AgentPProf: Semantic Profiler for Long Horizon AI Agents

Yusheng Zheng[1,2], **Chaokun Chang**[3], **Yu Mao**[4], **Tianyuan Wu**[3], **Yuxi Huang**[2], **Tao Ma**[5], **Wenan Mao**[5], **Shuyi Cheng**[5], **Andi Quinn**[1], **Wei Wang**[3]

[1]UC Santa Cruz [2]Eunomia Labs [3]HKUST [4]Independent Researcher [5]Alibaba Group

Abstract

AI agents increasingly orchestrate long-running activities with users, tools, and system resources for days and weeks. To improve quality, safety, and cost efficiency of AI agents, developers need to determine where failures happen, what triggers unsafe effects, and what tasks consume the most budget. They can then optimize those tasks to reduce token and time cost. In systems software, profiling answers similar questions by aggregating resource consumption and attributing it to responsible code paths to identify hotspots. Yet existing agent observability tools focus on per-execution debugging and tracing rather than cross-run, long term profiling, making these questions difficult to answer at scale. Agent observability needs profiling, not only debugging. Profiling agents is challenging: the responsible entities are task intent like *diagnose authentication*, *compare branches* rather than code paths, and lack stable identifiers for aggregation. We propose a *semantic operation stack model* that adapts profiling to agent trajectories. Uniform *operations* represent all activities, and *operation stacks* replace the runtime call stack, enabling hierarchical attribution at different granularities. We observe that an agent's task occupies a contiguous span and decomposes into subtasks, so we introduce *recursive operation segmentation*, which recursively splits trajectories at task boundaries. AgentPProf is a profiler that aggregates agent trajectories into pprof-compatible profiles, enabling flame graph visualization and analysis. On CodeTraceBench, AgentPProf reaches 0.764 B^3 F1 against human annotations. On three problem-localization benchmarks, the profile raises MAP by up to 56%. AgentPProf effectively attributes resources, locates problems, and helps optimize token cost at practical cost. AgentPProf is available at https://github.com/eunomia-bpf/agentsight.

Introduction

AI agents (Yang et al. 2024; Anthropic 2025; Xie et al. 2024) orchestrate multi-step activities that span two layers, high-level intent (prompts, LLM calls, tool invocations) and low-level system effects (process spawns, file and network I/O). As agents evolve from short, scripted workflows into complex systems that run for hours and days, with a large number of interactions each, production teams accumulate many such trajectories over weeks or months.

To improve agent quality, safety, and cost efficiency, developers need to analyze operational behavior across trajectories and interactions. Which categories of tasks consume the most token budget? Where do failures concentrate across workflows? Which behavioral patterns trigger unsafe system effects? Answering these questions is the first step. Acting on the answers, by optimizing high-cost tasks and fixing failure-prone workflows, is the goal. Answering these questions requires analysis and evaluation across many trajectories (Mazaheri and Mazaheri 2026; Lù et al. 2025), a task becoming costly for manual inspection or per-trajectory LLM evaluation (Zheng et al. 2023) at scale. In traditional systems software, profiling answers these questions by aggre-

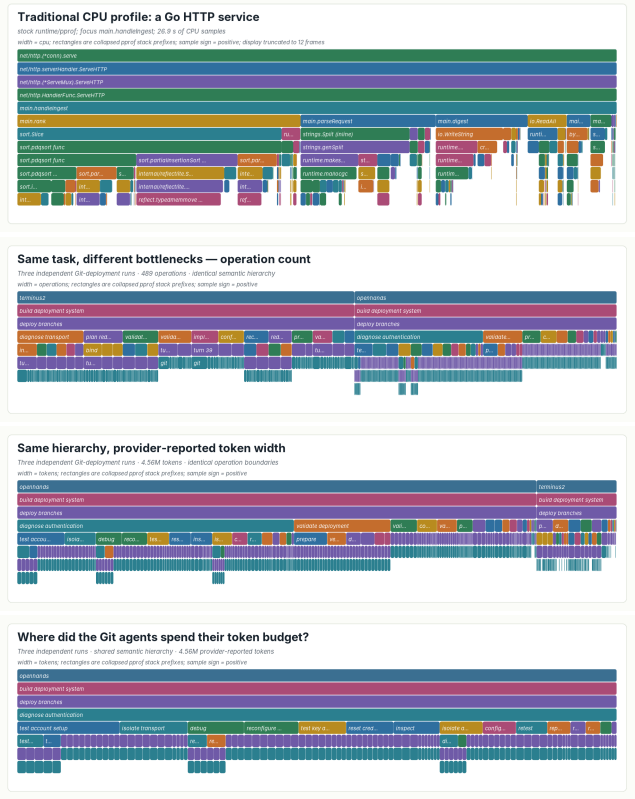

Figure 1: One renderer, four standard pprof profiles. *(a)* CPU time in a Go HTTP service. *(b, c)* Three agent executions under one fixed hierarchy: width is operation count (b) or tokens (c). *(d)* Token profile focused on `diagnose authentication`.

gating resource consumption and attributing it to responsible entities, distinct from per-execution debugging and tracing.

Yet existing agent tools support debugging and tracing but not profiling. Some (LangChain 2024; Langfuse 2024; Arize AI 2024a; OpenTelemetry 2024) organize events into per-execution span trees, effective for diagnosing a single run but unable to aggregate by task intent or workflow phase. Others (Datadog 2025; Laminar 2025) cluster inputs or extract structured signals, but characterize *input distributions* ("30% of users ask code questions") rather than *resource attribution* ("review tasks consume 40% of the token budget"), because tags at the request level do not propagate to downstream tool calls and system effects.

Adapting profiling to agent trajectories is challenging. In traditional software, the responsible entities are code paths, and profiling is straightforward because function names are stable and runtime call nesting provides the attribution hierarchy. Agent behavior differs. To answer the questions above, the responsible entities must be task intent and workflow phase, not code paths. Agent trajectories lack the two properties that make traditional profiling possible. Prompts are natural language, so two prompts expressing the same intent share no common string. Agent events have no runtime call-stack hierarchy because prompts, tool calls, and system effects are not nested by execution the way function calls produce stack frames. A sequence of prompts can be a task or multiple tasks, and a task can be part of a bigger task.

Despite these differences, the fundamental profiling methodology transfers to agent trajectories by projecting resources onto responsible entities, assigning stable identifiers, and attributing hierarchically. We propose a *semantic operation stack model* with two components. *Operations* are uniform records with string fields and additive measures that represent all agent activities (prompts, LLM calls, tool invocations, and system effects) without type-specific objects. *Operation stacks* provide hierarchical attribution, replacing the runtime call stack. Choosing different fields changes the attribution granularity: the same operations can be attributed by task category, by execution phase, by session, or by action type, and one fixed hierarchy replays under any additive measure (Figure 1).

We implement this model in AgentPProf, an offline profiler that compiles local agent trajectories into pprof-compatible profiles (Figure 2). One algorithm, *recursive operation segmentation*, supplies the missing identifiers. Labeling every prompt individually would be prohibitively expensive, but task structure is recoverable at lower cost: a task occupies a contiguous span of a trajectory and decomposes into contiguous subtasks, so recovering it requires only finding the transitions between them. The algorithm finds where responsibility changes, names the resulting intervals, and yields short names starting with a verb (like `diagnose authentication`) that fold across sessions. The interface is implementation-neutral.

We evaluate whether AgentPProf effectively attributes resources, locates problems, and recovers human-recognizable task structure at practical cost. Across all 405 CodeTraceBench (Li et al. 2026) trajectories, recursive segmentation agrees with human stage annotations at 0.764 B^3 F1 (Bagga and Baldwin 1998), versus 0.663 for a statistical baseline and 0.541 for raw actions. On three complete localization benchmarks, the profile improves ranking over each benchmark's own diagnostic, raising MAP by 0.031, 0.107, and 0.117; as a navigation aid, it reduces content to inspect by 12 percentage points at equal ranking quality. A profile-derived repair reduces agent tokens by 19% without degrading task quality, and annotation cost is reduced by 20%.

This paper makes three contributions:

1. **Semantic operation stack model.** We identify the missing profiling layer in agent observability and propose a model of uniform operations and query-time operation stacks (Background and Design).
2. **System: AgentPProf.** A profiler that implements this model, producing pprof-compatible output.
3. **Evaluation.** We evaluate profiler output against independent ground-truth annotations that stay hidden from the profiler, on eight public benchmarks and three real trajectory datasets.

Background and Motivation

LLMs and AI Agents. A typical agent trajectory interleaves two layers of activity. At the intent layer, the agent receives a user prompt, issues LLM calls, and invokes tools (code execution, web search, file editing). Each tool invocation triggers system effects at the lower layer: process spawns, file reads and writes, and network requests. A single trajectory may contain hundreds of cycles between intent and system effects, and a production team accumulates many trajectories over time.

System Profiling. In traditional software, the profiling pipeline works as follows: a profiler periodically samples execution state, attaches each sample to the current call stack, and merges samples with identical stacks. Flame graphs (Gregg 2011) and pprof (Google 2024) call graphs visualize the result, where width or node size represents aggregate cost. Engineers use these profiles to find where CPU or memory budget is spent, then optimize the responsible code paths to reduce cost. These tools support flexible aggregation, but they all assume stable function names and a runtime call stack. Pprof's `tagroot`/`tagleaf` options can add label-derived pseudo-frames, but only on top of an existing execution stack that agent trajectories do not have.

A Motivating Case. Three independent agent attempts at one real Git-deployment task all failed to deliver the requested password-authenticated endpoint. The developer's question is *what went wrong, and is it the same problem across all three runs?* Reading three transcripts totaling 489 operations and 4.6M tokens would take hours. Figure 1 places a conventional CPU profile beside the semantic profile of those three runs. Both are standard pprof profiles drawn by one renderer and read by identical rules, where width is aggregate cost, a parent contains its children, and identical stacks recorded at different moments fold into one frame. What differs is where the frames come from. A call stack supplies `net/http.(*conn).serve` at every sample for

free, but nothing in an agent trajectory supplies `diagnose authentication` because the three runs express that intent through different prompts and different shell commands. Once that name is constructed, the profile answers the engineer's question directly.

The profile answers in one view (Figure 1): all three runs share a `diagnose authentication` responsibility that consumes 46% of their combined token budget but only 21% of operation count. Expanding the view shows all three executions verifying substitute transport paths without ever establishing the target endpoint. The insight is that the agents kept retrying SSH variants instead of fixing the actual credential issue, and operation count alone would hide this because authentication commands are cheap to issue but expensive to verify. The profile points to where to intervene. Adding early credential validation or limiting retry depth in `diagnose authentication` could cut token cost for similar tasks.

Challenges for Agent Profiling. Adapting profiling to agent trajectories faces two core challenges. The first is that the responsible entities differ. In traditional software, profiling attributes cost to code paths, but in agent trajectories the relevant entities are task intent and workflow phase. Existing tools (OpenTelemetry 2024; Arize AI 2024b) do not capture these entities because prompts are natural language: two prompts expressing the same intent may differ, so the profiler cannot aggregate them the way it aggregates identical function names. The second challenge is that agent trajectories have no runtime hierarchy for attribution. In CPU profiling, the call stack determines attribution at every level: `malloc`'s cost belongs to `parse`, to `process`, and to `main` simultaneously. In AI Agents, sequence of prompts may constitute one task or several, and a task may be part of a larger workflow, but no runtime mechanism marks these boundaries or determines at which granularity to attribute cost.

Design

Agent profiling needs the three mechanisms that call stacks provide automatically, but must achieve them without a call stack: **R1** (cross-layer projection): connect system-layer consumption to intent-level categories; **R2** (stable identifiers): derive short, repeatable identifiers from natural-language prompts before folding; **R3** (hierarchical attribution): construct attribution hierarchies from data, not execution nesting. Operations satisfy R1 by representing intent and system-effect activities in a single record with tag inheritance, so intent tags propagate to the system effects they trigger via AgentSight (Zheng et al. 2025). *Recursive operation segmentation* satisfies R2 and R3 together by deriving short, stable interval names from trajectory content and nesting those intervals into the attribution hierarchy that replaces the runtime call stack. *Operation stacks* then attribute resources at every level of that hierarchy at query time, so the same data supports multiple views without rebuilding the input. The profiling pipeline has four stages: (1) parse raw trajectories into operations, (2) segment trajectories into named nested intervals (or apply mapping rules), (3) project each operation onto a stack frame sequence chosen at query time, and (4) fold operations with identical stacks, summing their weights.

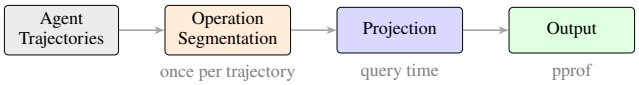

Figure 2: Data-flow pipeline. Local histories and public datasets both produce uniform operations; segmentation runs once, projection and folding at query time.

Semantic Operation Stack Model

This section defines operations and operation stacks, the data structures that satisfy R1 (cross-layer projection) and enable R3 (hierarchical attribution). R2 (stable identifiers) requires deriving identifiers from trajectory content and is addressed in Section .

Because agent activities span both intent and system-effect layers, a profiler should not distinguish between them in the data representation. AgentPProf therefore represents every activity as a single abstraction, the `operation`, a weighted record with string fields and additive measures. Each operation carries string fields (`project`, `agent`, `session`, `prompt_tag`, `kind`, `model`, `path`, `domain`, `status`, ...) and additive measures (token count, duration, event count). A prompt, an LLM call, a tool invocation, a file read, a GUI click, and a process event are all operations with the same schema, distinguished only by which fields carry values.

An `operation stack` provides hierarchical attribution for agent trajectories, replacing the runtime call stack. An ordered list of fields $[f_1, f_2, \ldots, f_k]$ projects each operation o to a frame sequence $\langle o.f_1, o.f_2, \ldots, o.f_k \rangle$, and operations with identical sequences are merged, summing their weights. Unlike a call stack, the fields are chosen at query time, not determined by execution. Changing the fields changes the attribution without changing the data, so the same operations can be attributed by task, by session, or by action type. Sessions and spans are optional fields, so the same data supports both debugging (include `session`) and aggregate profiling (exclude it). Built-in views cover common questions (e.g., `tokens`, `time`, `files`).

Recursive Operation Segmentation

Operation stacks can project any field an operation already carries, but agent trajectories lack the two fields that make traditional profiling work: stable task identifiers and a nesting hierarchy. Labeling every prompt individually would be prohibitively expensive. Recursive operation segmentation derives both fields from trajectory content at lower cost by marking only the transition points where responsibility changes.

The algorithm rests on one structural intuition: an agent's task occupies a contiguous span of the trajectory, and it decomposes into smaller contiguous subtasks. We model task structure as a set of nested named intervals, and recovering it only requires finding the transition points between them.

The algorithm reads a trajectory, finds the transition points where the agent's responsibility changes, and names the in-

tervals on both sides. It then recurses into each interval, cutting again wherever a finer responsibility change remains, and stops when an interval reads as one coherent piece of work. Formally, a trajectory is an ordered step sequence $T = (t_1, \ldots, t_n)$, and a segmentation is a set S of named intervals over T that is nested (any two intervals are disjoint or one contains the other), covers every step, and contains the session-wide interval as its root. The algorithm computes S by one recursive rule: SEGMENT(I) selects zero or more transition points inside interval I, which partition I into consecutive child intervals. Each child receives a short name starting with a verb and is segmented recursively. Selecting no transition terminates the branch. A step's operation path is the name chain of the intervals containing it, from the session root to its innermost interval. Equal paths fold across sessions. Concretely, in one Git-deployment execution the algorithm cuts the session into `build deployment system` and, inside it, cuts again where the agent stops writing configuration and starts debugging access, yielding `deploy branches` and `diagnose authentication`. A step inside the latter carries the path `build deployment system > diagnose authentication`, and because the same names reappear in the other two executions, their costs fold together in Figure 1. Any implementation able to find transitions and name intervals (an agent, a language model, or a statistical rule) can serve as the segmenter. In evaluation, we use Codex (OpenAI 2025) with GPT-5.6-sol-high, with one fixed instruction per trajectory:

> *Input:* one trajectory (each step's prompt, command, and output summary, with no labels or scores).
> *Action:* read it, emit a mark wherever the responsibility changes, and use AGENTPPROF to check and update the segmentation, and re-read and update the marks iteratively until you think the segmentation is complete.
> *Output:* sparse path marks, e.g.,
> `step 1: [deploy git server]`
> `step 4: [deploy git server, diagnose SSH access]`
> `step 15: [deploy git server, verify web endpoint]`
> —one line only where the path changes; names start with a verb, one to three words.

The sparse marks still produce a complete segmentation because steps between two marks inherit the preceding path. Segmentation is iterative, so the agent can revise marks incrementally without re-annotating the entire trajectory. The model supports both offline evaluation and online annotation during execution.

Implementation

This section describes how AGENTPPROF realizes the design from Section . AGENTPPROF is implemented as a Rust CLI (∼9.8K LOC, Figure 2). It reads Codex and Claude Code local JSONL history files and AgentSight (Zheng et al. 2025) recordings for system effects. These files are written incrementally during agent execution, so AGENTPPROF can build profiles while the agent is still running. The pipeline has two stages: (1) preprocessing extracts a readable trajectory summary, and (2) the segmentation agent reads this summary and writes interval marks to an annotation file, which it can revise until satisfied. The CLI validates that marks are nested and cover every step, then emits a standard pprof protobuf profile that `go tool pprof` reads directly. Projection is deterministic: each operation contributes its semantic path and LLM/tool evidence, with session identities kept as pprof labels rather than visible frames, so equal semantic prefixes fold across sessions. Switching the additive measure replays the same annotations and changes only stack widths.

Fields already recorded literally (e.g., skill invocations) become stack levels without annotation cost. Multi-agent orchestration works automatically because subagent operations inherit the outer agent's semantic path. A non-LLM statistical segmenter is also available, scoring transitions by NPMI (Bouma 2009) and calibrating an unsupervised cutoff with k-means (MacQueen 1967).

Evaluation

We evaluate whether AGENTPPROF effectively attributes resources, locates problems, and recovers human-recognizable task structure at practical cost. The evaluation uses three data classes. *Real inputs*: 41 long-horizon coding-agent sessions (three of which repeat one Git-deployment task and form the RQ1 case), 440 mixed web-agent trajectories (Lù et al. 2025), and one developer's complete local session history from the authors' workstation (1,394 sessions), whose 42 long-horizon development sessions are the annotated portion. *Annotated trajectories*: 405 CodeTraceBench trajectories with 20,866 operations and 2,948 human-annotated stages (Li et al. 2026), 287 OSWorld-Human sessions (WukLab 2025), 1,012 AgentBoard goals (Ma et al. 2024), and 2,737 published action labels (Bouzenia and Pradel 2025). *Problem-localization benchmarks*: the complete AgentProcessBench, HINTBench, and TraceElephant workloads, with independently annotated faulty operations, over 27,346 operations (Fan et al. 2026; Wang et al. 2026b; Chen et al. 2026a). All annotations, outcomes, and scores stay hidden from every method until its output is produced. Benchmark trajectories average 8–52 operations (short to medium). The 42-session portion from the research of this paper supplies long-horizon coverage, whose longest sessions span tens of hours, and the RQ4 union covers 27,765 operations.

RQ1: Does Semantic Profiling Improve Resource Attribution?

Setup. If semantic profiling improves resource attribution, then the same fixed hierarchy should (1) reunite cross-run responsibility that alternative organizations scatter, and (2) reveal materially different bottlenecks when the additive measure changes. We test both on 41 real long-horizon agent trajectories (3,146 user turns, 5,750 operations), including three independent executions of one Git-deployment task whose 735 steps receive 96 recursive annotations reaching semantic depth five.

The motivating case revisited. The Git-deployment case from Section demonstrates that semantic organization reunites cross-run responsibility that alternative organizations scatter. Here we add a third measure (elapsed time) to show

measure-sensitivity. The same hierarchy replays under all three measures with exact conservation, and the `diagnose authentication` subtree's share rises from 21% (count) through 37% (time) to 46% (tokens). Changing only the measure moves the attributed share by more than a factor of two. A quantitative control shows why both alternatives fail. *Raw-action grouping* (by literal action-type, e.g., `run`, `edit`) fails: of 105 authentication operations, 102 carry only the label `run`, and that same bucket mixes in 97 unrelated operations. A coarser action-kind grouping scatters the work across six kinds (execute 39%, version-control 22%, edit 19%, inspect 12%, search 7%, install 1%). *Native call trees* (each operation under its source LLM or tool call) place each operation under a distinct source call with no shared responsibility node. Only the semantic hierarchy reunites authentication into one focusable cross-run path while preserving all call/tool evidence as leaf nodes. Separately, real AgentSight (Zheng et al. 2025) eBPF recordings replay the same way, and all 1,520 captured process spawns and file operations fold under task responsibilities with exact conservation, demonstrating the system-effect layer end to end.

Use case 2: where did this project's agent budget go? A developer spent weeks building AgentPProf and this paper with coding agents, accumulating a local session history whose long-horizon portion (42 sessions, 18 Codex and 24 Claude Code, with 1,252 prompts, 5,620 LLM calls, and 1.38 billion tokens) is annotated here. The question is *what did the agents actually spend tokens doing?* The profile shows the answer is not one dominant sink but a broad distribution, and the largest path (`refine paper > align evaluation`) holds only 1.7% of tokens. The three longest sessions (spanning tens of hours each) keep distinct dominant responsibilities (evaluation alignment, evidence inspection, and merge resolution). Switching to operation count shifts where mass concentrates. Token mass stays at prompt depth (70%), while operation mass resolves deeper (44% at depths three and four). The history also demonstrates multi-agent composition at scale, with 98 subagent delegations across 14 sessions where each delegation contains all downstream subagent operations and composes with annotated semantic levels.

The same hierarchy replays under FILE-READ, FILE-WRITE, and NETWORK-target widths, making side effects auditable by responsibility and revealing which tasks wrote most artifact files or made most network calls.

Skill invocations appear literally in the session log and cost no annotation, so they cover the developer's complete history (1,394 sessions, 6.9B tokens). Under that level, `paper-writing-style` leads both measures (29M tokens, 99.47% cache reads), highlighting its repeated-context workflow as the first thing to inspect. The largest single session holds only 31% of that mass, so the priority appears only after folding. The 99.47% cache-read ratio suggests the workflow repeatedly rebuilds similar context. Restructuring it to reuse context across invocations could reduce token cost.

Across 440 AgentRewardBench trajectories (7,229 operations, 51,904,621 tokens, both conserved exactly), ranking operations by count versus tokens yields high agreement

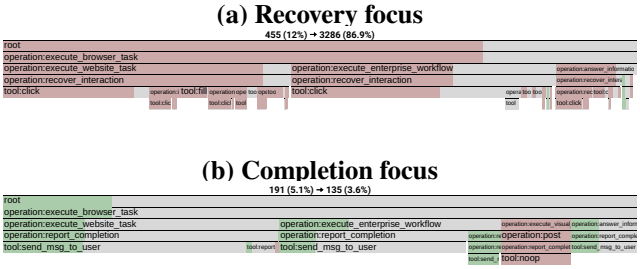

Figure 3: Stock-pprof differential flame graphs (bad minus good). Box width sums both contributions; inset shows net difference (rose: bad excess; green: good excess). Recovery dominates bad runs; completion favors good runs.

(mean tau-b 0.886). The 13% of tasks where agreement falls below 0.7 are exactly where multi-measure profiling adds value (details in Appendix).

Takeaway. Both conditions hold: the semantic hierarchy reunites cross-run responsibility that raw-action and native organizations scatter (the authentication case), and switching the measure on that fixed hierarchy moves attributed share by more than 2× (21% to 46%), revealing bottlenecks that operation count alone would hide. These bottlenecks are optimization targets: tasks consuming disproportionate budget can be restructured to reduce cost.

RQ2: Does Profiler Output Correspond to Real Problems?

Setup. To evaluate whether profiler output corresponds to independently annotated real problems, we apply three tests: (1) *differential analysis* checks whether the profile's failure structure matches expert behavioral labels across 440 runs; (2) *localization* checks whether adding profile scores to existing benchmark judges improves fault ranking; (3) *reading study* checks whether semantic names help an LLM agent reach the same ranking while inspecting less content.

Use case 3: what do failing runs do differently? A team has 440 web-agent runs (Lù et al. 2025) over 125 tasks, where every task has both successful and failed attempts (202 successful, 238 failed). The question is *what behavior separates success from failure?* Reading 440 transcripts (7,229 operations) is infeasible. We pair failed and successful runs within each task (338 pair occurrences), build a profile for each group, and subtract (failed minus successful) to get a signed difference profile (Figure 3). The answer is immediate: failed runs spend 44.6% of their steps in `recover interaction` (retrying, re-searching, re-navigating) versus only 12.0% for successful runs. The hierarchy decomposes the recovery responsibility into verification challenges, repeated searches, mistaken navigation, and record retries, with source labels identifying which sessions contributed each width. The insight is that failing agents get stuck in retry loops rather than backing out and trying a different approach. This structure matches independent expert looping labels on 435 trajectories at AP .634 versus random baseline .398 (dif-

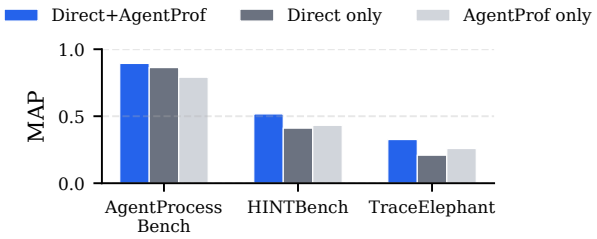

Figure 4: MAP over the complete RQ2 datasets (614, 400, and 220 queries included in MAP). Higher is better.

ference interval [.181, .293]), confirming the profile surfaces a real behavioral pattern. A fixed-chain repeated/error control reaches the same detection quality (AP .656), but only the recursive profile provides the decomposition above with navigation to source.

Supplementing benchmark's own diagnostics. *Setup.* Each query included in MAP has one or more annotated faulty operations. A method outputs a ranking of that query's operations, scored by average precision (AP equals reciprocal rank for one faulty operation) and averaged as MAP (Robertson 2008). We run the complete AgentProcessBench, HINTBench (the complete released test snapshot, 536 of the paper-reported 629 trajectories), and TraceElephant workloads. All profiler outputs are computed before any fault annotations are revealed. We report per-query AP averaged over 614, 400, and 220 queries included in MAP. The 522 trajectories without an annotated faulty operation are consumed for dataset coverage but excluded from MAP. The *direct diagnostic* is the per-operation output of each benchmark's own judge, including LLM-as-judge methods; it reads the full trace and reference answer before naming the responsible agent and decisive step. *Direct-only* ranks by that diagnostic alone; *Direct+AgentPProf* breaks ties by group score (vote mean for AgentProcessBench; 95% Wilson bound otherwise), with HINTBench field order fixed on 80 validation trajectories; *AgentPProf-only* uses only the profiler without the benchmark diagnostic (Figure 4).

Results. Direct+AgentPProf beats Direct-only by 0.031, 0.107, and 0.117 MAP on the three workloads (all statistically significant; 95% intervals in Appendix ; Figure 4). Grouping with retained evidence drives the localization gain. Semantic naming enables cross-run attribution (RQ1) and concentrates the reader's attention, reducing evidence opened from 65% to 53% (measured below).

Profile-guided reading on TraceElephant. On 220 queries, a Grok 4.5 (xAI 2026) agent-as-judge ranks operations by likely fault. Full-trace reading reaches MAP .502 at 12.6K tokens/query. A profile-guided reader selects at most five groups and reaches .455 while opening 53% of the source, versus .465 and 65% with raw-action names (intervals in Appendix). A per-query full read is also bounded by the model context window. *Takeaway.* Failure structure surfaced by the profile matches independent human labels across 435 trajectories. Profiles add localization signal on top of existing judges, and semantic names concentrate the agent's attention, reducing source characters opened from 65% to 53%.

Method	B^3 P	B^3 R	B^3 F1	Bound. F1
Codex segmentation	0.793	0.736	**0.764**	**0.480**
Statistical recurrence	0.782	0.575	0.663	0.266
Causal Qwen2.5-3B task stack	0.557	0.792	0.650	0.257
Raw-action grouping	0.891	0.388	0.541	—
Native source tree	0.975	0.249	0.397	0.259
Source-native step	0.983	0.221	0.361	0.246

Table 1: Agreement with human stages on all 405 CodeTraceBench trajectories.

Use case 4: guiding a real repair. The preceding experiments establish correspondence and inspection value; this case tests whether a profile finding can guide an effective repair. A standard AgentPProf profile over eight ToolSandbox scenarios isolates a recurring call-ID syntax failure (5/21 tool operations). A profile-only analyst—receiving only the pprof output, not raw traces—identifies the compatibility-layer boundary as the repair target. The derived one-line fix removes all syntax failures and, on 23 held-out confirmation scenarios (69 BEFORE/repair pairs), reduces agent-model tokens by 19.0% while passing a fixed $-.05$ quality threshold on official similarity (details in Appendix).

RQ3: How Accurate Are the Identifiers?

Setup. To evaluate whether recursive segmentation recovers task structure that humans would recognize, we compare segmentation output against 2,948 human-annotated stages over all 405 CodeTraceBench trajectories. F1 (Bagga and Baldwin 1998) asks whether operations that belong together end up grouped together; exact adjacent-boundary F1 (Ruokolainen et al. 2016) asks whether cuts land exactly where humans cut. Baselines receive identical inputs (Table 1). Each method output is evaluated at the level it predicts. Literal names use accuracy and macro-F1 (Lewis et al. 2004), permutation-invariant partitions by V-measure (Rosenberg and Hirschberg 2007) or ordinary B^3, and adjacent interval boundaries by exact precision, recall, and F1. For legacy field-valued datasets a conversion step maps each predicted group into the same interval-annotation format before AgentPProf folds the original additive weights. Codex (OpenAI 2025) receives each trajectory (prompts, commands, and output summaries, with no labels or scores visible) and emits sparse interval marks over 17,148 steps without access to stage annotations (loaded after all 405 outputs were fixed): 4,496 marks at semantic depth one (3), two (2,873), three (1,588), or four (32), with no prompt requesting a specific depth. Name normalization then maps the 3,895 free-form interval names to 783 stable short names (e.g., `diagnose authentication`) with zero adjacent display-path collisions, rejecting unresolved ones. Identical names across sessions merge so that identical paths fold in the final profile.

Results. Codex segmentation reaches 0.764 B^3 F1 and 0.480 boundary F1, beating the strongest automatic baseline

(multi-resolution recurrence) by +0.101 and +0.214 (intervals in Appendix). A stateful per-turn Qwen2.5-3B baseline reaches 0.650 B^3 F1, showing smaller models can also segment meaningfully. The marks conserve exactly 20,866 operations and 494,862,929 tokens. Predicted groups remain largely pure subsets of gold stages (B^3 precision 0.793), and the residual error is over-segmentation rather than missed transitions: when segmentation disagrees with human stages, it subdivides work rather than merging unrelated responsibilities, preserving per-group coherence (breakdown in Appendix). The interface generalizes across method families: on OSWorld-Human (WukLab 2025), a supervised Naive Bayes predictor reaches 0.816 B^3 F1, and a no-LLM recurrence segmenter reaches 0.786 (details in Appendix). The framework supports different segmentation scales: on OSWorld-Human, the unchanged Codex instruction uses coarser task-level boundaries (0.448 B^3 F1), while a supervised adapter matched to its finer action-group granularity reaches 0.816. For closed-label literal tags, a Qwen3.6-27B (Qwen Team 2026) classifier labels 1,012 AgentBoard goals (Ma et al. 2024) at 0.695 macro-F1, and 2,737 action labels from AutoCodeRover, OpenHands, and RepairAgent trajectories (Sajadi et al. 2026; Bouzenia and Pradel 2025) at 0.498 macro-F1 (details in Appendix). *Takeaway.* Recursive segmentation recovers human-recognizable task structure: 0.764 B^3 F1 against human stages, beating the strongest automatic baseline by +0.101. When segmentation disagrees with humans, it predominantly over-segments (subdivides work) rather than merging unrelated responsibilities, preserving per-group coherence. Non-LLM methods implementing the same interface remain viable when no model is available.

RQ4: What Is the Profiling Cost?

Setup. To evaluate whether profiling cost is practical, we measure segmentation time and tokens and profile replay speed. Cost separates into a one-time automatic segmentation pass per corpus and deterministic construction/replay afterwards.

Results. Segmenting all 405 CodeTraceBench trajectories with Codex (GPT-5.6-sol-high) completes in 37 minutes with up to four workers, averaging 29,754 input and 573 output tokens per trajectory. With marks fixed, construction scales linearly: the 27,765-operation union completes in 1.16 s at 465 MiB peak RSS, adding only 5.25 MiB (1.14%) over raw-action grouping (details in Appendix). Deterministic construction of the full 440 trajectories takes 0.26 s for operations and 0.25 s for tokens, so construction cost is dominated by the segmentation step while construction stays sub-second.

Reducing annotation cost. Automatic annotation dominates profile construction. Showing only a compact skeleton plus selected full results (SELECTIVE) instead of every turn's complete content (FULL) reduces provider tokens by 20.4% on 32 held-out CodeTraceBench task clusters while retaining 100% operation coverage and meeting a fixed $-.03$ quality threshold on B^3 and boundary F1 (details in Appendix).

Takeaway. Segmentation is a one-time pass (37 min for 405 trajectories); every later view, measure switch, and inspection costs about one second. Selective evidence reduces annotation tokens by 20.4% with complete coverage and maintained structural quality.

Related Work

Classic profilers fold runtime call stacks over stable code identity, and Pivot Tracing groups causally related measurements (Google 2024; Gregg 2011; Mace, Roelke, and Fonseca 2015). Agent observability platforms such as LangSmith, Langfuse, and Phoenix (LangChain 2024; Langfuse 2024; Arize AI 2024a) record spans with metadata for per-execution debugging. Analytics layers such as Datadog Patterns and LangSmith Insights (Datadog 2026; LangChain 2026) roll up cross-trace hierarchies or input distributions, but none constructs recursive semantic responsibility with conserved additive measures in a standard profile. Cross-run analyses (TraceProbe, Graphectory, Act·onomy, CHIEF, Hodoscope, TraceGraph) (Shu et al. 2026; Liu et al. 2026a; Gao et al. 2026; Wang et al. 2026c; Zhong, Saxena, and Raghunathan 2026; Nian et al. 2026) and per-run diagnosis and localization (Barke et al. 2026; Wang et al. 2026a; Mazaheri and Mazaheri 2026; Liu et al. 2026b; Mulian et al. 2026; Li et al. 2026; Fan et al. 2026; Wang et al. 2026b; Chen et al. 2026a; In et al. 2026) answer what an agent did and which step went wrong. AgentPProf instead asks how much of an additive resource a task or workflow phase is responsible for. It recovers variable-depth responsibility from source content (where Act·onomy fixes three levels and TraceProbe one), conserves measures exactly across count, token, and time widths, keeps LLM/tool calls as leaf nodes, and emits standard pprof. No compared system provides all four, and AgentPProf stays complementary to per-run diagnosis, as RQ2 measures directly.

Conclusion

Agent observability needs profiling, not only debugging. AgentPProf shows that the semantic operation stack model, driven by recursive operation segmentation, brings hierarchical attribution to agent trajectories. Against independent annotations, segmentation recovers human stage structure at 0.764 B^3 F1, profiles add localization signal on three complete public benchmarks, and one hierarchy replays across additive measures with exact conservation. A repair guided by a single profile reduces agent-side model tokens by 19.0% without material official-quality loss; selective annotation reduces provider tokens by 20.4% with complete coverage and maintained structural quality. Profilers and debuggers are complementary. The profile answers cross-run questions and guides inspection to the content worth opening, reducing opened content by 12 percentage points while reaching similar ranking. Multi-project evaluation and tracing-ecosystem integration are the most impactful next steps.

References

Anthropic. 2025. Claude Code: An Agentic Coding Tool. https://code.claude.com/docs/en/overview.

Arize AI. 2024a. Arize Phoenix. https://arize.com/docs/phoenix.

Arize AI. 2024b. OpenInference Specification. https://arize-ai.github.io/openinference/spec/.

Bagga, A.; and Baldwin, B. 1998. Entity-Based Cross-Document Coreferencing Using the Vector Space Model. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, 79–85.

Barke, S.; Goyal, A.; Khare, A.; Singh, A.; Nath, S.; and Bansal, C. 2026. AgentRx: Diagnosing AI Agent Failures from Execution Trajectories. arXiv preprint arXiv:2602.02475.

Bouma, G. 2009. Normalized (Pointwise) Mutual Information in Collocation Extraction. In *From Form to Meaning: Processing Texts Automatically, Proceedings of the Biennial GSCL Conference 2009*, 31–40.

Bouzenia, I.; and Pradel, M. 2025. Understanding Software Engineering Agents: A Study of Thought-Action-Result Trajectories. In *2025 IEEE/ACM 40th International Conference on Automated Software Engineering (ASE)*, 2846–2857.

Chen, M.; Wang, J.; Mu, F.; Wang, Y.; Liu, Z.; Feng, H.; and Wang, Q. 2026a. Seeing the Whole Elephant: A Benchmark for Failure Attribution in LLM-Based Multi-Agent Systems. arXiv preprint arXiv:2604.22708.

Chen, X.; Yin, Z.; He, S.; Huang, B.; Lei, S.; et al. 2026b. Safactory: A Scalable Agentic Infrastructure for Training Trustworthy Autonomous Intelligence. arXiv preprint arXiv:2605.06230.

Datadog. 2025. LLM Observability. https://docs.datadoghq.com/llm_observability/.

Datadog. 2026. LLM Observability Patterns. https://docs.datadoghq.com/llm_observability/monitoring/patterns/.

Fan, S.; Ye, X.; Huo, Y.; Chen, Z.-Y.; Guo, Y.; Yang, S.; Yang, W.; Ye, S.; Chen, J.; Chen, H.; Cong, X.; and Lin, Y. 2026. AgentProcessBench: Diagnosing Step-Level Process Quality in Tool-Using Agents. In *Proceedings of KDD*.

Gao, J.; Sun, K.; Huang, J.-t.; Van Koevering, K.; Ji, S.; Huang, H.; Shi, W.; Lu, Z.; Xiao, Z.; Khashabi, D.; and Dredze, M. 2026. How to Interpret Agent Behavior. arXiv:2605.13625.

Google. 2024. pprof. https://github.com/google/pprof.

Gregg, B. 2011. Flame Graphs. https://www.brendangregg.com/flamegraphs.html.

In, Y.; Tanjim, M.; Subramanian, J.; Kim, S.; Bhattacharya, U.; Kim, W.; Park, S.; Sarkhel, S.; and Park, C. 2026. Rethinking Failure Attribution in Multi-Agent Systems: A Multi-Perspective Benchmark and Evaluation. arXiv:2603.25001.

Laminar. 2025. Signals: Structured Event Extraction from Traces. https://docs.lmnr.ai/signals/introduction.

LangChain. 2024. LangSmith Observability. https://docs.langchain.com/langsmith/observability.

LangChain. 2026. LangSmith Insights. https://docs.langchain.com/langsmith/insights.

Langfuse. 2024. Langfuse Observability. https://langfuse.com/docs/observability/overview.

Lewis, D. D.; Yang, Y.; Rose, T. G.; and Li, F. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *Journal of Machine Learning Research*, 5: 361–397.

Li, H.; Yao, Y.; Zhu, L.; Feng, R.; Ye, H.; Wang, J.; He, Y.; Zou, P.; Zhang, L.; Lei, X.; Huang, H.; Deng, K.; Sun, M.; Zhang, Z.; Ye, H.; and Liu, J. 2026. CodeTracer: Towards Traceable Agent States. arXiv:2604.11641.

Liu, S.; Chen, Y.; Krishna, R.; Sinha, S.; Ganhotra, J.; and Jabbarvand, R. 2026a. Process-Centric Analysis of Agentic Software Systems. *Proceedings of the ACM on Programming Languages*, 10(OOPSLA1): 1961–1988.

Liu, Y.; Zhang, C.; Han, Z.; Liu, H.; Wang, Y.; Yu, Y.; Wang, X.; and Yin, Y. 2026b. TrajAD: Trajectory Anomaly Detection for Trustworthy LLM Agents. arXiv preprint arXiv:2602.06443.

Lu, J.; Holleis, T.; Zhang, Y.; Aumayer, B.; Nan, F.; Bai, H.; Ma, S.; Ma, S.; Li, M.; Yin, G.; Wang, Z.; and Pang, R. 2025. ToolSandbox: A Stateful, Conversational, Interactive Evaluation Benchmark for LLM Tool Use Capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 1160–1183. Association for Computational Linguistics.

Lù, X. H.; Kazemnejad, A.; Meade, N.; Patel, A.; Shin, D.; Zambrano, A.; Stańczak, K.; Shaw, P.; Pal, C. J.; and Reddy, S. 2025. AgentRewardBench: Evaluating Automatic Evaluations of Web Agent Trajectories. arXiv preprint arXiv:2504.08942.

Ma, C.; Zhang, J.; Zhu, Z.; Yang, C.; Yang, Y.; Jin, Y.; Lan, Z.; Kong, L.; and He, J. 2024. AgentBoard: An Analytical Evaluation Board of Multi-turn LLM Agents. In *Advances in Neural Information Processing Systems*, volume 37, 74325–74362.

Mace, J.; Roelke, R.; and Fonseca, R. 2015. Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems. In *Proceedings of the 25th Symposium on Operating Systems Principles*, 378–393.

MacQueen, J. B. 1967. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, 281–297.

Mazaheri, P.; and Mazaheri, K. 2026. AgentAtlas: Beyond Outcome Leaderboards for LLM Agents. arXiv preprint arXiv:2605.20530.

McCallum, A.; and Nigam, K. 1998. A Comparison of Event Models for Naive Bayes Text Classification. In *AAAI-98 Workshop on Learning for Text Categorization*, 41–48.

Mulian, H.; Zeltyn, S.; Levy, I.; Galanti, L.; Yaeli, A.; and Shlomov, S. 2026. AgentFixer: From Failure Detection to Fix Recommendations in LLM Agentic Systems. In *Proceedings of the 1st International Workshop on Agentic Engineering (AGENT '26, co-located with ICSE)*.

Nian, J.; Chen, K.; Zhang, G.; Cao, Y.; and Jiang, Y. 2026. TraceGraph: Shared Decision Landscapes for Diagnosing and Improving Agent Trajectories. arXiv:2605.31308.

OpenAI. 2025. Codex CLI: Lightweight Coding Agent That Runs in Your Terminal. https://github.com/openai/codex.

OpenTelemetry. 2024. OpenTelemetry GenAI Semantic Conventions. https://github.com/open-telemetry/semantic-conventions-genai.

Qwen Team. 2026. Qwen3.6-27B: Flagship-Level Coding in a 27B Dense Model. Official model card.

Robertson, S. 2008. A New Interpretation of Average Precision. In *Proceedings of the 31st Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, 689–690.

Rosenberg, A.; and Hirschberg, J. 2007. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 410–420.

Ruokolainen, T.; Kohonen, O.; Sirts, K.; Grönroos, S.-A.; Kurimo, M.; and Virpioja, S. 2016. A Comparative Study of Minimally Supervised Morphological Segmentation. *Computational Linguistics*, 42(1): 91–120.

Sajadi, A.; Nguyen, T.; Huynh, K.; Parra, E.; and Chatterjee, P. 2026. TraceView: Interactive Visualization of Agentic Program Repair Trajectories. arXiv:2606.22110.

Shu, R.; Chong, C. Y.; Zhou, X.; Peng, Y.; Wu, Z.; Han, X.; Zhuang, Z.; Yuan, G.; and Wang, Y. 2026. What Resolve Rate Hides: Trajectory Structure Diagnostics for Coding Agents. arXiv:2607.06184.

Wang, J.; Feng, Z.; Wu, J.; Li, R.; Xie, Q.; Ren, Y.; Zhu, H.; Han, X.; Meng, F.; Feng, J.; and Liu, J. 2026a. Where Do Deep-Research Agents Go Wrong? Span-Level Error Localization in Agent Trajectories. arXiv preprint arXiv:2606.02060.

Wang, J.; Hou, J.; Wang, F.; Jian, P.; Bao, C.; and Lv, Z. 2026b. HINTBench: Horizon-Agent Intrinsic Non-Attack Trajectory Benchmark. arXiv preprint arXiv:2604.13954.

Wang, X.; Wang, B.; Lu, D.; Yang, J.; Xie, T.; et al. 2025. AgentNet. https://huggingface.co/datasets/xlangai/AgentNet.

Wang, Y.; Wu, W.; Wang, J.; and Wang, Q. 2026c. From Flat Logs to Causal Graphs: Hierarchical Failure Attribution for LLM-based Multi-Agent Systems. arXiv:2602.23701.

WukLab. 2025. OSWorld-Human. https://github.com/WukLab/osworld-human.

xAI. 2026. Introducing Grok 4.5. https://x.ai/news/grok-4-5.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Hua, T. J.; Cheng, Z.; Shi, D.; Tong, J.; Lu, K.-W.; Garg, V.; Wang, Y.; Dai, Z.; Li, F.; Bisk, Y.; and Yu, T. 2024. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems 37 (NeurIPS), Datasets and Benchmarks Track*.

Yang, J.; Jimenez, C. E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; and Press, O. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS)*.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E. P.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS), Datasets and Benchmarks Track*.

Zheng, Y.; Hu, Y.; Yu, T.; and Quinn, A. 2025. AgentSight: System-Level Observability for AI Agents Using eBPF. In *Proceedings of the 4th Workshop on Practical Adoption Challenges of ML for Systems (PACMI '25, co-located with SOSP)*.

Zhong, Z.; Saxena, S.; and Raghunathan, A. 2026. Hodoscope: Unsupervised Monitoring for AI Misbehaviors. arXiv:2604.11072.

Technical Appendix

Reproducibility Scope and Data

The eight primary public benchmarks are AgentRewardBench, AgentProcessBench, HINTBench, TraceElephant, CodeTraceBench, OSWorld-Human, AgentBoard, and the TraceView/ASE action corpus. Mind2Web and ScienceWorld provide two additional field-mapping checks. No novel benchmark dataset is introduced. Public benchmark targets, human stages, outcomes, and scores are used only by scorers after each method's output is fixed.

The paper's theoretical contribution is the semantic operation stack model, specified by the formal definitions in the Design section. AgentPProf is a 9.8K-LOC implementation. Its executable source, preprocessing scripts, and released-data manifests accompany the paper in the code-and-data package.

Automatic Segmentation and Identity

The direct annotation backend is the OpenAI Codex CLI (version 0.145.0, model `gpt-5.6-sol`, high reasoning effort, default decoding, sandboxed non-interactive mode). Each worker receives one source-only trajectory packet, makes one annotation call with at most one format retry, and shares no state with other workers. Prompt and response previews are truncated to 1,800 characters, tool commands to 600, and status previews to 200; additive measures come from structured counters and are not truncated. The fixed instruction is reproduced in the main paper. It emits sparse path changes whose names begin with a verb and contain one to three words; intervening steps inherit the preceding path.

Operationally, each call is one isolated backend request. Here, recursion refers to the variable-depth nested decomposition constructed by the annotation agent within that request, rather than to a sequence of backend requests. "Iteratively" refers to revising the mark set before returning one final sparse annotation file.

Before folding, one fixed source-only action–object map canonicalizes free-form interval names. The map is implemented as ordered verb, object, and qualifier phrase tables with fixed aliases and stop words. It reads operation names and sparse source-only marks, independent of task IDs, benchmark names, outcomes, target stages, and score rows. Unmatched source words remain available to the boundary-safe refinement. If canonicalization would make adjacent display paths equal, the refinement retains the minimum source words needed to distinguish them; unresolved collisions fail closed. Equal canonical names receive one stable operation ID across sessions. On the full CodeTraceBench dataset, the map covers all 3,895 free-form names, applies 503 name-specific refinements, and produces 783 stable names. Refinement reduces 329 initial adjacent collisions to zero, so the observed fail-closed count is zero, while preserving every temporal mark.

The non-LLM recurrence backend counts adjacent action transitions in reference sessions and scores each transition by normalized pointwise mutual information (NPMI),

$$\mathrm{NPMI}(a,b) = \frac{\ln[p(a,b)/(p_L(a)p_R(b))]}{-\ln p(a,b)},$$

where all probabilities use the same transition sample space (Bouma 2009). Weighted one-dimensional k-means with $k = 2$ initializes at the minimum and maximum scores, assigns distance ties to the lower center, and uses the converged midpoint as the cutoff (MacQueen 1967). The same deterministic procedure is applied to action-changing transitions. Same-action pairs use the global cutoff; different-action pairs use the smaller cutoff. A detailed `(action, action_detail)` arm may remove a coarse boundary but never add one; missing or unseen detail falls back to the coarse decision.

RQ1 Details

The Git case's 735 source nodes comprise 3 sessions, 3 prompts, 240 LLM calls, and 489 profiled tool operations.

System-effect alignment. For RQ1's controlled cross-layer test, AgentSight first matches each timestamped process, file, or network event to the same-PID process instance whose lifetime contains it. It then links that process to a recorded tool call by an explicit tool/event ID when available, or by recorded root-process identity and overlapping process/tool lifetimes; descendants are accepted only within that tool window. The matched wrapper tool identifies the task-level semantic interval, while unmatched events remain unassigned. Across 20 real Codex tasks, this rule attributes 1,520 of 1,574 scoped process/file effects (100% precision, 96.57% recall) and rejects all 1,629 concurrent-control effects. AgentPProf then folds exactly those joined events as unit-weight operations, preserving all 1,520 samples and every task-category total.

On AgentRewardBench, ranking operations by count versus tokens yields mean Kendall's tau-b 0.886 with 95% interval [0.857, 0.915] over the 77 tasks with at least three distinct operations. Pooled ranking agrees at tau-b 0.929. The 13% of tasks (10 of 77) where agreement falls below 0.7 are exactly where multi-measure profiling adds value.

RQ2 Protocol

AgentRewardBench pairing enumerates every failed–successful combination within each task. The 202 distinct successful and 238 distinct failed trajectories are therefore reused when a task has multiple runs, yielding 338 pair occurrences over the 125 mixed-outcome tasks. The signed aggregate is pair-occurrence weighted and contains all 338 failed sides and all 338 successful sides.

Within each workload, operations, target-blind paths, benchmark judge/localizer predictions, and scoring rules are fixed before test targets are loaded. AgentProcessBench averages frozen judge votes within a group. HINTBench and TraceElephant assign each root-to-frame prefix the 95% Wilson lower bound of its frozen positive-prediction fraction; an operation receives the maximum score over prefixes containing it. The released HINTBench snapshot contains 536 of the paper-reported 629 trajectories. A separate 80-trajectory validation snapshot selects one of 24 field orders before the 536 test trajectories are scored.

The three localization workloads contain 1,756 trajectories. Their 1,234 queries included in MAP contribute 614,

400, and 220 AP values, respectively, while the other 522 trajectories without an annotated faulty operation remain in the dataset inputs. AP equals reciprocal rank for a query with one relevant operation and also handles queries with multiple relevant operations. MAP is the arithmetic mean over these queries because AP is undefined without a relevant item. The reported Direct+AgentPProf-minus-Direct-only intervals use 10,000 paired resamples of trajectory clusters within benchmark-defined groups. AgentProcessBench uses seed 20260723, HINTBench uses 20260823, and TraceElephant uses 20260923.

The 95% confidence intervals for Direct+AgentPProf-minus-Direct-only are [0.024, 0.039], [0.093, 0.120], and [0.088, 0.148] for the three workloads respectively. For the reading study, paired raw-minus-semantic differences are $+.010$ [$-.021$,$+.042$] MAP and $+.120$ [$+.103$,$+.137$] content fraction.

RQ2 Repair Case Details

Setup. We run ToolSandbox's official stateful scenarios and evaluator with a local tool-using agent (Lu et al. 2025). One standard AgentPProf pprof over eight real no-policy BEFORE traces contains 21 tool operations. Stock pprof shows that 5/21 carry `call_id:invalid` and `result:invalid-call-id`; four of those failures are followed by an exact same-tool, same-argument recovery repeat.

Analyst and repair. An independent analyst—a profile-only Codex agent that receives only the pprof output and experiment plan, without access to the raw traces or ToolSandbox internals—prioritizes the compatibility-layer call-ID boundary as the repair target rather than suppressing the useful retries. Inspection of that boundary reveals that the compatibility converter interpolates an opaque protocol call ID into executable Python source. The repair retains the original ID in assistant/tool protocol history but derives a separate Python-identifier-safe internal variable for execution; it does not change tool names, arguments, order, scenario state, user, agent policy, or evaluator.

Validation. Exact-state replay of all 21 profiled operations reproduces every original BEFORE response and post-state, removes all five affected syntax failures, preserves all 21 protocol IDs, and leaves the responses and post-states of all 16 valid-ID controls unchanged. After development on eight scenarios, the unchanged repair runs on 23 disjoint confirmation scenarios, each repeated three times. All 69 BEFORE/repair pairs complete with the official evaluator and enter the full-run analysis (Table 2).

Interpretation. Agent-side model tokens fall by 19.0%, and model calls, tool calls, and turns fall with them. Official similarity changes from .831 to .857; its scenario-cluster interval passes the fixed $-.05$ quality threshold but crosses zero, so this is an efficiency result without quality loss rather than a significant outcome-improvement claim. The server does not request-seed its opaque-ID allocator, so the two methods encounter different numbers of raw invalid IDs; the 19.0% reduction is the full-run effect, not a per-invalid-ID saving.

Metric	BEFORE	Repair
Raw invalid IDs	28	19
Python syntax failures	28	**0**
Agent model tokens	211,222	**171,139**
Agent model calls	255	**227**
Tool calls	159	**137**
Turns	457	**401**
Official similarity	.831	.857

Table 2: Profile-guided repair on 23 ToolSandbox confirmation scenarios with three repetitions each (69 pairs, 138 episodes). Agent-token AFTER/BEFORE ratio is .810 (scenario-cluster 95% interval [.734,.921]); the official-similarity delta is +.026 [-.022,.076].

For the TraceElephant reading study, the Grok 4.5 CLI reader is invoked with tools and subagents disabled. Full-trace reading uses one single-turn call per query. Profile-guided reading first exposes only the target-blind operation skeleton; the reader selects at most five groups, after which a second single-turn call receives only those groups' source-visible evidence. Unranked operations are appended in original order. Semantic and raw-action conditions differ only in the operation names used by the skeleton. The observed reduction from 65% to 53% is a paired difference of 0.120, or 12 percentage points.

RQ3 Protocol

For CodeTraceBench, all 405 annotations are materialized before the 2,948 human stage labels are loaded by the scorer. Codex receives prompts, commands, and output summaries, but no label or score. It emits 4,496 sparse marks over 17,148 source-native steps. CodeTraceBench stage labels define trajectory-local temporal groups, so B^3 scores the resulting operation partition and exact adjacent-boundary precision, recall, and F1 score the transition decisions. Paired task-cluster intervals preserve the benchmark's task grouping. Literal-name accuracy is evaluated separately on the complete AgentBoard and TraceView/ASE datasets.

The 95% paired task-cluster intervals for Codex improvement over baseline are [0.087, 0.116] for B^3 F1 and [0.193, 0.235] for boundary F1. The stateful Qwen2.5-3B baseline that makes push/replace/stay/pop decisions reaches 0.650 B^3 F1 and 0.257 boundary F1. Boundary error breakdown: recall 0.626, precision 0.389.

The OSWorld-Human study uses all 287 sessions, containing 3,978 operations, 3,691 adjacent pairs, and 2,042 human groups. The supervised Bernoulli Naive Bayes predictor (McCallum and Nigam 1998) fixes its model family, nine input fields, adjacent-pair features, and training procedure. Five session-held-out folds fit parameters and the decision threshold on training sessions and predict each held-out session exactly once. Label-free recurrence uses only visible transitions from the other four folds. Reference-calibrated recurrence additionally fits one scalar NPMI cutoff using the training-fold group annotations; it never reads held-out

groups.

Detailed OSWorld-Human results: supervised Naive Bayes reaches 0.739 boundary F1 / 0.816 B^3 F1; calibrated recurrence reaches 0.734 / 0.801; no-LLM recurrence segmenter reaches 0.680 / 0.786 (strongest control: 0.645 / 0.678). On Mind2Web (9 sessions) and ScienceWorld (100 sessions), unsupervised TF-IDF/K-Means reaches V-measure 0.557 and 0.815 respectively. The same Codex instruction without adaptation uses coarser task-level boundaries (0.448 B^3 F1 on OSWorld-Human).

The evaluation-only Qwen3.6-27B llama.cpp backend classifies all 1,012 AgentBoard goals from goal text and the nine declared family descriptions. Three complete runs produce identical assignments. The action-label adapter uses the fixed eight TraceView action definitions and the current thought/action only; two complete runs over all 2,737 labels produce identical assignments. Literal labels use accuracy and macro-F1, partitions use V-measure or ordinary B^3, and transitions use exact adjacent-boundary precision, recall, and F1.

AgentBoard goal classification: 0.695 macro-F1 and 0.733 accuracy (majority baseline 0.044 / 0.248). Action-label classification on AutoCodeRover, OpenHands, and RepairAgent trajectories: 0.498 macro-F1 and 0.628 accuracy (majority baseline 0.061 / 0.323). The 0.437 macro-F1 gain has a 95% bootstrap interval of [0.380, 0.494]. The fixed runtime is llama.cpp version 9870 at revision `2d973636e` with the Qwen3.6-27B Q4_K_M artifact. It uses one 4,096-token slot, full GPU offload, Jinja templates, reasoning disabled, and both RAM cache and prompt reuse disabled.

RQ4 Construction Cost

Workload	Ops	Semantic (s)	Raw action (s)	Peak RSS (MiB)
AgentRewardBench	729	0.04	0.03	19.3
SATraj-OS	4,285	0.17	0.14	73.9
OSWorld-Human	6,010	0.24	0.21	109.1
AgentNet	16,741	0.70	0.58	279.3
Union	27,765	1.16	0.97	465.2

Table 3: RQ4 profile-construction cost (Chen et al. 2026b; Wang et al. 2025): three-run median times and largest semantic-profile peak RSS. The 729-operation AgentRewardBench row is the fixed public release sample used for cost measurement, distinct from RQ2's 7,229-operation mixed dataset.

Reducing Annotation Cost

We compare two annotation methods: FULL shows every turn's complete content to the LLM agent, while SELECTIVE shows only a compact skeleton (intent, action, progress) for each turn plus full results for a selected subset. Both use the same model, instruction, output schema, and retry limit; SELECTIVE still returns marks for the complete trajectory.

The confirmation dataset contains 32 `task_name` clusters balanced across four agent frameworks, covering 1,639 operations. Table 4 counts input plus output tokens from every attempt, including one format retry per method.

Metric	FULL	SELECTIVE	Difference
Provider tokens	984,321	**783,121**	-20.4%
B^3 F1	.732	.758	$+.026$
Boundary F1	.429	.453	$+.024$
Covered operations	1,639	1,639	0
Provider calls	33	33	0

Table 4: Paired annotation cost on 32 CodeTraceBench `task_name` clusters. The SELECTIVE/FULL token ratio is .796 (task-cluster 95% interval [.734,.863]); B^3 and boundary deltas meet the fixed $-.03$ quality threshold.

SELECTIVE reduces tokens by 20.4%, and the task-cluster ratio interval has an upper bound below one. Both methods retain 100% operation coverage; SELECTIVE's B^3 and boundary F1 point estimates are higher, and their lower confidence bounds exceed the fixed $-.03$ quality threshold.

Runs, Uncertainty, and Computing Environment

The CodeTraceBench direct backend completed 405 trajectories in 415 calls. The ten calls after each trajectory's first include format retries and one authorized repair call for trajectory 53. All 405 final annotations pass the same schema validation. RQ2 fixes one profile and one benchmark prediction per trajectory, then computes uncertainty with 10,000 paired resamples. The TraceElephant reader makes one call per stage and query. OSWorld-Human uses five held-out folds. The closed-label repeat counts are given above. RQ4 construction times are medians of three complete runs; peak RSS is the largest observed semantic-profile peak. Once marks are fixed, projection, folding, serialization, and pprof readback are deterministic.

The RQ4 construction measurements use `agentpprof 0.2.37` on a 24-core Intel Core Ultra 9 285K with 125 GiB RAM and Linux 6.15.11. The 27,765-operation union completes semantic construction in 1.16 s at 465.2 MiB peak RSS, versus 0.97 s for raw-action grouping. This observed range spans 729 to 27,765 operations, with 23,935 operations/s at the union scale.

The automatic CodeTraceBench pass consumes 12,050,384 input tokens, including 6,008,320 cached tokens, and 231,886 output tokens. These totals average 29,754 input and 573 output tokens over 405 trajectories. Up to four isolated workers complete 8,689.405 s of summed backend requests in 2,215.858 s of active wall time, and the deterministic downstream pipeline takes 11.516 s.

The separate AgentRewardBench pass processes 440 trajectories in 12 outcome-blind batches on a fixed two-worker schedule. It completes in 3,521.621 s of end-to-end wall time and consumes 12,039,417 input tokens, including 10,929,408 cached tokens, and 312,433 output tokens. These totals average 27,362 input and 710 output tokens per trajectory. All 12 batches pass profile validation. Deterministic construction then takes 0.26 s for operation weights and 0.25 s for token weights. The token counters separate cached

input, uncached input, output, and reasoning output independently of provider pricing.

Privacy, Ethics, and Responsible Use

Agent histories can contain prompts, model responses, source code, file paths, commands, network targets, and system-effect records. The evaluation keeps local case-study histories local and reports aggregate measurements and task-level paths. The reproduction package uses the cited public datasets.

Automatic annotation applies fixed preview bounds and supports field minimization, credential and personal-data filtering, and approved provider-retention policies. Local model serving and the non-LLM recurrence backend provide local-processing options. Profiles, annotations, and intermediate packets receive the same access control and retention policy as their source traces.

The intended use is authorized engineering inspection, resource analysis, and source-linked debugging. Operators scope capture to systems and trajectories they are authorized to observe, and human review remains part of operational access-control and safety decisions.

The reported benchmark results characterize the named datasets. Reusing fixed annotations avoids repeated inference, while the non-LLM backend offers a deterministic alternative.

Selective Annotation Protocol

SELECTIVE annotation retains a compact skeleton for every turn: intent, planned action, progress indicator, and operation ID. Only turns selected by a deterministic source-visible rule receive full visible results.

The selector uses only source-visible information: result length, error/pass/timeout status, test/build/verify actions, and progress changes. It does not read human stages, rewards, final outcomes, or existing model annotations. Both FULL and SELECTIVE use the same model (Codex GPT-5.6-sol-high), the same one-call output format, the same retry limit (one format retry), and the same downstream scorer.

The confirmation dataset contains 32 `task_name` clusters balanced across four agent frameworks (AutoCodeRover, OpenHands, RepairAgent, CodeAct), covering 1,639 operations. Provider tokens are input plus output tokens from every attempt, including one format retry per method.

The SELECTIVE/FULL token ratio is .796 (task-cluster bootstrap 95% interval [.734,.863]). B^3 difference is +.026 and boundary F1 difference is +.024; both exceed the preset $-.03$ quality threshold.

Reproducibility Checklist Notes

Checklist items concerning novel datasets are not applicable because the paper introduces no benchmark dataset. The paper's theoretical contribution is the semantic operation stack model, specified through the Design section's formal definitions. Its claims require neither theorems nor proofs. This appendix supplies the final algorithm settings, target-blind freezing rules, evaluation metrics, run counts, and measured RQ4 environment. The accompanying code-and-data package supplies preprocessing and experiment source.