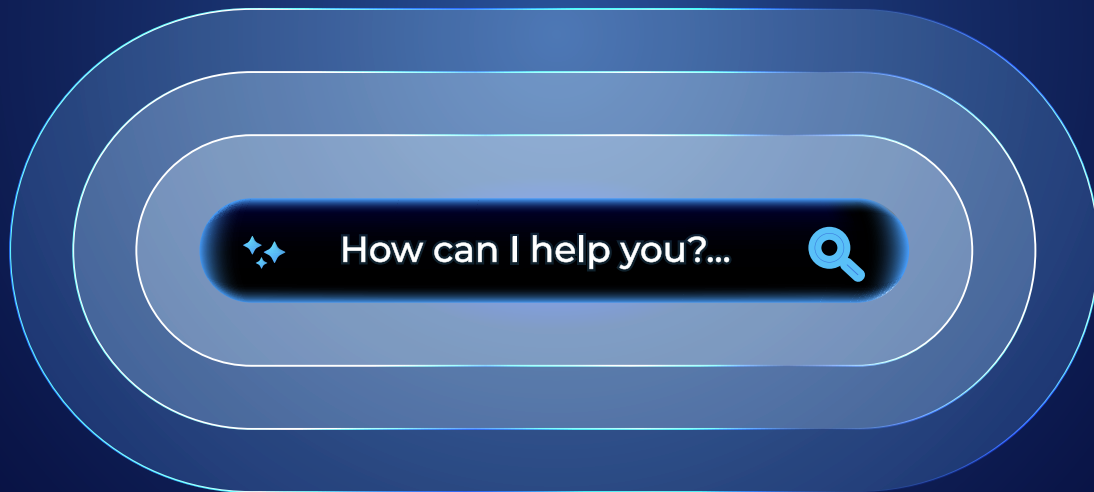


Harness State Engineering for Reliable Agentic AI Systems

A Runtime-State-Centric Survey

 Research paper

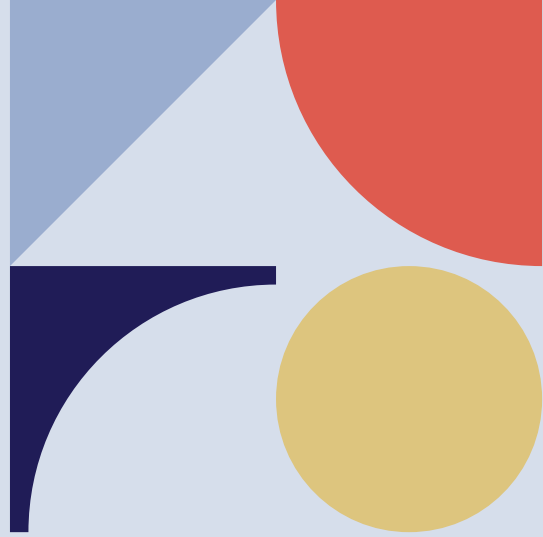




Large language model (LLM) agents increasingly operate in long-horizon, tool-using, and multi-step environments where reliability depends not only on the underlying model but also on the execution harness surrounding it. Recent harness surveys have established the harness as an independent systems layer, while work on context engineering has clarified how information should be assembled for model inference. However, existing harness literature remains primarily architecture-, tooling-, or trajectory-safety-oriented. This survey proposes Harness State Engineering as a complementary runtime-state-centric perspective for studying reliable agentic systems. Whereas context engineering focuses on what information the model should

see, Harness State Engineering focuses on what the runtime must track, control, validate, recover, and audit during execution. We synthesize the literature through seven state dimensions: execution, tool, memory/context, verification, observability, governance, and recovery state. We further propose a harness-state lifecycle, a maturity model, and an evaluation protocol that connect these dimensions to practical assessment. The survey is positioned as a unifying framework for building enterprise-grade agentic AI systems that require traceability, controlled failure, safe tool use, and auditable evolution.

Additional KeyWords and Phrases: agentic AI; LLM agents; harness engineering; runtime state; observability; verification; governance; recovery; context engineering; AI safety



Introduction

The recent shift from single-turn prompting toward persistent, tool-using, and multi-agent execution has changed where the main engineering bottleneck lies in AI systems. Early LLM applications primarily depended on prompt engineering: the design of instructions, examples, and formatting constraints for a single model call. Context engineering extended this view by treating the full inference-time information payload as an object of engineering, including retrieval, memory, compression, tool context, and conversation state. As agentic systems increasingly plan, use tools, call APIs, modify files, coordinate sub-agents, and continue over many turns, reliability depends on more than prompts or context alone.

Harness engineering has emerged to describe the infrastructure surrounding an agent: execution environments, tool registries, context managers, state stores, lifecycle hooks, observability systems, verification interfaces, and governance controls. Recent work has made major progress in formalizing this layer and showing that harness design can influence agent reliability even when the underlying model is unchanged. However, a key concern remains underdeveloped: the runtime state that the harness must maintain to make agent behavior controllable, recoverable, and auditable.

This survey proposes Harness State Engineering as a runtime-state-centric lens for reliable agentic AI systems. If

context engineering asks what information the model should see, Harness State Engineering asks what the runtime must know, store, update, validate, recover, and audit while the agent acts. This framing is especially important in enterprise and regulated settings, where failures are not limited to incorrect final answers. A system may complete a task while bypassing a permission boundary, losing provenance, failing to record validation outcomes, or producing an action sequence that cannot be reconstructed after the fact.

Our goal is not to claim that harness engineering itself is new. Existing surveys already study harness architectures, while recent empirical work examines automatic harness evolution and trajectory-level safety auditing. Instead, this survey contributes a state-centric synthesis: it organizes harness reliability around the runtime state needed for execution control, safe tool use, memory continuity, verification, observability, governance, and recovery.

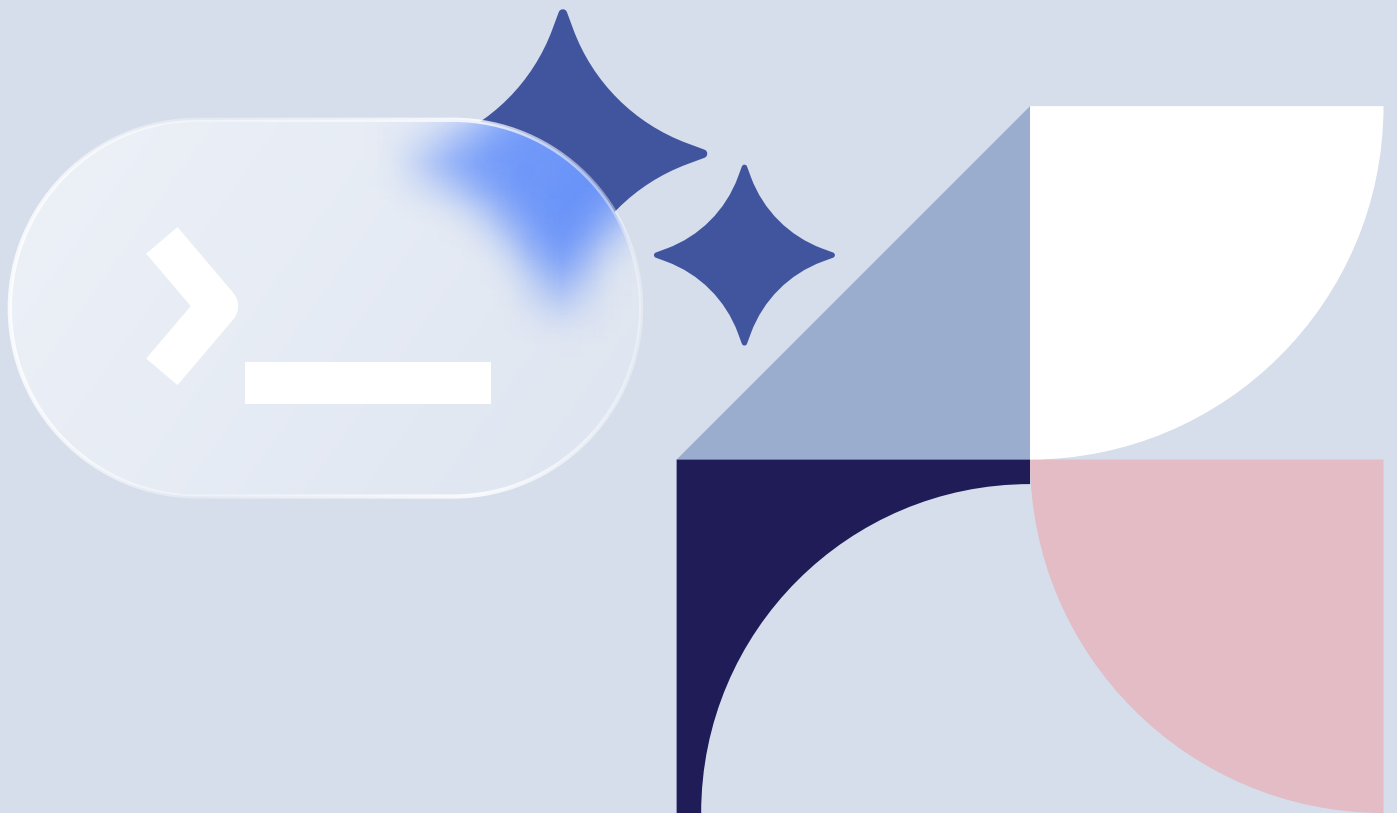
Scope, Methodology, and Research Questions

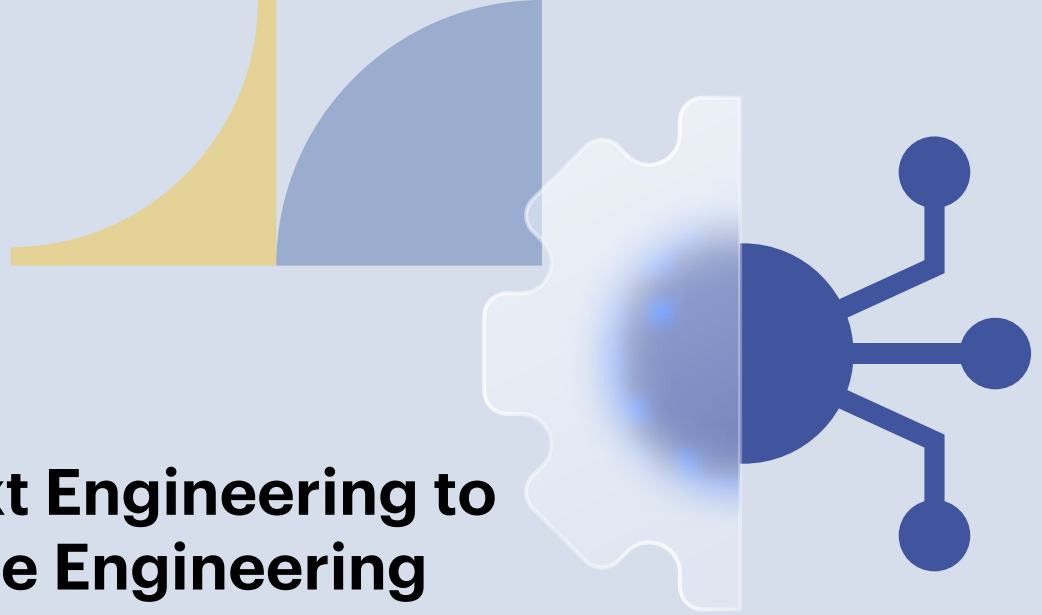
This survey is structured as a narrative-systematic review. The scope covers LLM-based agentic systems that perform multi-step execution, interact with tools or environments, or require durable state across turns. We exclude work that only studies single-turn prompting without external action, purely model-training methods without an execution harness, and application papers that do not expose reusable harness, state, evaluation, or governance concepts.

The literature was organized using seed papers on context engineering, agent harness surveys, observability-driven harness evolution, and trajectory-level harness safety. These were expanded with related work on LLM agents, tool use, agent benchmarks, memory systems, multi-agent orchestration, RAG, verification, observability, and governance. Papers were grouped by the primary runtime concern they address: execution, tools, memory/context, verification, observability, governance, recovery, or evaluation.

The review is guided by five research questions. RQ1: What runtime state must an agent harness maintain for reliable long-horizon execution? RQ2: How does Harness State Engineering differ from, and complement, context engineering and general harness engineering? RQ3: How do existing agent frameworks, benchmarks, and safety studies implicitly manage runtime state? RQ4: Which evaluation metrics are needed to assess harness-state completeness, consistency, recoverability, auditability, and compliance? RQ5: What open research problems remain for standardizing and governing runtime state in enterprise-grade agentic AI systems?

This methodology is designed to avoid a chronological catalog of papers. Instead, the survey develops an original organizing framework: a seven-state taxonomy, a lifecycle model, a maturity model, and an evaluation protocol. This organization aligns with the expectations of survey venues that require synthesis, classification, and perspective rather than paper-by-paper summaries.





From Context Engineering to Harness State Engineering

Context engineering focuses on constructing the information payload available to the model at inference time. It includes selecting documents, retrieving relevant memory, compressing conversation history, ranking tool results, and ensuring that the model sees the most useful information under context-window constraints. This view is essential because LLM outputs are conditioned on the information they receive.

Harness engineering shifts attention from model input to runtime infrastructure. Agents do not merely answer; they act. They call tools, write files, execute code, query external systems, communicate with other agents, and continue across multiple steps. The

harness defines what actions are exposed, how tool calls are routed, how failures are handled, how traces are collected, and how governance is enforced.

Harness State Engineering sits inside this broader harness layer but treats state as the primary object of analysis. A harness can contain tools and logs without having a coherent state model. For example, a system may log a tool call but fail to record whether the call was authorized, which validation checks preceded it, whether the result was used in a final decision, or whether rollback remains possible. A state-centric view therefore asks not just what components exist, but what runtime facts must be explicit for safe continuation and auditability.

Table 1. Positioning Harness State Engineering against related work.

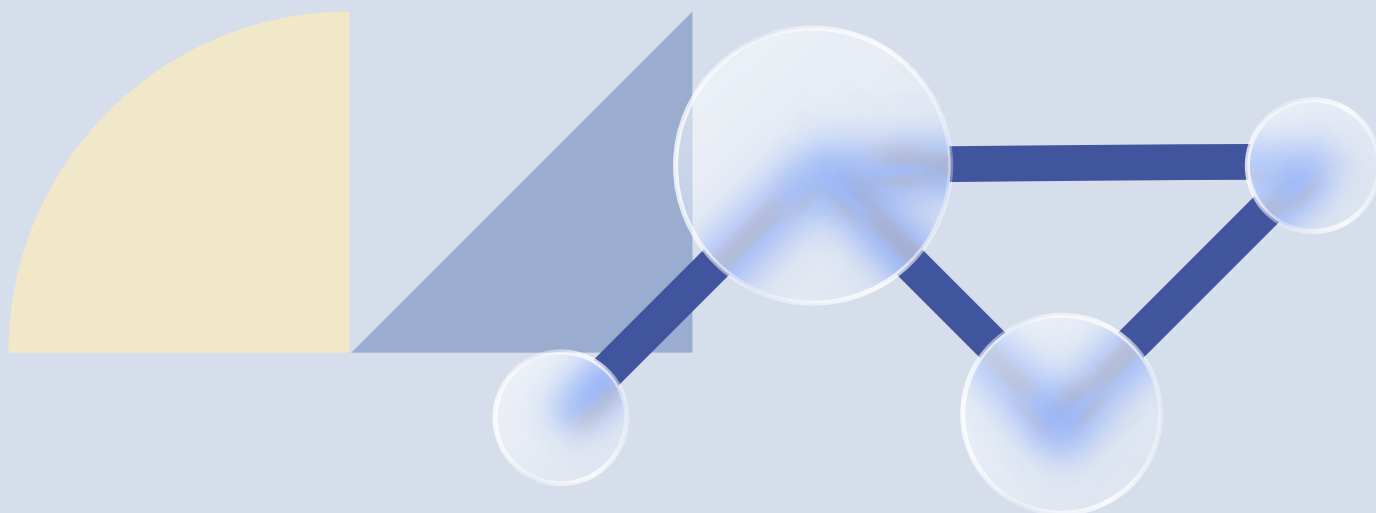
Literature Cluster	Primary Focus	Representative Contribution	Remaining Gap
Context Engineering	Inference-time information payload	How to select, manage, and compress information for LLMs	Does not fully address runtime control, recovery, or audit
Harness Surveys	Agent infrastructure and components	Execution, tools, lifecycle, observability, verification, governance	Often architecture-centric rather than state-centric
AHE	Automatic harness evolution	Observability-driven improvement of coding-agent harnesses	Focused on coding agents and evolution loops
HarnessAudit	HarnessAudit Trajectory-level safety auditing	Permission boundaries, execution fidelity, stability	Safety-focused, not a full state taxonomy
This Survey	Runtime state engineering	State taxonomy, lifecycle, maturity model, evaluation protocol	Requires future empirical benchmarking

Related Work Landscape

Existing work can be organized into five clusters. First, agent architecture and harness surveys formalize the execution layer around LLM agents. These works establish that reliability is shaped by execution loops, tools, context managers, lifecycle hooks, observability, verification, and governance. Second, LLM-agent research studies planning, tool use, memory, reflection, and environment interaction. Third, benchmark and evaluation work such as Agent-Bench, WebArena, SWE-bench, and related environments evaluates agent behavior over tasks and trajectories. Fourth, safety and governance work studies guardrails, policy compliance,

information-flow risks, and human oversight. Fifth, context-engineering and RAG literature studies how external knowledge and memory are selected and presented to the model.

Harness State Engineering is complementary to all five clusters. It does not replace architecture taxonomies, tool-use methods, or safety benchmarks. Instead, it asks how runtime state connects them. Planning requires execution state; tool use requires tool state; memory requires provenance and retention state; guardrails require verification and governance state; debugging requires observability state; and failure handling requires recovery state.



Formalizing Harness State

We define harness state at time t as $Hs(t) = (Et, Tt, Mt, Vt, Ot, Gt, Rt)$, where E is execution state, T is tool state, M is memory/context state, V is verification state, O is observability state, G is governance state, and R is recovery state. The tuple is not intended as a mandatory implementation schema. Rather, it provides a conceptual model for the minimum classes of runtime information required to make an agent inspectable, governable, recoverable, and safe to evolve.

The state tuple separates architecture from state. Two systems may both contain a tool registry and a tracing system, but only one may record sufficient tool authorization, schema version, input provenance, output validation, and rollback state. Similarly, two systems may both use memory, but only one may preserve retrieval provenance, compaction decisions, and sensitive-data filtering outcomes. Harness State Engineering therefore evaluates the quality and purpose of runtime state rather than merely checking component presence.

Seven Dimensions of Harness State

Execution state captures workflow progress: active step, task phase, planning branch, loop counter, termination condition, and checkpoint identity. It is essential for resumption and debugging in long-horizon workflows.

Tool state captures the runtime status of external tools and interfaces: tool availability, permissions, argument schemas, cached outputs, side-effect records, and error states. Tool state prevents the agent from relying on stale or unauthorized action assumptions.

Memory/context state records what information has been retrieved, compressed, retained, discarded, or injected into the model context. It links context engineering to runtime accountability by preserving retrieval provenance and memory update decisions

Verification state records whether intermediate or final actions satisfy constraints. It may include schema validation, tests, policy checks, business-rule checks,

or milestone verification. This state is crucial because final-output success may hide invalid intermediate behavior.

Observability state stores traces, logs, metrics, timing information, cost indicators, and failure classifications. It enables diagnosis, comparison, regression analysis, and evidence-driven harness improvement.

Governance state records permission boundaries, approval status, role bindings, information-flow restrictions, compliance requirements, and audit actors. In regulated domains, governance state is necessary to explain not only what happened, but whether it was allowed to happen.

Recovery state records checkpoints, rollback pointers, retry budgets, fallback paths, escalation triggers, and postfailure annotations. It supports controlled failure, graceful degradation, and safe resumption after interruptions.

Harness State Lifecycle

A state-centric survey should explain not only what state categories exist, but how they are created, checked, retained, and evolved during an agent run. We define the Harness State Engineering lifecycle as a recurring sequence: observe, represent, validate, commit, recover, audit, and evolve.

In the observe phase, the harness records signals from agent actions, tool calls, memory retrievals, validation checks, and governance policies. In the represent phase, these signals are normalized into typed state objects rather than unstructured logs. In the validate phase, the harness checks whether a proposed state transition satisfies execution constraints, tool permissions, schema expectations, and governance requirements. In the commit phase,

accepted transitions are persisted through snapshots, event logs, or hybrid stores. In the recover phase, persisted state supports retry, rollback, resume, or escalation. In the audit phase, state transitions become inspectable to developers, operators, and reviewers. In the evolve phase, accumulated state evidence informs improvements to prompts, tools, middleware, verification rules, or governance policies.

This lifecycle draws a boundary between ordinary observability and Harness State Engineering. Observability asks whether we can inspect what happened. Harness State Engineering asks whether the runtime state is structured enough to support correct continuation, safe intervention, accountable recovery, and governed evolution.

Maturity Model

We propose a five-level maturity model. Level 0, Implicit State, describes systems where runtime behavior is mostly captured as conversation text or raw logs. Level 1, Logged State, records execution and tool calls for post-hoc inspection but offers limited recovery. Level 2, Typed State, explicitly structures execution, tool, memory, and verification states,

enabling partial resumption and structured failure analysis. Level 3, Governed State, makes governance and recovery state first-class objects, supporting approval gates, rollback, escalation, and audit. Level 4, Evolvable State, uses state evidence to drive governed harness improvement with versioning, reversibility, and attribution.

Table 2. Harness State Engineering maturity model.

Level	Name	Capability
0	Implicit State	Runtime behavior is mostly captured as conversation or raw logs.
1	Logged State	Execution and tool calls are recorded for post-hoc inspection.
2	Typed State	Execution, tool, memory, and verification states are structured.
3	Governed State	Governance and recovery states support approval, rollback, escalation, and audit.
4	Evolvable State	State evidence drives governed harness improvement with versioning and reversibility.

Table 3. Evaluation metrics for Harness State Engineering.

Metric	What It Measures	Relevant State Dimension
State Completeness	Required runtime information is captured	Execution, Tool, Memory, Verification
State Consistency	State transitions remain coherent	Execution, Tool, Memory
Recoverability	System can resume, retry, rollback, or escalate	Recovery, Execution
Auditability	Actions and decisions can be reconstructed	Observability, Governance
Governance Compliance	Permissions and approvals are respected	Governance, Tool, Verification

The maturity model reframes harness comparison from a component checklist into a reliability assessment. Instead of asking only whether a framework has tools, memory, or observability, we ask

which runtime states are explicit, which transitions are validated, which failures are recoverable, and which changes are auditable.



Evaluation Protocol and Metrics

A state-centric framework is useful only if harnesses can be compared and evaluated beyond final task success. We propose an evaluation protocol that treats the harness as the unit of evaluation. The benchmark task should specify intermediate milestones, tool boundaries, validation rules, recovery expectations, and governance constraints. During execution, the harness records state transitions across the seven state dimensions. The resulting trajectory is then evaluated at both the outcome level and the state level.

Outcome-level metrics measure task completion and final-output validity. State-level metrics inspect the runtime trajectory. We define five evaluation dimensions: state completeness, state consistency, recoverability, auditability, and governance compliance. State completeness asks whether the harness records the information required for

continuation, validation, and audit. State consistency asks whether state transitions remain coherent across execution, tools, memory, verification, and policy. Recoverability asks whether the system can resume, retry, rollback, or escalate after interruption. Auditability asks whether a reviewer can reconstruct why actions occurred. Governance compliance asks whether actions respect permission boundaries and information-flow constraints across the full trajectory.

The evaluation protocol also aligns with the maturity model. Level 0 and Level 1 systems may support only post-hoc inspection. Level 2 systems enable typed validation and partial resumption. Level 3 systems support governed recovery and compliance. Level 4 systems enable safe harness evolution because every runtime change is backed by versioned and auditable state evidence.

Enterprise and Regulated-Domain Implications

Enterprise deployments introduce requirements that are often absent from research prototypes. In financial services, healthcare, legal operations, and software engineering, agents must operate under access controls, policy constraints, audit requirements, and human approval workflows. A correct final answer is insufficient if the path to that answer involved unauthorized access, missing validation, or unrecoverable side effects.

Harness State Engineering provides a vocabulary for

these operational requirements. Governance state makes policy boundaries explicit. Verification state records compliance checks. Observability state supports incident investigation. Recovery state supports controlled failure and rollback. Memory/context state preserves provenance and sensitive-data handling. Tool state records side effects and authorization. Execution state enables continuity across long-running processes. Together, these states form the runtime evidence layer needed for trustworthy enterprise agents.

Open Research Challenges

Formal semantics for harness state transitions remain underdeveloped. Without standard transition semantics, it is difficult to compare harnesses or prove safety properties across frameworks.

Cross-framework state schemas are needed. Current agent systems vary in how they represent traces, memories, tool calls, policies, and checkpoints. Standardized schemas would improve portability, benchmarking, and auditability.

Scalable recovery remains difficult for long-horizon and multi-agent tasks. Recovery must account not

only for execution state, but also for tool side effects, memory updates, and governance decisions.

Governed self-evolution is an emerging challenge. AHE-like systems show that harnesses can improve through observability-driven evidence, but enterprise settings require versioning, approval, rollback, and accountability for every harness modification.

Privacy-preserving observability is also open. Rich state records may contain sensitive data, tool inputs, user context, or policy details. Future work must preserve auditability without excessive retention or leakage risk.

Threats to Validity and Publication Integrity

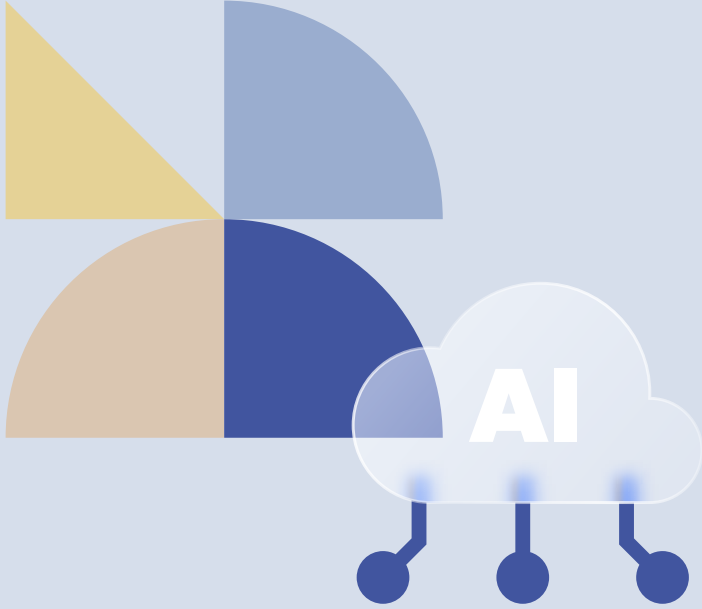
This survey avoids unsupported quantitative claims. Where empirical values are not directly established in the cited literature, we use qualitative language rather than invented percentages or benchmark numbers. This is important for both scientific accuracy and publication integrity.

The proposed taxonomy is interpretive. Other taxonomies could organize the same literature by architecture, risk, lifecycle, or application domain. We mitigate this by explicitly positioning Harness State Engineering as complementary to, rather than a replacement for, existing harness taxonomies.

To reduce plagiarism risk, the manuscript is written in original language and cites source papers for claims about existing work. Direct quotations are avoided

unless necessary, and all definitions introduced by this survey are framed as our synthesis. Before submission, authors should run an institutional plagiarism check and verify every BibTeX entry, citation, and claim.

Regarding AI-assisted writing, ACM policy permits generative AI tools under conditions that the work does not plagiarize, misrepresent, or falsify content, and that authors remain responsible for all material. If AI tools are used for research methodology, data generation, analysis, or artifact creation, that use should be described in the methods section. If used only for writing assistance, authors should still verify all claims, citations, and originality before submission.



Conclusion

Harness engineering has become central to reliable agentic AI systems, but the next conceptual step is to study not only harness architecture, but also the runtime state that makes controlled execution possible. This survey proposes Harness State Engineering as that step. By organizing the field around seven runtime state dimensions and connecting them through a lifecycle, maturity model,

and evaluation protocol, the survey provides a practical framework for reliable, recoverable, auditable, and governable agentic AI systems.

The central takeaway is simple: context engineering optimizes what the model sees; Harness State Engineering optimizes what the runtime must know and manage while the agent acts.

Authors:



Saipriyadarshan Cindula,
Zensar.com



Nikhil Gajanan Bajaj,
Zensar.com

References

2026. Bommasani, R. et al. On the Opportunities and Risks of Foundation Models. arXiv:2108.07258, 2021. Reference placeholder; verify before final submission.

[2] 2026. Chandy, K. M. and Lamport, L. Distributed Snapshots: Determining Global States of Distributed Systems. ACM TOCS, 1985. Reference placeholder; verify before final submission.

[3] 2026. Gao, Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997, 2023. Reference placeholder; verify before final submission.

[4] 2026. Jimenez, C. E. et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR, 2024. Reference placeholder; verify before final submission.

[5] 2026. Lamport, L. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 1978. Reference placeholder; verify before final submission.

[6] 2026. Lewis, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. NeurIPS, 2020. Reference placeholder; verify before final submission.

[7] 2026. Li, G. et al. CAMEL: Communicative Agents for Mind Exploration of Large Language Model Society. NeurIPS, 2023. Reference placeholder; verify before final submission.

[8] 2026. Li, J. et al. Agent Harness Engineering: A Survey. OpenReview / TMLR submission, 2026. Reference placeholder; verify before final submission.

[9] 2026. Lin, J. et al. Agentic Harness Engineering: Observability-Driven Automatic Evolution of Coding-Agent Harnesses. arXiv:2604.25850, 2026. Reference placeholder; verify before final submission.

[10] 2026. Liu, C. et al. Auditing Agent Harness Safety. arXiv:2605.14271, 2026. Reference placeholder; verify before final submission

[11] 2026. Liu, X. et al. AgentBench: Evaluating LLMs as Agents. ICLR, 2024. Reference placeholder; verify before final submission.

[12] 2026. Mei, L. et al. A Survey of Context Engineering for Large Language Models. arXiv:2507.13334, 2025. Reference placeholder; verify before final submission.

[13] 2026. Meng, Q. et al. Agent Harness for Large Language

Model Agents: A Survey. Preprints.org, 2026. Reference placeholder; verify before final submission.

[14] 2026. NIST. Artificial Intelligence Risk Management Framework (AI RMF 1.0). 2023. Reference placeholder; verify before final submission.

[15] 2026. Ouyang, L. et al. Training Language Models to Follow Instructions with Human Feedback. NeurIPS, 2022. Reference placeholder; verify before final submission.

[16] 2026. Park, J. S. et al. Generative Agents: Interactive Simulacra of Human Behavior. UIST, 2023. Reference placeholder; verify before final submission.

[17] 2026. Raji, I. D. et al. Closing the AI Accountability Gap. FAccT, 2020. Reference placeholder; verify before final submission.

[18] 2026. Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. NeurIPS, 2023. Reference placeholder; verify before final submission.

[19] 2026. Shinn, N. et al. Reflexion: Language Agents with Verbal Reinforcement Learning. NeurIPS, 2023. Reference placeholder; verify before final submission.

[20] 2026. Wang, G. et al. Voyager: An Open-Ended Embodied Agent with Large Language Models. TMLR, 2024. Reference placeholder; verify before final submission.

[21] 2026. Weidinger, L. et al. Ethical and Social Risks of Harm from Language Models. FAccT, 2022. Reference placeholder; verify before final submission.

[22] 2026. Wu, Q. et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. arXiv:2308.08155, 2023. Reference placeholder; verify before final submission.

[23] 2026. Yao, S. et al. ReAct: Synergizing Reasoning and Acting in Language Models. ICLR, 2023. Reference placeholder; verify before final submission.

[24] 2026. Yao, S. et al. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. NeurIPS, 2023. Reference placeholder; verify before final submission.

[25] 2026. Zhou, S. et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. ICLR, 2024. Reference placeholder; verify before final submission.



zensar

An  **RPG** Company

At Zensar, we're 'experience-led everything.' We are committed to conceptualizing, designing, engineering, marketing, and managing digital solutions and experiences for over 145 leading enterprises. Using our 3Es of experience, engineering, and engagement, we harness the power of technology, creativity, and insight to deliver impact.

Part of the \$4.8 billion RPG Group, we are headquartered in Pune, India. Our 10,000+ employees work across 30+ locations worldwide, including Milpitas, Seattle, Princeton, Cape Town, London, Zurich, Singapore, and Mexico City.

For more information, please contact: info@zensar.com | www.zensar.com