

TraceForge: A Human-Gated System for Auto-Hardening Tool-Using LLM Agents into Immutable Workflow DAGs

Santhiya Rajan

Multiverse Computing, Paseo de Miramón 170, 20014 Donostia / San Sebastián, Spain
santhiya.rajan@multiversecomputing.com

Abstract

Tool-using language agents are flexible but non-deterministic: an orchestrator LLM re-plans on every request, adding latency, cost, and paths that are hard to audit. We demonstrate **TraceForge**, an on-premise system that observes how an agent *actually* solves a recurring task and, on operator request, hardens it into a deterministic, immutable workflow DAG. TraceForge attaches to the OpenTelemetry traces an agent already emits (zero code changes), reconstructs each run’s execution DAG, and runs a human-gated loop: a statistical aggregation gate (proposes a DAG only once enough runs agree), a Judge-LLM reflection step, and a human freeze gate that writes a versioned, immutable specification with lineage and rollback. The pipeline is topology-agnostic—applied to single-agent tool-callers and multi-agent orchestrators alike—and all components run on-premise with no external dependencies. On held-out requests, frozen workflows reproduce the dynamic agent’s outputs exactly while removing every planner-LLM call and cutting latency $1.39\times$. The entire loop is driven from a browser console with no coding, and is shown end-to-end in the accompanying screencast.

1 Introduction

Modern tool-using agents built on frameworks such as ReAct (Yao et al., 2023b), Toolformer (Schick et al., 2023), and multi-agent orchestrators (Wu et al., 2023; LangChain, 2024) let a language model decide, at runtime, which tools to call and in what order. This dynamism is essential while a use case is explored, but a liability once a task recurs in production: the orchestrator still consults an LLM to re-derive a plan on every turn, adding latency and cost, yielding non-deterministic, hard-to-audit execution, and leaving no stable artifact a compliance reviewer can freeze—first-class blockers in regulated, latency-sensitive settings (clinical decision support, finance, industrial control).

That recurrence is measured, not assumed: after only five steps, at least 14.3% of APPWORLD tasks follow an equivalent trajectory—and that count is conservative (Abuzakuk et al., 2026)—because agents *repeatedly solve similar task instances from scratch* (Guo et al., 2026); individual trajectories carry redundant steps that consume substantial resources while contributing little to completion (Hu et al., 2026), and the resulting trace collections are correspondingly redundant (Chang et al., 2026). The waste is paid in inference cost before it is paid in time: every re-derived plan bills a fresh planner prompt of system instructions, tool schemas, and accumulated history, and the latency the user feels is a consequence of that spend. Deduplicating even a recurring *subsequence* of tool calls cuts token usage by up to 11.9% (Abuzakuk et al., 2026); freezing the whole recurring composition removes the planner call outright (§8).

A natural resolution separates *exploring* a use case from *running* it: keep the dynamic agent for novel tasks, execute a fixed workflow for recurring ones. The open question is obtaining that workflow without hand-authoring it or trusting one idiosyncratic run. Our observation: the answer is already emitted—agents instrumented with OpenTelemetry produce standardized GenAI spans (OpenTelemetry Authors, Cloud Native Computing Foundation, 2024) for every invocation, model call, and tool execution, a faithful record of *how the agent actually behaved* across many runs.

We demonstrate **TraceForge**, a system that turns that trace stream into governed, deterministic workflows. It is deliberately conservative and human-in-command: it never rewrites an agent, never freezes from insufficient evidence, and never commits a specification without explicit human approval. Concretely, this demonstration contributes:

1. **Zero-instrumentation capture and trace-to-DAG reconstruction**—a trace tap on the agent’s existing OpenTelemetry provider recovers per-

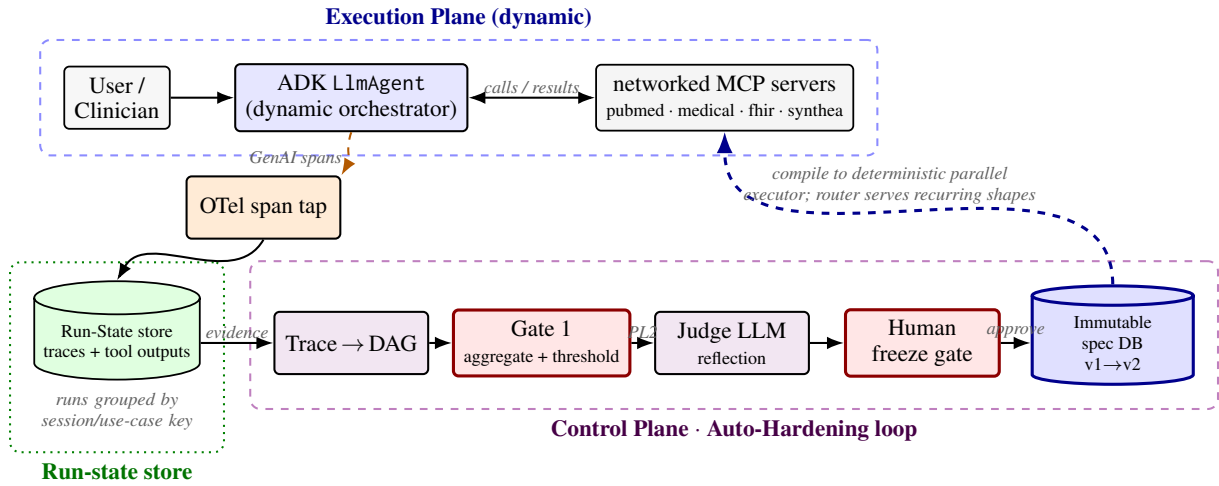


Figure 1: TraceForge architecture. The *execution plane* runs the unchanged dynamic agent (an ADK LLM Agent orchestrating MCP servers); its OpenTelemetry GenAI spans are tapped into the run-state store. Two persistent stores back the loop: the *run-state* store holds the captured traces—the hardening evidence, including the tool outputs a reviewer inspects at the freeze gate—and the *immutable spec* store holds frozen workflow versions with lineage. Runs are grouped into a use case by the session/use-case key carried on each span (not a separate store). The *control plane* reconstructs each run’s DAG, applies the three hardening gates, and writes an immutable, versioned specification that is compiled back into a deterministic workflow for the execution plane.

run tool trajectories from the GenAI span tree with no code edits.

2. A **human-gated hardening loop** with three gates (statistical aggregation, Judge-LLM reflection, human freeze) writing to an *immutable*, versioned store with lineage and rollback, and **topology-agnostic generalization** to multi-agent, multi-hop systems via hierarchical reconstruction with parallel-branch detection.
3. A **2×2 study** over two clinical systems and a **real-trace validation** over 610 public trajectories (all analyzable ones convert to DAGs) plus a labelled golden set and a **held-out frozen-vs-dynamic comparison** establishing output equivalence and the efficiency gain—exposed through a live, on-premise *bring-your-own-trace* console.

All components run locally, a hard requirement in the privacy-constrained domains we target. **Who this is for.** TraceForge is built for the ML/platform and MLOps engineers who put tool-using agents into regulated, audit-sensitive production, and for researchers working on agent trace analysis and process mining. A screencast of the live walk-through (§7) accompanies this submission, and the console is demonstrated live at the conference.

2 System Overview

TraceForge follows a two-plane design (Figure 1). The *execution plane* is the user’s existing agent, run

unmodified. The *control plane* is the TraceForge auto-hardening loop. The two planes are coupled only through telemetry: the control plane reads what the execution plane emits and, when a workflow is frozen, returns a deterministic specification that a hardened runtime can execute in place of the dynamic orchestrator.

This separation is the core design principle: TraceForge is a *drop-in* layer (an agent adopts it by exposing traces, not refactoring), and hardening is strictly additive—a frozen workflow is a new artifact, while the dynamic agent still serves novel requests.

Persistent stores. Two stores back the loop (Figure 1): *run-state* holds captured traces—the evidence Gate 1 aggregates and the audit record justifying a freeze—and the *immutable spec* store holds each frozen version with its lineage. Runs group into a *use case* by the session key carried on every span, not a separate store.

3 From Traces to DAGs

Capture. Agents instrumented per the OpenTelemetry GenAI conventions emit a span tree per turn: a root `invoke_agent` span containing `call_llm` and `execute_tool` children, the latter carrying attributes such as `gen_ai.tool.name` (OpenTelemetry Authors, Cloud Native Computing Foundation, 2024). TraceForge registers an

additional span processor on the tracer provider, so spans still flow to the operator’s normal backend *and* are persisted locally, keyed by trace. The processor only *reads* finished spans, so attaching it requires no agent-logic changes; for agents we cannot modify, a thin wrapper installs the tap before import.

Reconstruction. For each trace, TraceForge extracts the tool-execution spans, orders them by start time, and links them into a tool DAG (App. Figure 5), recovering the linear execution order—which compiles cleanly to a sequential workflow—while retaining the full span tree so later analysis can infer independent branches and data dependencies from read/write sets. With explicit read/write attributes (emitted by instrumented agents), the dependency DAG is exact; on *un-annotated* public traces we recover dependencies heuristically from argument/result token overlap, defaulting to sequential order absent evidence—so parallelism is claimed only from positive value-flow evidence (Section 8). The result is a compact, canonical description of one run.

4 Gated Auto-Hardening

Hardening is triggered *only* when an operator explicitly selects a use case for conversion; it then proceeds through three gates.

Gate 1: Aggregation and Sample Threshold. A workflow must never be frozen from a single run. TraceForge groups captured traces by use case, clusters them by tool trajectory, and selects the modal (most frequent) trajectory as the canonical candidate, proposing a DAG only once there are $\geq N$ runs with support (fraction agreeing) $\geq \theta$; otherwise it reports precisely why hardening is blocked. The candidate is emitted as a structured workflow specification with per-step risk labels, flagging sensitive tools (e.g. report generation, record creation) for human-in-the-loop and reflection at runtime.

Gate 2: Judge-LLM Reflection. Following LLM-as-a-judge (Zheng et al., 2023) and self-reflection (Shinn et al., 2023), a judge reviews the candidate DAG and returns APPROVE, REVISE, or REJECT with reasons and a confidence score. The judge is pluggable. The default *offline* judge is deliberately a transparent rule, not a model: it scores $\min(1, 0.4 + 0.6s)$ in the support s and then deducts for degenerate structure (empty tool

names, immediately repeated steps, over-long trajectories), approving above 0.75. Being a monotone function of s , it cannot contradict Gate 1 on stable candidates—so in the 2×2 study (all cells $s = 0.8$) it necessarily returns the same 0.88, and its value there is a structural sanity check, not an independent vote. Pointing the judge at an OpenAI-compatible endpoint (e.g. the agent’s own on-premise model) substitutes natural-language critique. Either way the judge is advisory—it can block a freeze but never authorize one; the binding automatic check is behavioural equivalence.

Gate 3: Human Freeze Gate and the Immutable Store. Only after Gates 1–2 pass is the operator asked, in plain language, to freeze the enumerated workflow. On approval, TraceForge appends to an *immutable*, versioned store: every freeze creates a new version ($v_1 \rightarrow v_2 \rightarrow \dots$) with a `parent_version` link, the justifying evidence run IDs, and the judge verdict; exactly one version is *live*, and rollback re-points it without mutating any stored specification—a complete, auditable lineage from raw traces to governed workflow.

Behavioural equivalence. Structural agreement is not correctness: a candidate could drop a step, mis-order a dependency, or parallelize something that was not independent and still look plausible. Before any freeze that is not individually approved by a human—i.e. whenever the router graduates a shape automatically—TraceForge replays *held-out* inputs through both the dynamic agent and the candidate plan and requires the outputs to match exactly, refusing the freeze otherwise. This is the check that makes unattended graduation safe, and Section 8 reports it.

5 Multi-Agent and Multi-Hop Hardening

The reconstruction of Section 3 treats a run as a flat tool trajectory, which is exact for a single agent. Production systems increasingly use a *single orchestrator that delegates to specialist sub-agents*; their traces are nested (an `invoke_agent` span per agent, each with its own tool children), so flattening them discards the very structure that makes the hardened workflow useful. TraceForge therefore reconstructs a *hierarchical* agent graph, and the identical gates, judge, and immutable store then operate on it.

Hierarchical reconstruction. From the span tree (Figure 2) we (i) treat every `invoke_agent` span

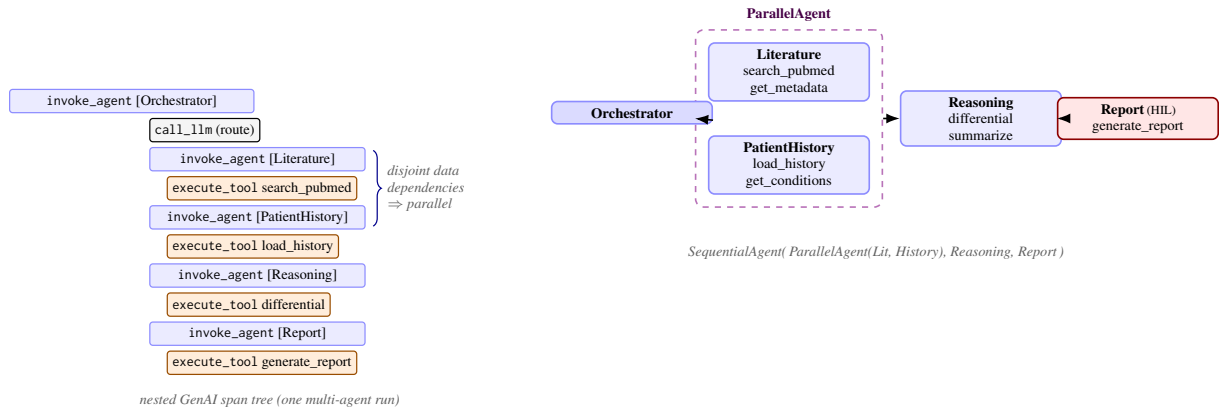


Figure 2: Multi-agent hardening. The nested span tree of one orchestrated run (left) is reconstructed into a hierarchical composite workflow (right): tools are attributed to their owning agent, and the LITERATURE and PATIENTHISTORY specialists—whose spans overlap in wall-clock—are recovered as a parallel branch, followed by sequential REASONING and a human-in-the-loop REPORT step.

as an agent node; (ii) attribute each tool span to its *nearest enclosing agent*; and (iii) order sibling sub-agents by start time, grouping those whose intervals *overlap* into a parallel branch (a timing signal, degrading conservatively to sequential when unreliable) and leaving the rest sequential. The result—a structure-aware signature of agents, tool steps, and `;/||` composition—clusters runs in Gate 1 exactly as the linear signature does for single agents. The frozen specification itself is framework-agnostic: our current runtime realizes it with ADK `SequentialAgent/ParallelAgent` composites carrying per-step human-in-the-loop flags—deterministically ordered agents, not a DAG engine—but the same specification plugs directly into conventional DAG frameworks such as Apache Airflow, Prefect, Dagster, or Kubeflow Pipelines, and, because every frozen version persists in the immutable spec store, it can also be exported from that database and deployed as an independent, standalone system.

Two systems, one method. We exercise this on a medical *multi-agent* system (Figure 2: an orchestrator delegating to four specialists) alongside the original single-agent tool-caller, each behind one entry point; *multi-hop* manifests as a longer tool chain (single agent) or deeper, partly-parallel delegation (multi-agent). The reconstruction code, thresholds, and gates are identical across both systems—we do not hand-tune per-system parsing rules—though both remain within the OpenTelemetry GenAI span schema TraceForge assumes throughout.

6 Implementation

TraceForge is Python with no heavyweight dependencies (OpenTelemetry SDK and `sqlite3` for capture; reconstruction, gates, and the spec store are dependency-free); the console is a single FastAPI service. The single-agent system is a Google ADK (Google, 2025) `LlmAgent` orchestrating four MCP (Anthropic, 2024) servers on an on-premise Gemma-4-E4B model (Figure 3); the multi-agent system wires specialist `LlmAgents` as sub_agents of the same package. **Live vs. offline numbers:** the live demo (Figures 3–4) exercises the real ADK/LiteLLM/MCP path end-to-end; every *tabulated* number in Sections 8–8 instead regenerates from a deterministic reference runtime that emits the identical GenAI spans a live run produces, so results are reproducible offline without any external service or model call. Reconstruction, the gates, the judge, and immutable versioning are each validated end-to-end.

7 Demonstration Walkthrough

Everything below is driven live from the browser console, entirely through point-and-click interaction. The demo is a single user navigation journey: the user opens the console in a browser and then:

1. **Talks to the dynamic agent.** Typing a clinical request into the *Live control plane* (Figure 3) runs the real ADK/LiteLLM/MCP agent on a local model; TraceForge captures the trace and reconstructs the run’s DAG in place.
2. **Opts a use case in.** Discovered use cases show their run count and canonical trajectory. Nothing hardens until the user ticks *Convert to DAG*;

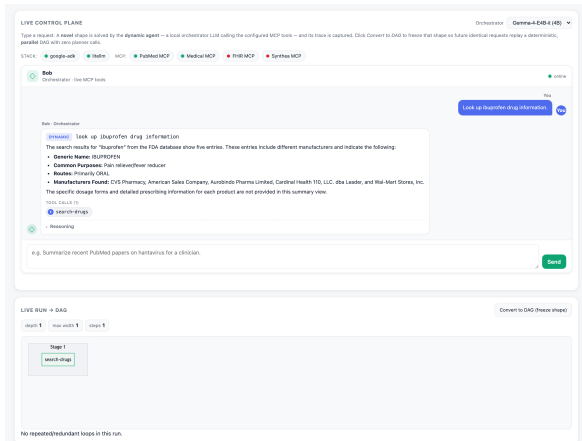


Figure 3: **Live control plane.** The user types a request; the local Gemma-4-E4B orchestrator solves it *dynamically* by calling a medical MCP tool (search-drugs, shown in the TOOL CALLS strip), and the captured run is reconstructed on the spot into the tool DAG below—ready for one-click freezing.

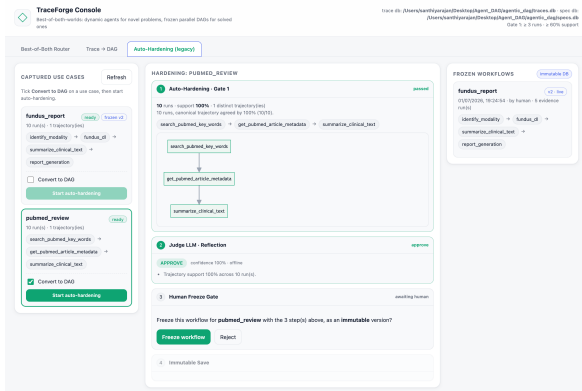


Figure 4: **The human-gated hardening loop** (pubmed_review): Gate 1 aggregates 10 runs into a canonical DAG at 100% support, the offline judge approves, and the human freeze gate is armed. A previously frozen use case (fundus_report) sits at v_2 live in the immutable store on the right.

the system never freezes on its own.

3. **Runs the three gates** (Figure 4). *Start* triggers Gate 1 (aggregated DAG, sample count, support—or a plain-language reason it is blocked), then the Judge-LLM verdict, then the armed human *Freeze/Reject* buttons.
4. **Freezes and rolls back.** Approving writes a new immutable version with lineage; a second freeze yields v_2 and *rollback* restores v_1 , all visible in the frozen-workflow store.

A toggle switches between the single- and multi-agent systems, each behind one front-end, so the same loop runs on both back to back. In a representative session a well-sampled use case (pubmed_review; 10 runs, 100% support) passes Gate 1 and freezes after approval, while an under-sampled one is correctly blocked before the judge

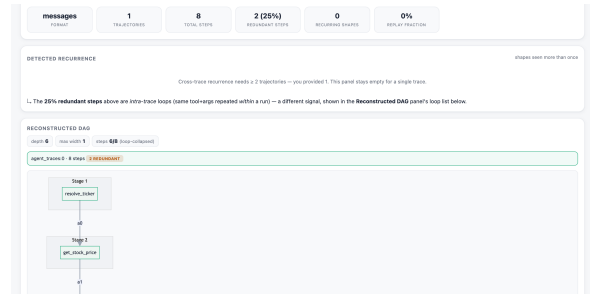


Figure 5: **Bring-your-own-trace tab.** A pasted trace is auto-detected (messages), summarized (8 steps, 25% intra-run redundant), and reduced to a canonical value-flow DAG with loop-collapse applied (8 $\rightarrow$ 6 steps). The recurrence panel correctly reports it needs ≥ 2 trajectories for a single pasted trace.

ever runs.

Bring your own trace. A third tab (Figure 5) turns the same reconstruction on *external* traces: the user pastes or links an agent trace (OpenAI/Anthropic messages, OpenTelemetry spans, Weave/HAL, ToolBench, or the unified schema), and TraceForge auto-detects the format, flags redundant loops, mines cross-run recurrence, and renders the loop-collapsed DAG—no instrumentation of their agent required.

8 Validation

Beyond the live demo, the same pipeline is validated offline (deterministic replay, no external service or model call); details and tables are in App. A–B.

It works at volume. A controlled 2×2 study (system type $\times$ hop depth, 20 runs/cell, 15% deviating) is a *mechanics* check: support tracks the injected deviation rate in every cell (80% at 15% deviation, so the figure restates the construction rather than discovering it), and the value of the cell is that the reconstructor recovers the correct agent count, delegation depth, and a width-2 parallel branch in the multi-agent/multi-hop cell under both the linear and hierarchical signatures (App. A). Streaming 300 synthetic requests through the router serves $66.3 \pm 4.2\%$ of traffic from frozen DAGs once shapes stabilize (5 seeds, range 61–73%), graduating only equivalence-verified shapes.

Frozen workflows match the dynamic agent, and cost less. The gates above decide *whether* to freeze; the question a deployment asks is what happens *after*. We therefore compare the two execution paths directly on requests neither was built

Measure	Dynamic	Frozen DAG
Planner-LLM calls / turn	9	0
Latency (ms)	52.8 ± 2.1	38.0 ± 1.0
Speed-up	—	$1.39 \times \pm 0.05$
<i>Correctness and safety on held-out requests</i>		
Outputs identical	25/25	
Divergences	0	
Fallbacks to dynamic	0	
<i>Traffic served from frozen DAGs (5 seeds $\times$ 300 requests)</i>		
Coverage	$66.3\% \pm 4.2$ (range 61–73%)	

Table 1: Frozen DAG vs. the dynamic agent on *held-out* requests (5 seeds $\times$ 300 requests; 23 graduations, all equivalence-verified). Outputs are identical on every held-out pair, so the efficiency gain is not bought with behavioural drift.

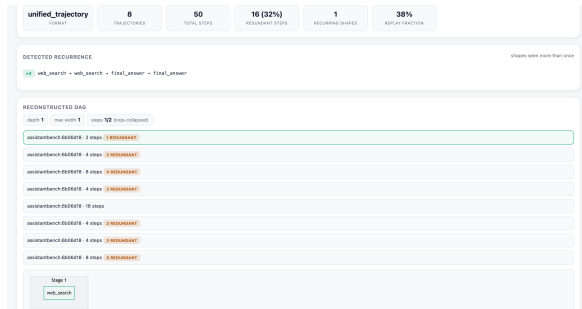


Figure 6: **Real HuggingFace traces, analyzed live.** The bring-your-own-trace tab on 8 HAL trajectories for one ASSISTANTBENCH task solved by different models: TraceForge auto-detects the format, flags 32% intra-run redundant steps (per-trajectory REDUNDANT badges), and mines a recurring shape four models converge to (two web_search steps followed by two final_answer steps; 38% replay fraction)—exactly the recurrence a frozen DAG would serve without a planner call.

from (Table 1). For every shape the router graduates, we hold out five fresh requests and run each through both the dynamic agent and the frozen DAG. Across 5 seeds $\times$ 300 requests, all 25 held-out pairs produce *identical* outputs, with no divergence and no fallback to the dynamic path, while the frozen DAG removes all 9 planner-LLM calls per turn and returns $1.39\times$ faster. The latency figure is a lower bound on the real saving: our reference runtime models tool latency with bounded I/O waits and charges nothing for the planner calls it eliminates, whereas a live planner call against even an on-premise model dominates the tool time it schedules.

The premise holds on real traces. On **610 real trajectories** from four public HuggingFace datasets (HAL (Stroebel et al., 2025), TOOLBENCH (Qin et al., 2023), AGENT-TRACES (trace-commons, 2024), an MCP benchmark (obaydata, 2024)), all 515 analyzable trajectories (≥ 2 tool

steps) reconstruct into DAGs. That conversion rate is uninformative on its own—absent dependency evidence the reconstruction falls back to a chain, and a chain is a DAG—so we report what it recovers: 47.2% of trajectories carry positive value-flow evidence (TOOLBENCH 100%, HAL 0%, which is why mean parallelism is only $1.02\times$). The payoff on un-annotated public traces is therefore replay and redundancy elimination, not parallelism: recurrence mining finds 44.9% of runs in a recurring shape (36.9% non-first occurrences a frozen DAG could serve without a planner call), and 19.1% of steps are value-level redundant. The bring-your-own-trace tab surfaces exactly these signals on any pasted corpus (Figure 6). A labelled golden set (134 trajectories, 26 task-shapes across finance, technical-incident, medical, and two MCP agent families—110 of the 134 non-medical) plants known recurrence and redundancy. Loop detection recovers the planted patterns exactly ($F1 = 1.0$); since the generator plants the three pattern families the detector is built to find and introduces no others, we read that row as verifying the detector against its specification rather than as a discovery result. The discovery result is the clustering: loop-collapse lifts exact-signature F1 from 0.62 to 0.81 at precision 0.97 (App. B). A learned trajectory encoder and an early-graduation classifier are evaluated separately (App. C).

9 Related Work

Reasoning-and-acting agents (Yao et al., 2023b; Schick et al., 2023; Wei et al., 2022), tree/graph successors (Yao et al., 2023a; Besta et al., 2024), tool planners (Shen et al., 2023; Patil et al., 2023), and multi-agent orchestrators (Wu et al., 2023) give the dynamic behavior we observe; graph run-times (LangChain, 2024) and pipeline compilers (Khattab et al., 2024) give deterministic targets but need the workflow hand-specified. *Process mining* (van der Aalst, 2016; van der Aalst et al., 2004) induces a process model from execution logs, and Gate 1’s modal-trajectory clustering is deliberately the simplest such inducer. Our contribution is not a better miner—we do not benchmark against classical inducers, and expect one would recover comparable canonical trajectories—but the layer around it: GenAI-span topology reconstruction, and promotion gated by support, equivalence, and a human. Unlike skill libraries (Wang et al., 2023) and workflow-memory induc-

tion (Wang et al., 2024b), which reuse experience, TraceForge induces workflows from traces (OpenTelemetry Authors, Cloud Native Computing Foundation, 2024) already emitted (zero instrumentation), gating promotion behind a statistical threshold, an advisory judge (Zheng et al., 2023; Shinn et al., 2023), and a human freeze into an *immutable, versioned* store. A skill or sub-agent is a *capability primitive*: it saves re-implementing a task, but the orchestrator LLM still re-plans which primitives to call and how to chain them every turn—exactly the non-deterministic, unauditable cost we remove by freezing the recurring *composition* one layer up (App. D). TraceForge thus complements a skill library rather than replacing it. Closest to us, concurrent work mines clusters of noisy traces into mostly deterministic workflows (El Yadouni and Li, 2026) and distils recurring trace structure into reusable procedural skills (Guo et al., 2026); El Yadouni and Li (2026) likewise admit a dependency only when a consumer argument carries a value attributable to an earlier producer, the same value-flow discipline we apply in §3. TraceForge differs less in the mining than in what surrounds it: capture is zero-instrumentation from OpenTelemetry spans rather than a curated trace dump, promotion is gated by behavioural equivalence and an explicit human freeze, and every frozen version lands in an immutable store with lineage and rollback. El Yadouni and Li (2026) measure call reduction but claim no net efficiency result; Table 1 reports a held-out one. Surveys (Wang et al., 2024a) flag reliability as open. TraceForge addresses this by governing *control flow* across runs rather than attributing individual answers—a different layer of the trust stack.

10 Conclusion

TraceForge hardens dynamic tool-using agents into immutable workflow DAGs by observing the OpenTelemetry traces they already emit, driven end-to-end from a browser console. One topology-agnostic pipeline handles single- and multi-agent systems, and a statistical gate, a Judge-LLM, and a human freeze gate write to an immutable, versioned store—so teams keep agent flexibility for novel tasks while gaining determinism, auditability, and safety for recurring ones. It is a hardening *layer over* a system’s capability primitives, not a replacement: the dynamic agent still serves novel requests, and only recurring *compositions* are frozen. The

payoff accrues where it is needed most—regulated, latency-sensitive deployments with stable, recurring request shapes.

Limitations

“Human-in-the-loop” refers to the operator’s freeze-gate decision in the running system, not a user study. The 2×2 verification and the routing simulation use fixed, constructed parameters, so they are mechanics checks: the 2×2 support figure restates the injected deviation rate, and the routing coverage is a function of the request mixture we chose (we therefore report it over 5 seeds, and the held-out equivalence and efficiency results in Table 1 are the load-bearing claims). The golden set’s loop-detection row verifies the detector against its own specification, not against independent labels; scoring it on human-annotated public traces is future work. Parallel-branch recovery is exercised on instrumented traces and in the router (where behavioural equivalence confirms the parallelization was safe), but no public corpus carries dependency ground truth, so on un-annotated traces it remains a heuristic lower bound. The early-graduation classifier (App. C) predicts cluster membership from a 3-step prefix, but for 32.2% of the corpus that prefix *is* the whole trajectory (TOOLBENCH 56%), so its ROC-AUC is optimistic and a length-stratified rerun is needed before the number is deployed as a routing signal. The request-keyed retrieval ceiling in Table 7 is a benchmark artifact—same-task runs share a byte-identical request string—so it should not be read as evidence that request text is a reliable key under paraphrase. The two live demonstration systems are clinical, but nothing in the pipeline is domain-specific and the evaluation already reaches well beyond that domain: the golden set spans finance, technical-incident, medical, and two MCP agent families (110 of 134 trajectories non-clinical), and the public corpora are web- and API-tool trajectories. Adopting a new domain therefore changes nothing in reconstruction, the gates, or the store—it requires only that the agent emit traces. Both demonstration systems are CPU-only; larger and GPU-backed deployments are untested.

Ethical Considerations

TraceForge is designed for privacy-constrained deployments: capture, reconstruction, gating, judging, and the specification store all run on-premise

with no external service or model call, so traces—which may carry clinical or financial content—never leave the operator’s boundary. The captured traces are the audit evidence and inherit the sensitivity of the underlying system; operators should apply their existing retention and access controls to the run-state store, which we do not attempt to replace. Freezing is deliberately never automatic in the console: a workflow becomes live only on an explicit human decision, recorded with the evidence run IDs and judge verdict that justified it, so accountability rests with a named operator rather than with the system. The rigidity that makes a frozen workflow auditable is also a hazard—a frozen DAG will keep executing a routine that has silently become inappropriate—which is why exactly one version is live, rollback is one action, and sensitive steps (report generation, record creation) stay flagged for human-in-the-loop at runtime. TraceForge governs control flow; it makes no claim about the correctness of the underlying agent’s answers, and must not be read as a clinical-safety mechanism.

References

- Sami Abuzakuk, Anne-Marie Kermarrec, Rishi Sharma, Rasmus Moorits Veski, and Martijn de Vos. 2026. [Optimizing agentic workflows using meta-tools](#). *arXiv preprint arXiv:2601.22037*.
- Anthropic. 2024. Introducing the Model Context Protocol. <https://www.anthropic.com/news/model-context-protocol>. Accessed 2026.
- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoeffler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.
- Ying Chang, Jiahao Xu, Xuan Feng, Chenyuan Yang, Peng Cheng, and Yuqing Yang. 2026. [From noisy traces to root causes: Structural trajectory analysis and causal extraction for agent optimization](#). *arXiv preprint arXiv:2607.07702*.
- Salma El Yadouni and Guanyi Li. 2026. [TraceCompiler: Skill-guided mining and compilation of LLM agent traces into mostly deterministic workflows](#). *arXiv preprint arXiv:2608.02680*.
- Google. 2025. Agent development kit (ADK): An open-source framework for building AI agents. <https://github.com/google/adk-python>. Accessed 2026.
- Zhongxin Guo, Danrui Qi, Hanwen Gu, Peng Cheng, and Yongqiang Xiong. 2026. [SkillDisCo: Distilling and compiling agent traces into reusable procedural skills](#). *arXiv preprint arXiv:2606.26669*.
- Minyang Hu, Bo Yang, Zhihao Zhou, Jiachen Liang, Guo Jiahao, Yiyang Yin, and Xiongwei Han. 2026. [Redundant or necessary? a benchmark for detecting redundant steps in agent trajectories](#). *arXiv preprint arXiv:2605.29893*.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. [DSPy: Compiling declarative language model calls into state-of-the-art pipelines](#). In *The Twelfth International Conference on Learning Representations (ICLR)*.
- LangChain. 2024. LangGraph: Building stateful, multi-actor applications with LLMs. <https://langchain-ai.github.io/langgraph/>. Accessed 2026.
- obaydata. 2024. MCP agent trajectory benchmark. <https://huggingface.co/datasets/obaydata/mcp-agent-trajectory-benchmark>. Public MCP agent trajectory dataset.
- OpenTelemetry Authors, Cloud Native Computing Foundation. 2024. OpenTelemetry: Semantic conventions for generative AI agents. <https://opentelemetry.io/docs/specs/semconv/gen-ai/>. Accessed 2026.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. [Gorilla: Large language model connected with massive APIs](#). *Preprint*, arXiv:2305.15334.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. [ToolLLM: Facilitating large language models to master 16000+ real-world apis](#). *Preprint*, arXiv:2307.16789.
- Nils Reimers and Iryna Gurevych. 2019. [Sentence-BERT: Sentence embeddings using Siamese BERT-networks](#). In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 68539–68551.
- Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugging-GPT: Solving AI tasks with ChatGPT and its friends

- in Hugging Face. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 8634–8652.
- Benedikt Stroebel, Sayash Kapoor, and Arvind Narayanan. 2025. The holistic agent leaderboard (hal): Standardized and cost-aware evaluation of agents. https://huggingface.co/datasets/agent-evals/hal_traces. Public agent trace dataset (Weave/OpenTelemetry spans).
- trace-commons. 2024. agent-traces: OpenAI/Anthropic tool-use agent sessions. <https://huggingface.co/datasets/trace-commons/agent-traces>. Public agent trace dataset (messages + tool_use).
- Wil M. P. van der Aalst. 2016. *Process Mining: Data Science in Action*, 2nd edition. Springer.
- Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *Preprint*, arXiv:2305.16291.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.
- Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *Preprint*, arXiv:2409.07429.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. 2023. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations (ICLR)*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:46595–46623.

A Controlled 2×2 Study

Per cell we report *support* (fraction of runs on the modal trajectory), *determinism* (1—normalized trajectory entropy), the recovered structure (agents, delegation depth, parallel width), *DAG nodes* and span *compression* (mean spans per node), *planner calls* per turn eliminated by freezing, and the Gate 1 and judge outcomes. All numbers are deterministic given the stated configuration (20 runs/cell, 15% deviation, fixed seed).

Cond.	System	Hops	Runs	Supp.	Distinct	Determ.	Agents	Depth
SA-SH	single	single	20	80%	4	0.76	1	1
SA-MH	single	multi	20	80%	4	0.76	1	1
MA-SH	multi	single	20	80%	5	0.74	3	2
MA-MH	multi	multi	20	80%	5	0.74	5	2

Table 2: Recovered structure across the 2×2 study. Support is 80% in every cell because 15% of runs are constructed to deviate—the informative columns are the recovered ones: the multi-agent cells recover the orchestrator plus specialists and a two-level delegation depth.

Cond.	DAG nodes	Max. width	Planner/turn	Compress.	Gate 1	Judge
SA-SH	3	1	4.0 → 0	2.7×	pass	approve (0.88)
SA-MH	6	1	7.0 → 0	2.3×	pass	approve (0.88)
MA-SH	4	1	8.0 → 0	3.8×	pass	approve (0.88)
MA-MH	8	2	14.0 → 0	3.4×	pass	approve (0.88)

Table 3: Efficiency gained and governance outcome. Planner-LLM calls per turn (eliminated under the frozen DAG) grow with topology, from 4 (single-agent, single-hop) to 14 (multi-agent, multi-hop); *Max. width* is the maximum number of agents/tools running concurrently (e.g. width 2 means two specialist sub-agents execute in parallel). The MA-MH cell recovers a width-2 parallel branch. Gate 1 passes in every cell; the offline judge’s 0.88 is $0.4 + 0.6s$ at $s = 0.8$ and so restates the support column rather than voting independently (§4).

Metric	Exact	Semantic
Recurring shapes (support ≥ 2)	18	41
Coverage (% runs in a shape)	20.8	44.9
Replay fraction (already solved)	17.3	36.9
<i>Intra-run redundant work (value-level)</i>		
overall		19.1%
hal		41.9%
toolbench		3.1%
agent_traces		5.7%
<i>DAG conversion</i>		
steps kept after loop-collapse		79.6%
mean parallelism (value-flow)		1.02×

Table 4: Recurrence and DAG conversion on the 515 analyzable trajectories (of 610 assembled). *Exact* clusters identical tool sequences; *semantic* clusters by normalized tool-sequence edit distance (≤ 0.25). *Replay fraction* is the share of runs that repeat a shape already seen. Redundant intra-run work is measured at the *value level* (same tool and arguments).

Dataset	Ingested	Analyzable	→ DAG	Keep
HAL	245	245	245	60.4%
ToolBench	300	243	243	97.1%
agent-traces	27	27	27	95.7%
MCP	38	0	0	—
Overall	610	515	515	79.6%

Table 5: Trace→DAG conversion on the public corpus (§8). *Analyzable* trajectories (≥ 2 tool steps) reconstruct into a value-flow DAG; *Keep* is the fraction of steps surviving loop-collapse. Every analyzable trajectory converts (515/515).

Task	Precision	Recall	F1
<i>Repetitive-pattern (loop) detection, vs. injected labels</i>			
step level	1.000	1.000	1.000
trajectory level	1.000	1.000	1.000
<i>Recurrence vs. gold task-shapes (pairwise)</i>			
exact signature	0.962	0.457	0.620
+ loop-collapse	0.970	0.689	0.806
edit + collapse	0.337	0.850	0.483

Table 6: Golden-data validation on 134 labelled trajectories (26 task-shapes; §8). The loop rows are a specification check—the generator plants the three families the detector implements, so 1.000 is expected by construction (App. B). The informative result is recurrence against gold task-shapes: collapsing loops lifts exact-signature F1 $0.62 \rightarrow 0.81$ (ARI 0.80, NMI 0.94) at unchanged precision.

B Golden-Data Construction

The golden set (§8, Table 6) supplies the ground-truth labels the public corpora lack. It is generated deterministically (stable hashing, no RNG, no LLM services), so the ground truth is exact and the evaluation is reproducible offline.

Agent arms. Five arms are modelled on the operated/replayed MCP agents in our trace reports—SEC-finance, technical-incident, medical-CDS, MCPHunt, and Snorkel-finance—each an evidence-gathering → orchestrator-LLM → scoring pipeline in which every shape maps to a distinct canonical tool sequence. This gives 26 task-shapes and 134 trajectories (finance 30, technical 50, medical 24, MCPHunt 15, Snorkel 15), i.e. 110 of 134 outside the clinical domain of the two demonstration systems.

Planted signals. (i) *Recurrence*: runs of the same task by different models share a gold shape; 15% of runs receive a structure-altering deviation (a dropped or swapped step) that never introduces a repeat. (ii) *Repetition*: 30% of runs receive one injected repetitive pattern—contiguous two-step thrash, a redundant identical call issued three times,

or an explicit `give_up_and_restart`—with the exact wasted-step indices recorded. Distinct calls to the same tool carry distinct target arguments, so only genuinely repeated work shares a value key.

Scoring and provenance. Loop detection is scored against the injected labels (step and trajectory level). Because the generator plants exactly the three pattern families the detector implements—contiguous thrash, a thrice-issued identical call, an explicit restart—and gives every other step a distinct value key, this row is a specification check whose perfect score is expected by construction, not a measurement of detection difficulty; recurrence scoring below is the informative result. Recurrence is scored by pairwise P/R/F1 and ARI/NMI against gold shapes, raw and loop-collapsed. Because the corpus is built by stable hashing from the parameters stated above, Tables 5–6 are fully determined by them and reconstructible from this description alone. Private source traces are not redistributed: the golden set reconstructs documented *architectures*, so it is self-contained.

C Learned Components: Recurrence Matching and Graduation

Recurrence matching (expanding §8) uses the HAL subset, whose `task_id` gives ground-truth identity, with the train/test split by task; the graduation study uses the full 515-trajectory corpus. Numbers average three seeds.

Recurrence matching (Table 7). Does a learned trajectory encoder beat Gate 1’s structural signatures? Against exact-signature, edit-distance, TF-IDF, and (fine-tuned) sentence-transformer (Reimers and Gurevych, 2019) baselines, two findings. *Tool structure alone* collapses every method to $\text{MAP} \approx 0.12$: the same task solved by different models yields unrelated sequences, so learning cannot manufacture an absent signal—this is the robust result, and it is why Gate 1 keys on a use-case identifier rather than on structure. Request text scores $\text{MAP} 0.999$, but on this benchmark same-task requests are byte-identical, so that column measures string equality through an encoder and cannot license a recommendation to key production recurrence on request text, where operators paraphrase. Testing that requires a paraphrased-request evaluation we have not run.

Early graduation. A length-3-prefix classifier (tools-only prefix embedding plus cheap structural

Method	Request+tools	Tools only
Exact signature	0.121 ± 0.001	0.121 ± 0.001
Edit distance	0.124 ± 0.003	0.124 ± 0.003
TF-IDF	0.875 ± 0.021	0.123 ± 0.005
Encoder (pretrained)	0.999 ± 0.002	0.122 ± 0.002
Encoder (fine-tuned)	1.000 ± 0.000	0.125 ± 0.004

Table 7: Same-task retrieval (mean average precision; HAL, held-out tasks, 3 seeds). Tool structure alone is a near-useless key ($\text{MAP} \approx 0.12$ for every method). The request-text column is a *ceiling artifact*—same-task HAL runs share a byte-identical request string—so it bounds what is achievable on this benchmark, not under paraphrase.

	Skill library	Frozen DAG
Unit	routine per task	mined composition
Source	authored	observed traces
Planning/turn	LLM re-plans	none (frozen)
Determinism	no	yes
Parallelism	LLM order	value-flow DAG
Lineage	git (intent)	immutable, signed
Adaptivity	graceful	rigid; re-freeze
Best when	variable traffic	stable, audited

Table 8: Skill libraries and TraceForge address different layers, so they compose. Parallelism is exact only on instrumented traces; on public traces it is a lower bound (§8).

counts; positive rate 54%) predicts recurring-shape membership at ROC-AUC 0.990 ± 0.008 (logistic regression) and 0.990 ± 0.010 (MLP), average precision 0.993 ± 0.004 and 0.994 ± 0.006 respectively, suggesting a router could flag freeze candidates mid-run. This figure needs one qualification: the label is membership in a cluster defined over the *whole* tool sequence, and for 166/515 trajectories (32.2%; TOOLBENCH 56.0%, HAL 11.8%) a 3-step prefix *is* that sequence, so those rows leak their label; only 37.5% of the corpus exceeds 6 steps. We therefore report a promising offline signal requiring a length-stratified rerun before use; the live router still aggregates whole trajectories (Gate 1).

D Positioning vs. Skill Libraries

A recurring objection: if a *skill library* (Wang et al., 2023, 2024b) already stores a routine per task, why freeze DAGs? Table 8 contrasts the two. They are complementary: a skill is a capability primitive whose *invocation* the orchestrator still re-plans each turn, whereas the frozen DAG removes that planner call.