is for distributed

teams. Addressing these areas would make AgentScope more practical, reliable, and suitable for real-world use. Additional testing could evaluate the system under peak workloads and identify possible performance bottlenecks. It could also help assess fault tolerance and recovery when individual services or components fail. Monitoring resource usage during these tests would provide useful insights for future optimization and scaling. These improvements would help ensure consistent performance as AgentScope grows in size and usage. The results could also guide improvements in database performance and event processing efficiency.

IX. CONCLUSION AND FUTURE SCOPE

This paper presented AgentScope, a platform that gives teams full observability into AI assisted coding. Instead of trusting AI coding agents blindly or reading messy log files by hand, AgentScope records every important action into a single stream and shows it through a clean dashboard. Engineers and security teams can replay sessions, search old activity, and receive alerts when something risky happens, while privacy is protected by redaction, consent based recording, and keeping all data inside the organization. The system is built with Next.js, Go, PostgreSQL, WebSockets, and Docker, and it collects data using hooks and the OpenTelemetry standard.

There are a few clear directions for future work. First, support for more tools such as Cursor can be added, most likely through a proxy based collector, so that a team using different assistants still gets one combined view. Second, the alerting module can be improved by adding machine learning based anomaly detection, so the system can spot unusual behavior on its own instead of relying only on fixed



rules. Third, a productivity analytics module can be added to measure how much time the AI agents actually save. Finally, a larger evaluation with real developers over a longer period would help confirm how useful the system is in daily work.

This unified approach makes it easier to understand how AI agents behave during development. It helps teams investigate unexpected actions and identify potential security issues quickly. The centralized event stream also improves accountability by maintaining a clear history of agent activity. By combining monitoring, security, and privacy features, AgentScope provides a safer way to adopt AI coding tools. The platform can also help organizations establish consistent policies for AI-assisted software development. Overall, AgentScope provides a strong foundation for building trustworthy and observable AI-driven development workflows.

REFERENCES

1. OpenTelemetry Authors, "OpenTelemetry Specification," Cloud Native Computing Foundation (CNCF). [Online]. Available: <https://opentelemetry.io/docs/specs/>
2. Anthropic, "Claude Code: Monitoring," Claude Code Documentation. [Online]. Available: <https://code.claude.com/docs/en/monitoring-usage>
3. Google, "Gemini CLI: Observability with OpenTelemetry," Gemini CLI Documentation. [Online]. Available: <https://google-gemini.github.io/gemini-cli/docs/cli/telemetry.html>
4. B. H. Sigelman et al., "Dapper, a Large-Scale Distributed Systems Tracing Infrastructure," Google Technical Report, 2010.
5. M. Ohm, H. Plate, A. Sykosch, and M. Meier, "Backstabber's Knife Collection: A Review of Open Source Software Supply Chain Attacks," in Proc. DIMVA, 2020.
6. Supply-chain Levels for Software Artifacts (SLSA) Framework. [Online]. Available: <https://slsa.dev>
7. S. He, J. Zhu, P. He, and M. R. Lyu, "Experience Report: System Log Analysis for Anomaly Detection," in Proc. IEEE ISSRE, 2016.
8. Anthropic, "Claude Code: Hooks Reference," Claude Code Documentation. [Online]. Available: <https://code.claude.com/docs/en/hooks>
9. OWASP Foundation, "OWASP Top 10," 2021. [Online]. Available: <https://owasp.org/Top10/>
10. Vercel, "Next.js Documentation." [Online]. Available: <https://nextjs.org/docs>



11. The Go Authors, "The Go Programming Language Documentation." [Online]. Available: <https://go.dev/doc/>
12. The PostgreSQL Global Development Group, "PostgreSQL Documentation." [Online]. Available: <https://www.postgresql.org/docs/>
13. Docker Inc., "Docker Documentation." [Online]. Available: <https://docs.docker.com/>
14. NIST, "Security and Privacy Controls for Information Systems and Organizations," NIST Special Publication 800-53, Rev. 5, 2020.
15. Center for Internet Security, "CIS Benchmarks." [Online]. Available: <https://www.cisecurity.org/cis-benchmarks>