Harness Engineering for Language Agents: The Harness Layer as Control, Agency, and Runtime

Chaoyue He *, Xin Zhou, Di Wang, Hong Xu, Wei Liu, Chunyan Miao

Posted Date: 23 March 2026

doi: 10.20944/preprints202603.1756.v1

Keywords: harness; harness layer; language agents; harness engineering; prompt engineering; context engineering; control-agency-runtime (CAR); agent evaluation; tool use

 doi:10.20944/preprints202603.1756.v1

Article

Harness Engineering for Language Agents: The Harness Layer as Control, Agency, and Runtime

Chaoyue He [1,*], Xin Zhou [1], Di Wang [1], Hong Xu [1], Wei Liu [2] and Chunyan Miao [1]

[1] Alibaba-NTU Global e-Sustainability CorpLab (ANGEL), Singapore
[2] Alibaba Group, China
* Correspondence: cyhe@ntu.edu.sg

Abstract

Language agents that act through tools, files, browsers, APIs, and persistent sessions are shaped by more than the base model or a single prompt. Their reliability depends on a *harness layer* that determines which instructions remain authoritative, what actions are available, how state is carried forward, and how failures are handled over time. This position paper argues that recent practice has made this layer visible enough to warrant explicit treatment in NLP. We propose and operationalize a working decomposition of the harness layer as **control, agency, and runtime (CAR)**; situate harness engineering in the arc from software engineering through prompt and context engineering; and provide a lightweight audit of **40** harness-relevant works in our selected evidence base, suggesting a visibility gap between academic papers and public engineering notes. We further argue that many reported agent gains may be partly **harness-sensitive** rather than purely model-driven, and propose HARNESSCARD as a lightweight reporting artifact, including a filled example. Grounded in papers, benchmarks, protocols, and engineering notes through **March 20, 2026**, we argue that progress in language agents should report not only the model, but also the harness layer that turns capability into governed action.

Keywords: harness; harness layer; language agents; harness engineering; prompt engineering; context engineering; control-agency-runtime (CAR); agent evaluation; tool use

1. Introduction

Reliable agency is designed, not inferred. Once language models act through tools, files, browsers, APIs, and persistent sessions, reliability depends on a *harness layer*: the extra-model layer that determines which instructions remain binding, what actions are available, how state is externalized, and how multi-step execution is kept bounded, recoverable, and inspectable. We argue that *harness engineering* is a useful name for work on that layer and that language-agent research should study it as an explicit object rather than leave it as hidden implementation residue. Figure 1 previews both the widening of the engineering object and our working CAR decomposition of the harness layer.

 doi:10.20944/preprints202603.1756.v1

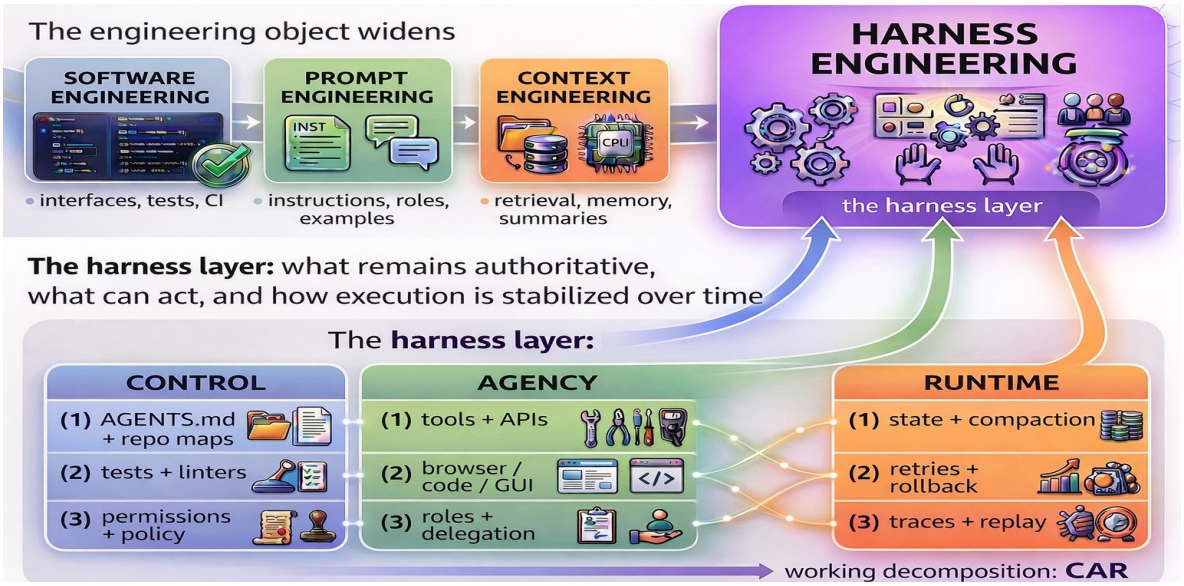

Figure 1. Top: the engineering object widens from software engineering (Naur and Randell 1969) through prompt engineering (Liu et al. 2023), context engineering (Rajasekaran et al. 2025), to harness engineering (Grace et al. 2026; Lopopolo 2026). **Bottom**: our decomposition of the *harness layer* into control, agency, and runtime (CAR).

Recent public engineering notes make the shift unusually visible. Anthropic defines an *agent harness* (or scaffold) as the system that enables a model to act as an agent, and emphasizes that evaluating an "agent" means evaluating model and harness together rather than the model in isolation (Grace et al. 2026). OpenAI uses the broader label *harness engineering* for long-horizon systems whose reliability depends on repository maps, `AGENTS.md`, architectural rules, cleanup loops, and runtime controls rather than on prompt wording alone (Choi 2026; Lopopolo 2026; OpenAI 2026a,b). Anthropic's long-running harness work and research-system notes push the same lesson beyond coding by emphasizing state externalization, progress files, orchestration, and recovery structure in multi-session or multi-agent work (Hadfield et al. 2025; Young 2025). Taken together, these materials motivate the working use in this paper: a *harness* is the extra-model system that couples durable control artifacts, mediated action interfaces, and runtime policies into a governed execution regime.

The underlying design problem, however, is older than the name. Work on tool use, browser agents, software agents, orchestration frameworks, interactive evaluation, runtime protocols, and deployment notes has long studied pieces of the same layer without naming it (Jimenez et al. 2023; Li et al. 2023; Liu et al. 2023; Schick et al. 2023; Wang et al. 2024a; Xie et al. 2024; Yang et al. 2024; Yao et al. 2022; Zhou et al. 2023). Our claim is therefore not that harness-like mechanisms are new. It is that recent practice makes the layer visible enough that the field now benefits from naming, decomposing, and reporting it directly.

This paper is a *position paper*, not a causal ablation study. It draws on a selective evidence base of harness-relevant papers, benchmarks, protocols, and engineering notes through **March 20, 2026**; Appendix A documents that evidence base and its selection logic. The contribution is fourfold. First, we situate harness engineering in the widening arc from software engineering through prompt and context engineering. Second, we propose and operationalize a working definition of the harness layer through a coupled *control–agency–runtime* (CAR) decomposition and use that decomposition to clarify the scientific object. Third, we add a lightweight descriptive audit of the in-scope evidence base to examine which parts of the harness are more visible in the literature and which are more often surfaced only in engineering notes. Fourth, we argue that language-agent progress should be interpreted and reported more *harness-sensitively*, and we propose HARNESSCARD as a lightweight disclosure artifact for that purpose. Our claim is not that scaffolds, runtimes, or orchestration are new, but that the harness layer has become a reportable scientific object in its own right, making claims more comparable, auditable, and reproducible.

 doi:10.20944/preprints202603.1756.v1

Table 1. Public formulations motivating the harness layer as a distinct systems object; fuller version in Appendix A.2.

Public source	Year	Object	Why it matters for the harness layer
Anthropic evals note (Grace et al. 2026)	2026	agent harness / evaluation harness	defines the harness as the system that enables action and makes explicit that agent evaluation measures harness plus model
OpenAI harness note (Lopopolo 2026)	2026	harness engineering	names the broader practice and ties it to project guidance, constraints, cleanup loops, and long-horizon coding work
OpenAI Codex notes (Bolin 2026; Chen 2026)	2026	Codex harness	treats the harness as reusable runtime logic that can power multiple surfaces, not just a one-shot prompt wrapper
Anthropic long-running note (Young 2025)	2025	long-running harnesses	makes state externalization, progress tracking, and recovery first-class engineering levers
OpenAI developer guidance (Choi 2026; OpenAI 2026a,b)	2026	`AGENTS.md`, long-horizon tasks	shows the harness as durable instruction plus executable verification and repair loops
MCP specification (Model Context Protocol 2025)	2025	protocol layer	moves tool interoperability and permission boundaries into an explicit systems interface

2. From Software Engineering to Prompt, Context, & Harness Engineering

Seen historically, harness engineering does not replace earlier forms of engineering around models; it nests them. Software engineering taught the field to care about interfaces, tests, modularity, and operational discipline (Naur and Randell 1969). Prompt engineering narrowed the immediate intervention surface to the design of instructions and examples (Liu et al. 2023). Context engineering widened that surface to the evolving token state an agent carries from step to step, including retrieval, memory, tool context, and history (Rajasekaran et al. 2025). Harness engineering widens it again: not only what enters context, but also how action is mediated, how runtime is controlled, how failures are repaired, and how governance and traces are maintained. Prompt design therefore remains inside control, context design spans control plus runtime, and harness engineering names the coupled layer that makes agency durable over time. Table 2 summarizes this widening by engineering question, typical artifacts, and what each frame still leaves under-described.

That widening matters because a system can improve even when the base model and prompt change little, simply because the system of record, retry policy, action surface, verifier loop, or recovery structure changed. Once agents act over time, the scientifically interesting question is no longer only what the model was told, but also which harness layer helped keep its behavior reliable.

2.1. The Label Is New, the Design Problem Is Not

The phrase *harness engineering* appears only recently in public AI discourse, but the underlying design problem has been converging for years. ReAct made reasoning-and-acting trajectories explicit (Yao et al. 2022). Toolformer and API-Bank made tool calls and evaluation concrete objects (Li et al. 2023; Schick et al. 2023). AgentBench, WebArena, GAIA, SWE-bench, BrowseComp, and OSWorld moved interactive action, not static text generation, toward the center of evaluation (Jimenez et al. 2023; Liu et al. 2023; Mialon et al. 2023; Wei et al. 2025; Xie et al. 2024; Zhou et al. 2023). SWE-agent, OpenHands, and CodeAct showed that interface design and action substrate choices can strongly affect software-agent performance (Wang et al. 2024a,b; Yang et al. 2024). MCP then made tool interoperability itself a protocol-level question rather than an ad hoc detail (Model Context Protocol 2025).

What was often missing was not the mechanism itself, but a commonly named abstraction for the layer that ties these elements together. Anthropic supplied one part of the vocabulary with *agent harness* and *evaluation harness* (Grace et al. 2026). OpenAI then made the broader engineering practice explicit under the name *harness engineering* (Lopopolo 2026). The significance of that naming move is methodological rather than merely terminological: it identifies a recurring systems object whose responsibilities cut across context specification, tool mediation, runtime control, verification, governance, and traces.

 doi:10.20944/preprints202603.1756.v1

2.2. Canonical Public Examples

Three recent families of public examples make the concept concrete. First, OpenAI's harness engineering note treats durable project instructions, architectural constraints, custom linters, cleanup loops, and automated review as the central levers for getting coding agents to produce maintainable code over long horizons (Lopopolo 2026). Second, Anthropic's long-running harness work frames state externalization, initialization, progress files, and recovery discipline as prerequisites for useful multi-session agency (Young 2025). Third, Anthropic's multi-agent research system and browser-style evaluation work make the same point outside repository coding: source hierarchy, tool access, parallel exploration, memory structure, and recovery policy can materially change behavior in research and browsing settings (Hadfield et al. 2025; Mialon et al. 2023; Wei et al. 2025; Zhou et al. 2023). Table 1 shows that these materials converge from different angles on the same point: if we want to understand language agents scientifically, we need to reason about the harness layer that turns a model into a system that can act, recover, and be governed.

This is also where the present paper diverges from adjacent surveys. Recent overviews of autonomous agents, multi-agent systems, and agent evaluation are valuable, but they mostly treat the agent, collaboration pattern, or benchmark as the unit of analysis rather than the harness layer itself (Luo et al. 2025; Piccialli et al. 2025; Wang et al. 2024; Yehudai et al. 2025). Our claim is not that those perspectives are wrong. It is that a harness-centered perspective cuts across them and helps explain why systems that look similar at the model level often behave differently in practice.

Table 2. The engineering object widens from prompts to context and then to the harness around the model. The labels differ not only in scope, but in what they make scientifically visible.

Frame	Main engineering question	Typical artifacts	What remains under-described if this frame is treated as sufficient
Software engineering	How should the system stay correct and maintainable?	modules, interfaces, tests, CI, operational procedures	model-facing instructions, evolving context, and agent-specific control policies
Prompt engineering	What should the model be told?	system prompts, examples, roles, output schemas	retrieval, memory, runtime policy, permissions, and tool mediation
Context engineering	What should the model see right now?	retrieved snippets, message history, tool descriptions, summaries, notes	action interfaces, approval logic, recovery policy, and observability over time
Harness engineering	How should a language agent be governed over time?	durable instructions, tool contracts, checkpoints, graders, budgets, approvals, traces	the agent is no longer reduced to the model alone; the harness layer becomes the thing that must be reported

3. A Working Definition: The Harness Layer

The public formulations above are suggestive rather than settled. Building on them, we propose a working definition for this paper. We define the *harness layer* as the extra-model layer that determines what an agent sees, what it can do, how its work unfolds over time, which feedback it receives, and how that behavior is constrained, observed, and evaluated. Harness engineering is the design and maintenance of that layer. Operationally, the harness layer mediates between model capability and situated action: it translates model outputs into governed execution and turns environmental feedback back into actionable state.

We write the harness layer compactly as $H = \langle C, A, R \rangle$, where C is the *control* layer, A is the *agency* layer, and R is the *runtime* layer. We use the acronym *CAR* because it keeps the ordering explicit and highlights that the harness is not just runtime plumbing. The notation is not meant as a rigid ontology. Its purpose is explanatory: it marks three coupled functions that papers routinely bundle together while leaving under-described.

Control.

The control layer contains durable artifacts that shape behavior before a step is taken: repository maps, `AGENTS.md`, tool descriptions, system instructions, architecture rules, tests, linters, permission

 doi:10.20944/preprints202603.1756.v1

policies, and success criteria (Lopopolo 2026; OpenAI 2026a,b). In other words, control is where human judgment becomes machine-readable constraint. A key harness insight is that reliable agents are rarely bounded by prompt wording; they are often bounded by specifications.

Agency.

The agency layer determines how the model is allowed to act. It includes action substrates such as code execution or browser interaction, planner–verifier or orchestrator–worker structures, reviewer roles, and the concrete interfaces that define the action space (Hadfield et al. 2025; Wang et al. 2024a; Yang et al. 2024). We use *agency* here in a narrow systems sense: the mediated action surface and delegation structure that the harness permits, not a claim about unrestricted autonomy or general capability.

Runtime.

The runtime layer governs what happens as work unfolds over time: context assembly, memory and compaction, checkpointing, retries, backtracking, approval flows, budgets, trace collection, and replay support (Chen 2026; Rajasekaran et al. 2025; Young 2025). This is where long-horizon behavior succeeds or collapses. Many agent failures are runtime failures: stale state, brittle retry loops, overgrown context, or poor recovery from intermediate mistakes.

Two concrete mini-cases.

Consider first a repository coding agent. Two systems may share the same frontier model and nearly the same task prompt, yet behave very differently because one harness adds a repository map, root-level `AGENTS.md`, required tests, a linter, bounded shell access, a progress file, and manual approval for privileged actions. The CAR lens explains why these are not minor details: the map, tests, and approval policy live in control; the shell and file-edit surface live in agency; and the progress file, retries, and escalation logic live in runtime. The reported performance is already partly a property of the harness layer, not of the model alone.

Now consider a browser or research agent. Two systems may share the same browsing-capable model and high-level task prompt, yet differ because one harness defines a source hierarchy, citation rules, note-taking format, uncertainty-triggered escalation, tool quotas, and replayable traces (Hadfield et al. 2025; Mialon et al. 2023; Wei et al. 2025; Zhou et al. 2023). Here, control includes source authority and citation policy; agency includes the search, browser, and delegation surface; and runtime includes scratchpads, branching traces, and recovery when evidence conflicts. Again, what looks like "agent quality" is partly a property of the harness layer around the model.

This definition also clarifies what harness engineering is *not*. It is broader than prompt engineering because prompts are only one artifact inside a larger control structure. It is narrower than "agent systems" because not every property of the environment belongs to the harness. It overlaps with platform engineering and MLOps, but it is more specifically about the layer through which language-centered models become usable agents. The harness begins where the system actively shapes trajectories: curated context, mediated tools, retries, checkpoints, graders, permissions, traces, and similar control points.

Two boundary clarifications are especially important. First, harness engineering is not merely another name for prompting. A paper that reports only the prompt while omitting which files count as the system of record, which tools can be called, which tests are binding, what is remembered, and when humans or policies must intervene is often omitting the most consequential part of the system. Second, harness engineering is not equivalent to all software infrastructure around a model. A web front-end that forwards text to a model is not yet a harness in the sense we mean. The harness begins where the system actively shapes trajectories over time, mediates action, encodes constraints, manages recovery, and produces traces that let others inspect what happened.

 doi:10.20944/preprints202603.1756.v1

Table 3. A lightweight audit of 40 in-scope works (Appendix A.3). Counts reflect whether a work primarily foregrounds a harness component; tags are non-exclusive, so rows exceed n. The pattern is suggestive within this selected evidence base: papers and benchmarks emphasize interfaces and observability, while engineering notes emphasize runtime, control, and governance.

Source type	Control	Agency	Interfaces	Runtime	Governance	Feedback	Observability
Papers / benchmarks ($n = 22$)	4	8	8	4	3	5	9
Notes / protocols / technical articles ($n = 18$)	6	2	4	8	5	3	4

4. A Lightweight Descriptive Audit of the Evidence Base

To keep the argument from being purely terminological, we ran a lightweight descriptive audit over the 40 in-scope works in Appendix A.3, reusing the primary harness-component tags already assigned in the inventory. The audit is intentionally modest: it is a visibility check over the evidence base used in this paper, not a prevalence estimate for the whole field. Still, the pattern is informative.

Table 3 indicates that papers and benchmarks in our evidence base most often foreground *interfaces*, *agency*, and *observability*. Public engineering notes, protocol documents, and technical articles more often foreground *runtime*, *control*, and *governance*. In other words, the academic-facing literature more readily names the action surface and the evaluation setting, whereas practitioner-facing documents more readily describe the durable instructions, recovery policies, and permission structures that make a system work day to day.

We do not claim that this audit settles a bibliometric question. The coding is interpretive and the evidence base is selective by design. But the asymmetry helps explain why the harness layer can feel newer in academic NLP than it does in practice: the parts of the layer that matter most for deployment and transfer are also the parts most often surfaced in engineering notes rather than formal system papers. This gap is one reason a reporting artifact like HARNESSCARD is useful.

5. Why the Harness Layer Changes What Counts as Progress

Many reported agent gains can be partly harness-sensitive.

When two systems use similar frontier models yet behave very differently, the explanation often lies at least partly in the harness layer. Software agents can improve when their action substrate is redesigned, not only when their model is swapped (Wang et al. 2024a; Yang et al. 2024). Interactive benchmarks can become more tractable when traces, retries, hidden checks, and verifiers are better coordinated (Jimenez et al. 2023; Liu et al. 2023; Pan et al. 2024; Zhou et al. 2023). Long-running agents can improve when state is externalized and progress is resumable (Wang et al. 2023; Young 2025). Research and browsing agents can improve when orchestration, source selection, and memory structure are better aligned with the task (Hadfield et al. 2025; Wei et al. 2025). Even where models matter greatly, the last mile of reliability is frequently a harness question.

Evaluation must become harness-sensitive.

Once an agent acts through tools and over time, evaluating the model in isolation misses the object. Anthropic makes this explicit: an agent evaluation is an evaluation of harness plus model (Grace et al. 2026). That has four implications. First, evaluations should measure trajectories and outcomes, not only strings. Second, papers should report action budgets, retries, checkpoints, graders, and interventions because these can change results. Third, productivity or reliability claims become more interpretable when paired with explicit acceptance tests and operating envelopes rather than anecdotal claims. Fourth, variance should be expected: interactive systems are more sensitive to infrastructure noise and control policies than static benchmark runs. Anthropic argues that small leaderboard gaps on agentic coding evaluations deserve skepticism until the evaluation resource configuration is documented and matched (Segato 2026).

Harness sensitivity also changes what a fair comparison means. Matching model family while changing tool access, retry budgets, verifier strictness, or escalation policy is not a controlled compari-

 doi:10.20944/preprints202603.1756.v1

son; nor is holding the prompt fixed while silently changing memory compaction or checkpointing. In agent settings, the experimental unit is the coupled execution regime. That is why papers should report not only success rate, but also the envelope within which success was obtained: budget ceilings, allowed side effects, approval requirements, and whether failures were recoverable or terminal. Without that envelope, benchmark numbers can look commensurable while actually reflecting different kinds of systems.

Reproducibility now depends on the harness layer.

A language-agent paper can appear more novel than it really is if the harness is under-described. Retry budgets, hidden human escalations, tool filters, repository instructions, or grader prompts often remain implicit even when they are load-bearing. This is not a minor reporting issue. It obscures which gains transfer across settings, which ones depend on domain-specific control logic, and which ones are artifacts of a particular runtime. A harness-centered research practice would reverse that asymmetry: the extra-model layer would be described as carefully as the model.

Table 4. Recurring harness patterns and associated reliability profiles; broader inventory in Appendix A.3.

Pattern	Typical harness levers	Representative works	Common strengths and failure modes
Single-agent tool loop	prompt assembly, tool schemas, light memory, bounded retries	ReAct (Yao et al. 2022), Toolformer (Schick et al. 2023), API-Bank (Li et al. 2023)	simple and efficient, but brittle when tasks are long, under-specified, or tool-heavy
Executable action substrate	code as action language, interpreter feedback, self-debugging	CodeAct (Wang et al. 2024a), OpenHands (Wang et al. 2024b)	flexible and compositional, but can amplify side effects without strong governance
Agent–computer interface	constrained command surface, file editing, search, browser or GUI actions	SWE-agent (Yang et al. 2024), OS-World (Xie et al. 2024)	large gains from interface design, but failures shift to navigation and grounding
Orchestrator–worker topology	decomposition, role specialization, routing, verifier or reviewer roles	AutoGen (Wu et al. 2024), MetaGPT (Hong et al. 2023), ChatDev (Qian et al. 2024), Anthropic research system (Hadfield et al. 2025)	better coverage and parallelism, but coordination overhead and cascading errors remain common
Long-running harness	state externalization, checkpoints, progress files, resumability	Voyager (Wang et al. 2023), BrowseComp (Wei et al. 2025), Anthropic long-running harnesses (Young 2025)	supports hours-long work, but state drift and context decay remain central risks
Policy-aware deployment	permissions, sandboxing, escalation, audit logs, protocol mediation	τ-bench (Yao et al. 2024), sandboxing notes (Dworken et al. 2025), MCP (Model Context Protocol 2025)	improves safety and accountability, but can reduce autonomy or hide control logic if under-reported

6. Why NLP Should Treat the Harness Layer as an Explicit Object of Study

At the moment, many of the clearest public descriptions of harness work come from product teams rather than from academic papers. That should be read as an opportunity rather than as a reason for the research community to stay away. NLP has repeatedly advanced by turning messy practice into explicit method: annotation, evaluation, retrieval, prompting, and human feedback all followed that path. Harness engineering is ready for the same move. The field can contribute formal task definitions, benchmark suites that isolate control-layer effects, reporting norms that travel across domains, and theories of when language is the right interface for state, tools, and oversight.

What makes harness engineering especially relevant to NLP is that the harness layer itself is often made of language-bearing artifacts. Repository maps, task decompositions, tool descriptions, approval prompts, error summaries, progress files, and policy messages are not just interface copy; they are operational control media. The harness therefore turns language from output modality into an instrument for specification, recovery, and oversight. That is exactly why NLP should care: questions of wording, authority, grounding, and state representation are no longer peripheral UX details, but part of system correctness.

A harness-centered program would also improve the quality of agent claims. The descriptive audit above suggests that academic papers more readily surface interfaces and evaluation than the

 doi:10.20944/preprints202603.1756.v1

durable control and runtime policies that often decide whether a system transfers. A stronger norm would reward the opposite behavior: state the harness clearly, vary it experimentally, and explain which parts are portable versus domain-specific.

Treating the harness as a layer also sharpens baselines. The most revealing comparison is often not only model A versus model B, but thinner versus richer control, narrower versus broader action surfaces, or stateless versus recoverable runtime on the same model. Layer-aware baselines would let the field estimate when progress comes from the model and when it comes from the layer around it. Without them, attribution and reproducibility remain entangled.

Recent adjacent position pieces on automated research systems, public agent ecosystems, and acceptance-test-centered productivity reporting reinforce neighboring parts of the same shift, even though they do not use the harness-layer framing directly (He et al. 2026a,b,c).

Two likely objections.

A natural objection is that harness engineering is "just software engineering." That objection misses the object-specific nature of the problem. Harnesses for language agents are not generic infrastructure. Their control artifacts are often partly linguistic objects: instructions, repository maps, tool descriptions, summaries, approval prompts, grader criteria, and progress files. Likewise, their governance is often mediated through language, not only through low-level system calls. Another objection is that the term may be too new or too vendor-specific to anchor research. That concern is reasonable, but it cuts in favor of clarification rather than silence. When a recurring design problem becomes visible across multiple agent families, making the abstraction explicit helps the field compare systems more honestly.

7. Research Questions from the Harness-Layer Lens

The harness perspective makes several research questions easier to see. One concerns *authority in context*. Which artifacts should an agent trust most when repository documentation, retrieved snippets, tool outputs, policy text, and runtime observations disagree? OpenAI's emphasis on a repository "system of record" and Anthropic's emphasis on progress files are practical answers, but NLP can ask the deeper question of how language should represent state so that later decisions remain grounded (Lopopolo 2026; Young 2025). This is not only a product question. It is a question about language as a control medium.

A second family of questions concerns *recovery*. Benchmarks often reward eventual completion, but long-running systems live or die by their ability to recover from bad intermediate steps. When should a harness retry, roll back, checkpoint, escalate, or terminate? How much memory of earlier failed attempts should be carried forward? Which summaries preserve the right information for future repair without contaminating later reasoning? These are not marginal implementation details. They determine whether an agent remains useful over repeated real-world runs.

A third family concerns *governance through language*. Policy checks, approval prompts, uncertainty statements, and provenance traces are often treated as UX residue. For language agents, however, they are part of the grammar through which the system and the human coordinate. What wording leads to calibrated escalation rather than either over-blocking or over-compliance? How should an agent explain a blocked action, a tool request, or a risky proposed next step? How should responsibility be assigned when a failure is partly model error and partly harness error? These questions sit at the boundary of NLP, HCI, security, and software engineering, and they become sharper once the harness layer is treated as an explicit object of study.

A fourth family concerns *transfer*. Some improvements travel with the model, some with the harness, and some only with a task environment. A stronger repository map or verifier loop may transfer across coding domains yet fail to help browser tasks; a better browsing policy may transfer across websites yet not to shell-heavy workflows. Reporting model transfer, harness transfer, and task transfer separately would make agent papers more cumulative because readers could tell whether a claimed gain is about general capability, portable control logic, or local infrastructure tuning.

 doi:10.20944/preprints202603.1756.v1

Table 5. Compact main-paper HARNESSCARD; governance and observability are merged here only for space, while Appendix B separates them and Appendix B.2 gives a filled example.

Field	Minimum disclosure	Priority
Base model(s)	model name, version, and decoding or adaptation settings	Required
Control artifacts	instructions, repo maps, `AGENTS.md`, architecture rules, tests, linters, success criteria	Required
Runtime policy	memory or compaction strategy, checkpoints, retries, rollback or escalation policy, budgets	Required
Action substrate	tools, APIs, browser or GUI access, code execution, interface schemas, MCP usage	Required
Execution topology	single-agent vs multi-agent structure, verifier or reviewer roles, routing logic	Required
Feedback stack	tests, graders, hidden checks, reflection prompts, human interventions	Required
Governance / observability	permissions, sandboxing, provenance logs, replay support, failure categories	Required
Evaluation protocol	task set, number of runs, outcome criteria, variance treatment, budget limits	Required

8. Design Patterns & Failure Modes

A harness-centered view also makes recurring design patterns easier to compare. Some systems rely on a relatively thin single-agent tool loop; others give the model an executable action substrate such as code; others build an agent–computer interface for browsing or software editing; others use orchestrator–worker or planner–verifier structures; and others emphasize long-running state plus policy-aware deployment. These are not cosmetic packaging choices. They redistribute where error happens and which interventions are possible.

The same is true for failure. Context drift, action-schema mismatch, planning collapse, verifier overfitting, policy violations, and cost blow-up are often discussed as separate problems. Through the CAR lens, these failures are related: each type appears when control, agency, and runtime are poorly matched to the task. Table 4 summarizes representative patterns and the failures they tend to surface. The point is not that one pattern is best. It is that different harness choices produce different reliability profiles, and agent papers should say so explicitly.

This has methodological consequences. A harness-centered experiment should not only ask whether a system succeeds, but also which control artifacts, runtime policies, and interface choices were necessary for success. That means benchmarking patterns as patterns rather than treating them as unreported implementation residue. It also means that papers should report when a failure was recovered by the harness rather than by the model alone, because recovery behavior is often one of the most practically important parts of the system.

9. A Research & Reporting Agenda

Harness engineering should not be treated as an implementation afterthought. It opens a distinct research agenda for NLP.

First, study control as executable specification.

Repository maps, `AGENTS.md`, architecture rules, and cleanup loops suggest that agentic "instruction following" is partly a problem of specification design, not only of compliance (Lopopolo 2026; OpenAI 2026b). NLP can study how language, code, schemas, and tests should be composed so that guidance remains durable and authoritative.

Second, treat agency as an interface question.

The agent's action space is harness-mediated. Tool schemas, interface design, and execution substrate can dominate both performance and safety (Aizawa et al. 2025; Wang et al. 2024a; Xie et al. 2024; Yang et al. 2024). On this view, agent capability is a property of the model inside a specific interface and control regime.

Third, treat runtime as a scientific variable.

Compaction, state persistence, recovery policy, and execution budgets are not residue. They determine whether a system remains coherent over long horizons (Choi 2026; Rajasekaran et al. 2025; Young 2025). Research should ask which runtime policies preserve the right information and when agents should backtrack, escalate, or recompute.

 doi:10.20944/preprints202603.1756.v1

Fourth, normalize reporting of the harness.

We propose HARNESSCARD, a lightweight reporting artifact for language-agent systems. Table 5 gives the compact main-paper version. At minimum, it should disclose the base model, control artifacts, runtime policy, action substrate, feedback stack, governance layer, and evaluation protocol; Appendix B gives the template and Appendix B.2 a filled example. The goal is not bureaucracy but auditable, transferable results. Unlike model cards, HARNESSCARD documents the apparatus that makes an agent claim interpretable. The appendix mini-cases in Appendix C suggest that the template travels beyond coding, and authors can disclose proprietary harnesses faithfully at a higher level of abstraction, including what was withheld.

Fifth, build layer-aware baselines.

Controlled comparisons should vary one layer at a time: the same model with different control artifacts, the same action substrate with different runtime policy, or the same runtime with different verification and governance regimes. That is necessary to estimate when progress comes from the model and when it comes from the layer. Reporting the harness clarifies both reproducibility and attribution.

10. Conclusion

Harness engineering makes explicit the extra-model layer that governs language agents once they act through tools and time. Through the lens of control, agency, and runtime as a working decomposition, many reported agent gains may be harness-sensitive rather than model-only, and many reproducibility failures are failures to disclose the layer that shapes instruction, action, and recovery. For NLP, the point is not to downplay models, but to study and report the harness layer clearly enough that model progress can be separated more credibly from harness progress, turning agent results into cumulative systems knowledge.

Limitations

This paper makes a selective argument rather than a complete field survey. Its aim is to sharpen an explicit systems layer, not to catalogue every agent paper. The evidence base is therefore intentionally selective and tilted toward work that illuminates the harness directly: interactive benchmarks, software agents, tool-use systems, protocols, and official engineering notes. That improves conceptual focus, but it also means that some adjacent literatures are under-represented, including robotics, embodied control, model-training methods, and general-purpose platform engineering.

A second limitation is conceptual. The public term *harness engineering* is recent, and the surrounding vocabulary is still evolving. Different communities use overlapping labels such as *scaffold*, *agent loop*, *orchestrator*, *runtime*, or *evaluation harness*. Our control–agency–runtime decomposition is therefore an analytic proposal rather than a settled ontology. It is useful because it organizes recurring design decisions, but other decompositions are possible and may prove better as the literature matures.

A third limitation is evidentiary. We rely substantially on official engineering documents from a small number of frontier organizations because they currently provide the clearest public descriptions of harness practice. Those sources are informative, but they are not a substitute for a mature peer-reviewed literature. They also reflect the priorities, products, and deployment settings of the organizations that publish them. Academic work should test how well those lessons transfer to other models, budgets, institutional settings, and languages.

A fourth limitation concerns the descriptive audit. Table 3 is a lightweight visibility check over this paper's in-scope evidence base, not a field-wide bibliometric estimate. The underlying tags are interpretive and non-exclusive. We use the audit to show an asymmetry in what different source types tend to foreground, not to claim a definitive prevalence distribution for the broader literature.

A fifth limitation concerns causal attribution. The paper argues that many reported agent improvements are harness-sensitive, but it does not itself provide controlled ablations that separate

 doi:10.20944/preprints202603.1756.v1

model effects from harness effects. In real systems the two are often entangled: a stronger model may need less scaffolding, while a stronger harness can make a weaker model look far more capable. A fuller empirical program still needs benchmark designs, reporting norms, and intervention studies that estimate how much each layer contributes.

A sixth limitation is scope. The argument is strongest for language agents that operate through tools, files, browsers, and persistent execution. Some language-model applications remain closer to structured prediction or short-form assistance, where the harness is thinner and the term may feel oversized. Likewise, most current public examples come from English-centric, high-resource environments. The research agenda should therefore be tested against multilingual, low-resource, and community-run settings rather than assumed to transfer unchanged.

Finally, HARNESSCARD is proposed here as a reporting artifact, not as a validated community standard. We give a concrete example, a cross-task coverage check, and a rationale, but we do not yet present author studies, reviewer trials, or benchmark evidence showing which fields are sufficient in practice, which are too burdensome, or how disclosure interacts with proprietary constraints. The proposal is intended to be revised in response to community use.

Ethical Considerations

A harness-centered research agenda has clear benefits, but it also carries meaningful risks. Better harnesses can make agents more reliable, and reliability is dual-use. The same control, recovery, and tool-integration improvements that help with benign coding, browsing, writing, or enterprise workflows can also make high-risk automation easier to scale in domains where mistakes or misuse have serious consequences. For that reason, governance should be treated as internal to harness engineering rather than as an optional wrapper added after capability work is complete.

There are also risks of opacity and misrepresentation. Agent papers can overstate autonomy if manual escalation, privileged tool access, hidden review, or extensive hand-tuned control logic are omitted from the description. A harness-sensitive reporting norm is partly an ethical response to that problem: readers should be able to tell where judgment came from, how the system was constrained, which interventions were necessary to obtain the reported result, and which parts of the workflow remained human-dependent.

Privacy and security risks intensify at the harness layer because the harness is where traces, permissions, and connectors live. Runtime logs, memory files, approval records, tool transcripts, and retrieved artifacts may contain sensitive user, organizational, or proprietary information. Tool access expands the attack surface; poorly governed shells, browsers, databases, or retrieval backends can expose systems to leakage, unsafe actions, or policy bypass. The more capable the harness becomes, the more important it is to treat permissions, sandboxing, auditing, and trace retention as first-class design decisions (Dworken et al. 2025; Model Context Protocol 2025; Yao et al. 2024).

A further concern is over-automation. A well-engineered harness can encourage organizations to delegate tasks whose failure modes are still poorly understood because the system feels disciplined and auditable. This can create a false sense of safety. The presence of retries, tests, checkpoints, or approval prompts does not guarantee that the right thing is being optimized or that downstream stakeholders can contest a harmful action in time.

There is also a risk of concentration and uneven access. Durable project instructions, privileged observability stacks, and reusable runtime infrastructure may concentrate advantage in a small number of organizations that can afford intensive harness engineering. That can distort scientific comparison if public papers benchmark base models while keeping the decisive systems layer private. It can also produce an uneven research landscape in which access to the harness, not only access to the model, determines who can participate meaningfully in frontier agent work.

Finally, stronger harnesses can hide human cleanup work just as easily as they can amplify machine capability, and they can substantially increase compute usage through retries, persistent traces, and background execution. Honest reporting should therefore surface where humans remained

 doi:10.20944/preprints202603.1756.v1

in the loop and what budgets, retries, and persistent runtime costs were required to obtain the reported result.

Appendix roadmap. The appendix is organized into four grouped sections to reduce float fragmentation and empty space while keeping the same supplementary material: evidence base, public formulations, and exhaustive inventory (Section A); HARNESSCARD materials (Section B); search strings, glossary, and additional mini-cases (Section C); and expanded visual summaries (Section D).

Acknowledgments: This research is supported by the RIE2025 Industry Alignment Fund (Award I2301E0026) and the Alibaba-NTU Global e-Sustainability CorpLab.

Appendix A. Evidence Base, Public Formulations, and Exhaustive Inventory

Appendix A.1. Evidence Base and Selection Logic

This paper is not a survey of all agent research. It is a selective argument grounded in a structured cited evidence base assembled to support the harness claim. We searched ACL Anthology, arXiv, OpenReview, official engineering notes, standards documents, and public technical articles using query stems such as `harness`, `agent harness`, `scaffold`, `context engineering`, `tool use`, `agent-computer interface`, `interactive evals`, `browser agents`, `software engineering agents`, and `long-running agents`. We retained work if it materially illuminated at least one harness-relevant function: control/specification, runtime/state, mediated agency and interfaces, feedback or verification, governance, or observability/evaluation. The literature cutoff is March 20, 2026.

The cited evidence base in this draft contains 49 unique sources. We divide them into 40 in-scope harness-relevant works and 9 adjacent framing pieces used for historical context, boundary-setting, or neighboring claims. Of the 40 in-scope items, 22 are papers or benchmarks and 18 are official engineering notes, protocol documents, developer guides, or technical articles. 28 of the 40 in-scope items are from 2024–2026. The exhaustive list of the 40 in-scope items appears in Appendix A.3. Because facet labels such as control, agency, runtime, governance, or observability are partly interpretive and non-exclusive, we report them per work in the inventory rather than as a single aggregate facet count table. Table 3 reuses those same inventory tags to provide the main paper's lightweight visibility audit; it should be read as a structured view of this paper's evidence base rather than as a field-wide bibliometric estimate. Figure A1 visualizes the composition of the evidence base, and Table A1 gives the exact counts used in the current draft.

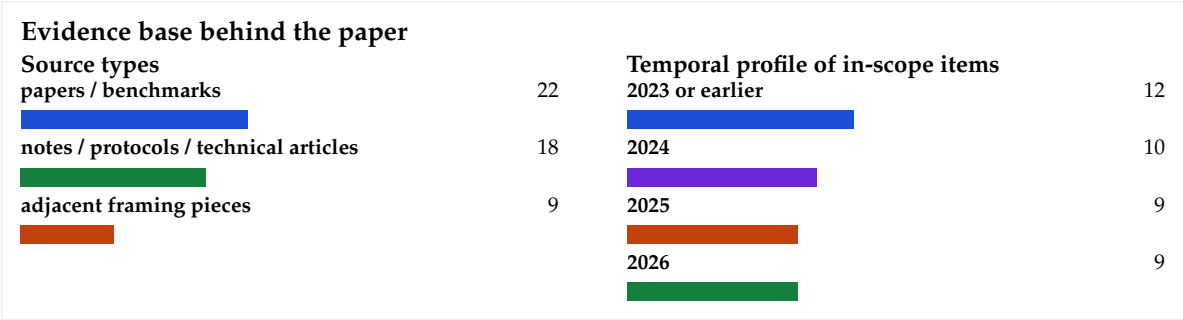

Figure A1. Exact counts for the current draft's cited evidence base. Source-type counts use all cited sources; the temporal profile uses only the 40 in-scope harness-relevant items.

 doi:10.20944/preprints202603.1756.v1

Table A1. Exact profile of the cited evidence base used in this draft.

View	Breakdown	Count
Total cited sources	unique cited sources in the current draft	49
Scope split	in-scope harness-relevant works	40
Scope split	adjacent framing pieces	9
In-scope source type	papers or benchmarks	22
In-scope source type	engineering notes, protocol documents, developer guides, or technical articles	18
In-scope time band	2023 or earlier	12/40
In-scope time band	2024	10/40
In-scope time band	2025	9/40
In-scope time band	2026	9/40
In-scope time band	2024–2026 combined	28/40

Appendix A.2. Public Formulations and Official Examples

Table A2 expands the compact main-paper inventory of public formulations that anchor the working definition.

Table A2. Public formulations and official examples that shape the paper's working definition.

Source	Year	Object or term	What it contributes to the concept
Anthropic evals note (Grace et al. 2026)	2026	agent harness / evaluation harness	defines the harness as the system that enables a model to act as an agent and stresses that agent evaluation measures harness plus model together
OpenAI harness note (Lopopolo 2026)	2026	harness engineering	names the broader practice and ties it to durable instructions, architectural constraints, cleanup loops, and long-horizon coding work
OpenAI Codex harness note (Chen 2026)	2026	Codex harness	treats the harness as reusable runtime logic and protocol support rather than a one-shot prompt surface
Anthropic long-running note (Young 2025)	2025	long-running harnesses	makes state externalization, progress tracking, clean-state discipline, and resumability central design levers
OpenAI developer guidance (Choi 2026; OpenAI 2026a,b)	2026	`AGENTS.md`, long-horizon tasks	provides concrete examples of harness work as durable project guidance plus iterative plan–act–test–repair loops
MCP specification (Model Context Protocol 2025)	2025	protocol layer	shows that tool interoperability and permission boundaries can themselves be harness design objects

Appendix A.3. Exhaustive in-Scope Works

Tables A3 and A4 provide the auditable list of the 40 in-scope harness-relevant works cited in this draft. Table A3 covers the earlier part of the inventory, while Table A4 continues with later papers, notes, protocols, and developer materials.

 doi:10.20944/preprints202603.1756.v1

Table A3. Exhaustive in-scope works cited in the paper, Part I.

Work	Year	Type	Why it is in scope	Primary harness component(s)
Yao et al. (2022)	2022	precursor paper	couples reasoning and acting in trajectories	agency, control
Schick et al. (2023)	2023	paper	makes tool calls an explicit part of model behavior	interfaces, agency
Shinn et al. (2023)	2023	paper	treats reflection as trajectory repair and memory support	runtime, feedback
Wang et al. (2023)	2023	paper	shows skill libraries and resumable state for long tasks	runtime, feedback
Li et al. (2023)	2023	paper	role-structured multi-agent execution	agency
Li et al. (2023)	2023	benchmark	API tool use and tool evaluation	interfaces, observability
Liu et al. (2023)	2023	benchmark	interactive agent evaluation	observability
Zhou et al. (2023)	2023	benchmark	web interaction as environment-level action	interfaces, observability
Wu et al. (2024)	2024	paper	orchestrated multi-agent conversations and control logic	agency, control
Hong et al. (2023)	2023	paper	specialist-agent software workflows	agency
Mialon et al. (2023)	2023	benchmark	general-assistant tasks that stress tool use and grounded action	observability, agency
Jimenez et al. (2023)	2023	benchmark	executable software tasks and hidden tests	feedback, observability
Khattab et al. (2023)	2023	paper	LM pipeline compilation and self-improvement	control, feedback
Qian et al. (2024)	2024	paper	role-specialized software agents	agency
Wang et al. (2024a)	2024	paper	code as an executable action substrate	interfaces, agency
Xie et al. (2024)	2024	benchmark	open computer-use environment	interfaces, observability
Yang et al. (2024)	2024	paper	agent-computer interface for software engineering	interfaces, control
Yao et al. (2024)	2024	benchmark	policy-aware tool-agent-user interaction	governance, observability
Wang et al. (2024b)	2024	paper	open software-agent platform with sandboxed runtime	interfaces, governance, runtime
Xu et al. (2024)	2024	benchmark	consequential workplace tasks for agents	governance, observability

Table A4. Exhaustive in-scope works cited in the paper, Part II.

Work	Year	Type	Why it is in scope	Primary harness component(s)
Pan et al. (2024)	2024	paper	trainable loops with verifiers and repair	feedback, observability
Wei et al. (2025)	2025	benchmark	persistent browsing and recovery	interfaces, runtime
Schluntz and Zhang (2024)	2024	note	codifies practical agent patterns and the workflow/agent distinction	control, feedback
Anthropic (2025)	2025	note	explicit reflection insertion in tool loops	feedback
Hadfield et al. (2025)	2025	note	orchestrator–worker research system	agency
Rajasekaran et al. (2025)	2025	note	state curation and compaction as engineering	runtime
Aizawa et al. (2025)	2025	note	interface and tool-surface design for agents	interfaces, governance
Dworken et al. (2025)	2025	note	security and containment for coding agents	governance
Wu et al. (2025)	2025	note	large tool surfaces and long-running tool access	interfaces, runtime
Young (2025)	2025	note	state externalization, resumability, and recovery	runtime
Grace et al. (2026)	2026	note	agent-harness and evaluation-harness distinction	observability, feedback
Segato (2026)	2026	note	infrastructure variance in agentic evaluation	observability
Model Context Protocol (2025)	2025	protocol	tool interoperability and permissions as protocol objects	interfaces, governance
Lopopolo (2026)	2026	note	explicit naming of harness engineering and its practical levers	control, runtime, governance, observability
Chen (2026)	2026	note	reusable runtime logic and protocol support	runtime, interfaces
Bolin (2026)	2026	note	stepwise action loop and runtime structure	agency, runtime
OpenAI (2026b)	2026	developer guide	durable in-repository instruction surfaces	control
OpenAI (2026a)	2026	developer guide	operational guidance for agentic coding loops	control, runtime
Choi (2026)	2026	developer blog	long-horizon execution discipline and resumability	runtime, control
Böckeler (2026)	2026	technical article	software-architecture articulation of the harness concept	control, governance, observability

 doi:10.20944/preprints202603.1756.v1

Appendix B. HARNESSCARD Materials

Appendix B.1. Expanded HARNESSCARD

The template below expands the compact main-paper version of HARNESSCARD into a fuller disclosure schema. Table A5 separates governance and observability and adds recommended release and risk fields.

Table A5. HARNESSCARD: a lightweight reporting artifact for language-agent systems.

Field	What should be disclosed	Priority
Base model(s)	model name, version, decoding settings, and any finetuning or adapters	Required
Control artifacts	system instructions, `AGENTS.md`, repo maps, architecture rules, schemas, tests, linters, done-when criteria	Required
Runtime policy	memory type, compaction or summarization policy, checkpointing, retry or rollback policy, budget limits	Required
Action substrate	tools, APIs, browser or GUI access, code execution, interface schemas, MCP usage	Required
Execution topology	single-agent vs multi-agent structure, planner/verifier roles, reviewer loops, routing logic	Required
Feedback stack	tests, graders, reflection prompts, hidden checks, human interventions, or repair loops	Required
Governance layer	permissions, sandboxing, escalation rules, policy checks, provenance logging, audit support	Required
Observability	stored traces, replay support, latency and cost logging, failure categories	Required
Evaluation protocol	task set, number of runs, success criteria, variance treatment, held-out checks or budget limits	Required
Release artifacts	prompts or programs, tool specs, traces, configs, environment setup, reproducibility notes	Recommended
Known limitations and risks	unresolved failure modes, portability caveats, safety concerns, or red-team findings	Recommended

Appendix B.2. Illustrative HARNESSCARD*: Repository Coding Agent*

The filled example below shows how the fields in HARNESSCARD can be instantiated for a repository coding agent. Table A6 is illustrative rather than product-specific and is meant to show the expected granularity of disclosure.

 doi:10.20944/preprints202603.1756.v1

Table A6. A filled illustrative HARNESSCARD for a repository coding agent. The example is a synthesis of recurring patterns in public OpenAI and Anthropic materials rather than a reverse-engineered product specification.

Field	Illustrative disclosure	Why it matters
Base model(s)	frontier coding model configured through repo or user profiles; effort tuned for long tasks	keeps model choice distinct from harness choice
Control artifacts	root-level `AGENTS.md`; repository map; build/test/lint commands; architecture rules; done-when criteria	reveals the durable instructions and constraints the agent actually reads
Runtime policy	repository treated as system of record; thread history; progress file; compaction near context limits; bounded retries	makes long-horizon state handling explicit
Action substrate	file edits, shell commands, test runs, diff generation, PR review, optional MCP tools	discloses what the model can actually do in the environment
Execution topology	plan → edit → run tools → observe → repair → update status → repeat; optional reviewer loop	captures the control structure rather than only the model
Feedback stack	failing tests, custom linter messages, self-review, grader checks, occasional human review	surfaces the verification signals that shape behavior
Governance layer	sandbox mode, approval policy for privileged actions, least-privilege connectors, audit trail	keeps permissions and safety visible rather than implicit
Observability	persisted thread events, replay support, latency and cost logs, categorized failures	makes debugging and comparison scientifically possible
Success criteria	merged change passes required checks, stays within budget, and leaves updated status artifacts	completion becomes operationally verifiable, not merely verbal
Known risks	stale docs, state drift, verifier overfitting, hidden human intervention, over-trusting automated review	shows why limitation disclosure belongs inside the reporting standard

Appendix C. Search Strings, Glossary, and Additional Mini-Cases

The evidence base was assembled using query stems such as `harness`, `agent harness`, `scaffold`, `context engineering`, `prompt engineering`, `tool use`, `interactive evals`, `browser agents`, `agent-computer interface`, `long-running agents`, and `software engineering agents`. We excluded work that used the word *agent* only in a loose product-marketing sense without giving enough detail about control artifacts, runtime structure, action interfaces, evaluation logic, or governance. We also excluded papers whose contribution was primarily model training or alignment unless the paper materially illuminated the extra-model layer around agent execution. The 9 adjacent framing pieces used outside the in-scope inventory are historical or boundary-setting citations rather than direct evidence for the harness taxonomy. Table A7 provides a compact glossary for neighboring terms, and Table A8 extends the cross-task illustration with additional mini-cases.

 doi:10.20944/preprints202603.1756.v1

Table A7. A compact glossary for neighboring terms that are often conflated in agent papers.

Term	Working meaning in this paper
Prompt engineering	writing and organizing instructions, examples, and role structure for desired model behavior
Context engineering	curating the evolving token state supplied to the model, including retrieval, memory, and tool context (Rajasekaran et al. 2025)
Agent harness / scaffold	the extra-model system that enables a model to act as an agent (Grace et al. 2026)
Harness layer	the extra-model layer that, in this paper's working definition, couples control artifacts, mediated action interfaces, and runtime policies into governed execution
Harness engineering	the design and maintenance of the control, agency, and runtime layer around the model (Lopopolo 2026)
Evaluation harness	the system that turns tasks, metrics, graders, and infrastructure into an executable evaluation regime (Grace et al. 2026)
Action substrate	the interface through which the agent can act: code, shell, browser, GUI, APIs, or role-structured delegation

Table A8. Mini-cases illustrating how the same control–agency–runtime decomposition appears across task families.

Task family	Control levers	Runtime levers	Agency levers and likely risks
Repository coding agent	repo map, `AGENTS.md`, tests, linters, architectural constraints	compaction, checkpoints, retries, cleanup passes, cost budgets	shell/file edit/PR review; risks include stale docs, verifier overfitting, and hidden human repair
Browser or research agent	source hierarchy, citation rules, task decomposition, grading rubric	search history, scratchpads, branching traces, escalation on uncertainty	browse, fetch, cite, summarize; risks include source drift, unsupported synthesis, and provenance loss
Enterprise support agent	policy text, workflow scripts, escalation rules, approval thresholds	queue state, customer history, retry and timeout policy, audit logs	tool/API access, human handoff, permissions; risks include privacy leakage, over-escalation, and inconsistent policy application

Appendix D. Expanded Timeline and Framework

Figures A2 and A3 restate the main paper's timeline and CAR decomposition in larger visual forms for quick reference. The first expands the historical widening view, and the second enlarges the control–agency–runtime breakdown.

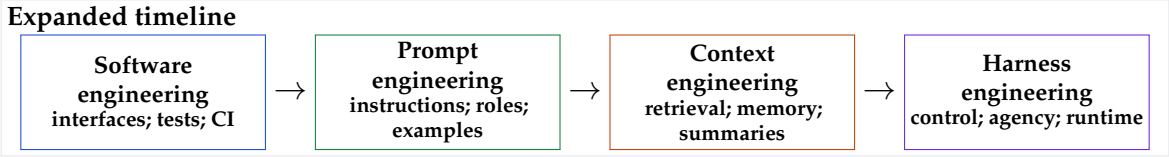

Figure A2. Expanded timeline view. The public label *harness engineering* is recent, but the design problem had already appeared in trajectories, tool use, interactive evaluation, action interfaces, context management, and long-running execution.

 doi:10.20944/preprints202603.1756.v1

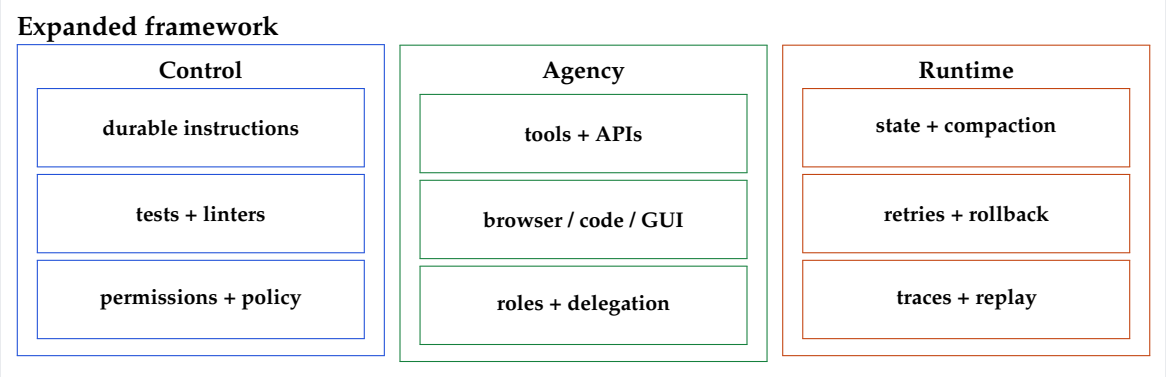

Figure A3. Expanded framework view. The harness layer becomes explicit when the extra-model layer is decomposed into control, agency, and runtime.

References

Aizawa, Ken, Barry Zhang, Zachary Witten, Daniel Jiang, Sami Al-Sheikh, Matt Bell, Maggie Vo, Theodora Chu, John Welsh, David Soria Parra, Adam Jones, Santiago Seira, Molly Vorwerc, Drew Roper, Christian Ryan, and Alexander Bricken. 2025. Writing effective tools for agents — with agents. Anthropic engineering note.

Anthropic. 2025. The "think" tool: Enabling Claude to stop and think in complex tool use situations. Anthropic engineering note.

Böckeler, Birgitta. 2026. Harness engineering. Thoughtworks article.

Bolin, Michael. 2026. Unrolling the Codex agent loop. OpenAI engineering note.

Chen, Celia. 2026. Unlocking the Codex harness: how we built the app server. OpenAI engineering note.

Choi, Derrick. 2026. Run long horizon tasks with codex. OpenAI developer blog.

Dworken, David, Oliver Weller-Davies, Meaghan Choi, Catherine Wu, Molly Vorwerck, Alex Isken, Kier Bradwell, and Kevin Garcia. 2025. Beyond permission prompts: making Claude code more secure and autonomous. Anthropic engineering note.

Grace, Mikaela, Jeremy Hadfield, Rodrigo Olivares, and Jiri De Jonghe. 2026. Demystifying evals for AI agents. Anthropic engineering note.

Hadfield, Jeremy, Barry Zhang, Kenneth Lien, Florian Scholz, Jeremy Fox, and Daniel Ford. 2025. How we built our multi-agent research system. Anthropic engineering note.

He, Chaoyue, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. 2026a. The autoresearch moment: From experimenter to research director.

He, Chaoyue, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. 2026b. Human-ai productivity claims should be reported as time-to-acceptance under explicit acceptance tests. https://doi.org/10.36227/techrxiv.177040595.50580086/v1.

He, Chaoyue, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. 2026c. Openclaw as language infrastructure: A case-centered survey of a public agent ecosystem in the wild.

Hong, Sirui, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The twelfth international conference on learning representations*.

Jimenez, Carlos E, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Khattab, Omar, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Li, Guohao, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems 36*, 51991–52008.

Li, Minghao, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 3102–3116.

Liu, Xiaoxia, Jingyi Wang, Xiaohan Yuan, Jun Sun, Guoliang Dong, Peng Di, Wenhai Wang, and Dongxia Wang. 2023. Prompting frameworks for large language models: A survey. *ACM Computing Surveys*.

 doi:10.20944/preprints202603.1756.v1

Liu, Xiao, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Lopopolo, Ryan. 2026. Harness engineering: leveraging Codex in an agent-first world. OpenAI engineering note.

Luo, Junyu, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, et al. 2025. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460*.

Mialon, Grégoire, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations*.

Model Context Protocol. 2025. Model context protocol specification. Model Context Protocol specification, version 2025-11-25.

Naur, Peter and Brian Randell. 1969. *Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*.

OpenAI. 2026a. Best practices. OpenAI Codex developer guide.

OpenAI. 2026b. Custom instructions with agents.md. OpenAI Codex developer guide.

Pan, Jiayi, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*.

Piccialli, Francesco, Diletta Chiaro, Sundas Sarwar, Donato Cerciello, Pian Qi, and Valeria Mele. 2025. Agentai: A comprehensive survey on autonomous agents in distributed ai for industry 4.0. *Expert Systems with Applications 291*, 128404.

Qian, Chen, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186.

Rajasekaran, Prithvi, Ethan Dixon, Carly Ryan, Jeremy Hadfield, Rafi Ayub, Hannah Moran, Cal Rueb, and Connor Jennings. 2025. Effective context engineering for AI agents. Anthropic engineering note.

Schick, Timo, 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. *Advances in neural information processing systems 36*, 68539–68551.

Schluntz, Erik and Barry Zhang. 2024. Building effective agents. Anthropic engineering note.

Segato, Gian. 2026. Quantifying infrastructure noise in agentic coding evals. Anthropic engineering note.

Shinn, Noah, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems 36*, 8634–8652.

Wang, Guanzhi, 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. *arXiv preprint arXiv:2305.16291*.

Wang, Lei, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science 18*(6), 186345.

Wang, Xingyao, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Wang, Xingyao, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Wei, Jason, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

Wu, Bin, Adam Jones, Artur Renault, Henry Tay, Jake Noble, Noah Picard, Sam Jiang, et al. 2025. Introducing advanced tool use on the Claude developer platform. Anthropic engineering note.

Wu, Qingyun, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2024. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*.

Xie, Tianbao, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems 37*, 52040–52094.

 doi:10.20944/preprints202603.1756.v1

Xu, Frank F, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. 2024. Theagentcompany: benchmarking llm agents on consequential real world tasks. *arXiv preprint arXiv:2412.14161*.

Yang, John, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems 37*, 50528–50652.

Yao, Shunyu, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Yehudai, Asaf, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. Survey on evaluation of llm-based agents. *arXiv preprint arXiv:2503.16416*.

Young, Justin. 2025. Effective harnesses for long-running agents. Anthropic engineering note.

Zhou, Shuyan, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.