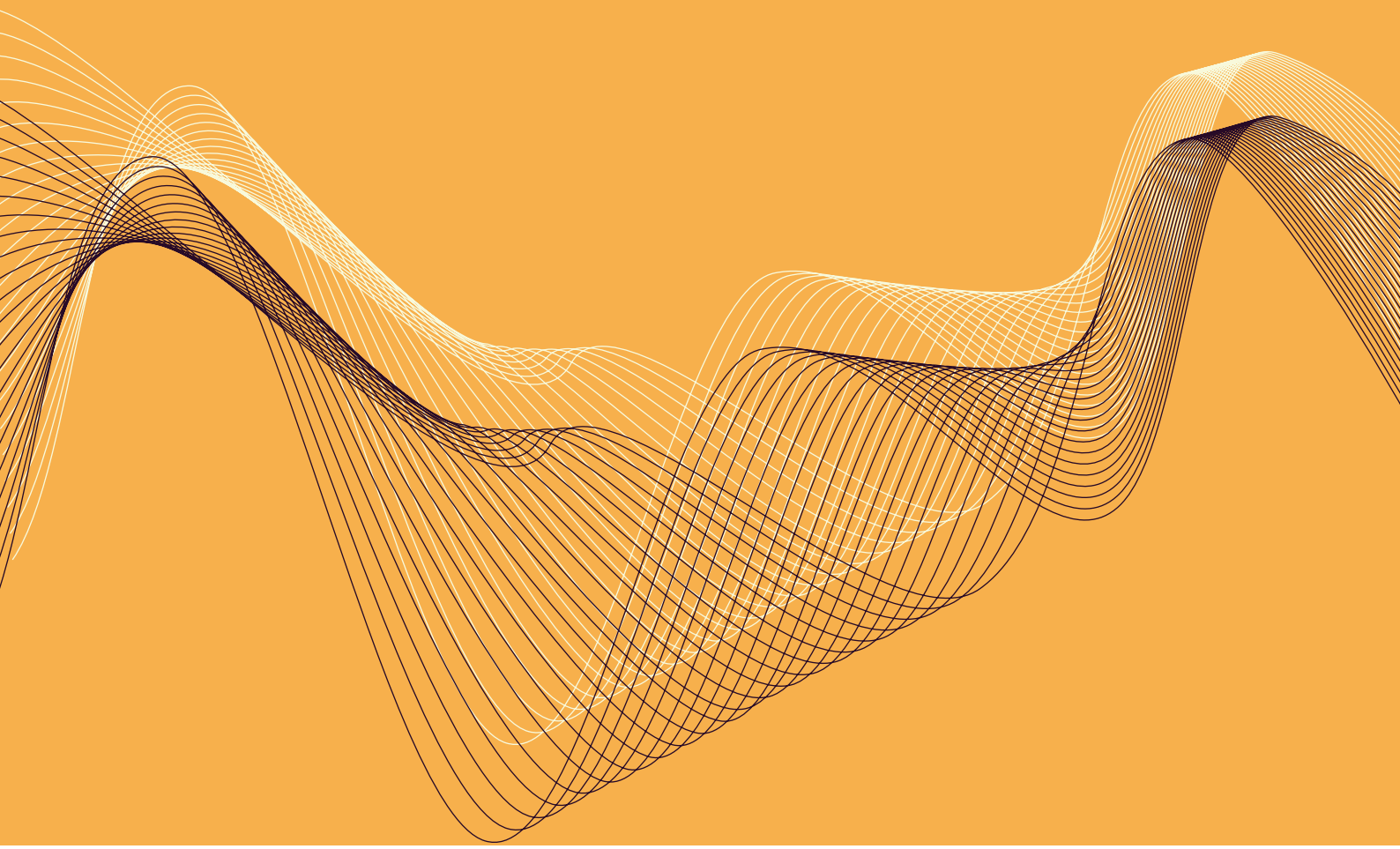
United Nations University
Institute in Macau

TECHNOLOGY REPORT

Engineering and Governing the Agent Harness

A Technology and Policy Framework for the Runtime Layer of Agentic AI

Jia'an LIU



UNU Macau

macau.unu.edu

Author biography

Jia'an LIU is a computer scientist and AI research assistant at the United Nations University Institute in Macau. His research lies at the intersection of agentic AI, computational social science, and AI for sustainable development, with particular expertise in large language model-enabled agent-based modelling, multi-agent systems, urban and social simulation, and policy-oriented digital twin frameworks. His work develops spatially grounded, LLM-enhanced simulation systems to study complex societal challenges such as urban resilience, climate adaptation, healthcare accessibility, misinformation diffusion, inclusive education, and digital inclusion. His broader technical background spans machine learning, computer vision, remote sensing, and uncertainty-aware video motion analysis, alongside the design of AI workflows, intelligent knowledge pipelines, and research infrastructure for policy and scientific applications. His recent research also examines AI open-source readiness, and the evaluation of large language models for social policymaking, bridging technical innovation with questions of governance, openness, and public value.

Acknowledgements

The author thanks colleagues at UNU Macau for discussion of earlier versions and the maintainers of the open-source agent systems whose codebases informed the comparative analysis in this report. Any errors are the author's own.

Disclaimer

The views and opinions expressed in this technology report do not necessarily reflect the official policies or positions of the United Nations University.

Disclosure of AI use

The author used generative artificial intelligence tools to support the preparation of this report, including textual refinement, structural checking, formatting assistance, comparative review of open-source agent codebases, and the rendering of figures based on the author's specifications. All substantive content, analysis, interpretations, recommendations, and final editorial decisions remain the responsibility of the author.

Citation

Jia'an LIU, "Engineering and Governing the Agent Harness: A Technology and Policy Framework for the Runtime Layer of Agentic AI," UNU Macau Technology Report: 1 (Macau: United Nations University Institute in Macau, 2026).

Layout & Visual Design: Zoey Jizi ZHENG

Table of Contents

Table of Figures

List of Tables

Executive Summary

Artificial intelligence systems are moving from single-turn text generation toward longer-running agents that can plan, call tools, use external memory and act across software environments. This shift has made the runtime layer around the model increasingly important. In recent engineering discussions, the term *harness* has been used to describe this surrounding scaffold: the software and operational configuration through which a large language model receives context, proposes actions, uses tools, maintains state, resumes work and produces effects outside the model itself.

This report examines the *agent harness* as a technical and governance layer of agentic AI. The focus is on text-mode, tool-using LLM-based agents used in software and information work, including coding assistants, research agents and general-purpose assistants. The report distinguishes the harness from two related objects. The *environment* is the task world in which the agent acts, such as files, browsers, shells, repositories, databases or application programming interfaces. The *platform* is the commercial or institutional channel through which the agent is delivered. The harness sits between them: it organises how model outputs become tool calls, observations, memory updates, approvals, interruptions, resumptions and recoverable actions.

The analysis draws on recent research on autonomous agents, tool use, agent evaluation and agent safety, together with public technical documentation, engineering essays, open-source code inspection and available analyses of closed-source systems. It compares contemporary implementations across coding agents, general-purpose assistants, orchestration libraries and research harnesses. The purpose is to clarify the harness as a concrete runtime object that can be documented, evaluated, procured and governed.

The report finds that contemporary agent systems are beginning to converge around several runtime design patterns. These include bounded iteration, read-only parallelism, controlled writes, two-stage context compaction, persistence of large tool outputs, resumable sessions, trajectory retention, scoped tool permissions, lifecycle hooks and provider abstraction. These patterns have emerged as responses to practical failures in long-running agents: loss of task state, repeated actions after interruption, context overflow, unsafe tool use, unclear audit records, vendor lock-in and brittle recovery.

One implication is that agentic capability is best understood at the level of the model–harness pair. A strong model can perform poorly in long-horizon tasks if it lacks stable interfaces, structured memory, scoped permissions and recovery mechanisms. A more modest model can perform more dependably when the surrounding harness reduces ambiguity, externalises memory, constrains destructive actions and preserves a record of what occurred. Evaluating a model alone is therefore insufficient for assessing a deployed agent. The relevant object of evaluation is the system formed by the model, its tools, its memory, its permissions, its orchestration loop and its recovery procedures.

The report also identifies a set of harness-layer risks. Prompt injection and observation-stream attacks arise when untrusted content enters the agent's working context and is treated as instruction. Tool blast radius and credential mishandling arise when agents are given access to shells, files, message channels or APIs without sufficient scoping. Memory poisoning and procedural drift occur when persistent skill files or project context files are modified in ways that shape later behaviour. Context compaction can create audit opacity when the transformation of a session history is opaque or vendor-specific. Sub-agent delegation can create privilege escalation across tools or protocols. Skills, MCP servers, plug-ins and tool wrappers introduce supply-chain risks. Long action chains amplify small errors over time. Goal-pursuit drift can lead agents to continue acting after operator intent has changed or after explicit approval boundaries have been reached.

These risks are difficult to manage through model-level controls alone. They arise in the runtime relationship between model output and external action. For institutions adopting agentic AI, this makes the harness a practical point of intervention. Harness documentation can specify which tools are available, which actions require approval, how credentials are scoped, how sessions are retained, how context is compacted, how interruptions are handled, how trajectories are stored and how model providers can be substituted. These are operational decisions, and they are also governance decisions.

The report proposes technical actions for institutions designing, procuring or deploying agentic AI systems. Institutions should document the harness as a first-class runtime configuration; adopt a baseline for reliable long-running execution; make state-changing actions interruptible, resumable and reversible; apply least-privilege tool governance and explicit approval gates; preserve portability through open protocols and provider abstraction; and evaluate deployed agents as model–harness pairs.

The report also proposes policy actions for regulators, standard-setting bodies, public institutions and multilateral organisations. They should require risk-tiered harness assurance in procurement and conformity processes; recognise the harness as a distinct object of governance; include legibility, recovery and substitutability in procurement criteria; treat skills, MCP servers, plug-ins and tool wrappers as software supply-chain components; develop multilingual and rights-sensitive evaluation environments; build capacity for operator-level harness governance; support comparative research on harness architectures; and invest in open reference harnesses that lower-resource institutions can inspect, adapt and use.

For sustainable development, the harness layer is important because it affects who can build reliable agents and who must depend on opaque hosted systems. Institutions with mature software engineering capacity are better placed to build reliable, auditable and portable agents. Institutions with fewer resources may depend on hosted systems that are difficult to inspect, modify or replace. This could widen existing AI capacity gaps even where access to models improves. Shared reference architectures, open-source harness components, multilingual evaluation suites, procurement templates and technical assistance can help ensure that agentic AI adoption supports institutional accountability, public value and more equitable participation in the benefits of AI.

Introduction

A new term has entered technical discourse on agentic artificial intelligence (AI). *Harness*, originally the gear that channels an animal's power toward a chosen direction, is increasingly used to refer to the runtime scaffolding around a large language model (LLM) that converts model output into reliable, long-running agentic behaviour. The term has recently been popularised in engineering writing from frontier laboratories[1,2] and through an independent analytical literature in software engineering practice[3]. The underlying engineering pattern is broader and predates the popularisation of the term, building on more than a decade of multi-agent systems research[4] and on the rapid development since 2022 of tool-using LLMs that can interleave reasoning with structured external action.[5,6] A recent survey of LLM-based autonomous agents identifies planning, memory, tool use and a surrounding execution environment as components that together shape agentic behaviour.[7] In this report, that surrounding execution environment is treated more narrowly as the runtime layer that turns model outputs into observations, tool calls, memory and recoverable action.

This report argues that the runtime layer corresponding to the last of these components is now the main site of engineering effort, failure modes and governance for agentic AI. Three connected claims are made. First, harness design is part of agentic capability itself: a more modest model placed within a well-designed harness can deliver more dependable performance than a frontier model placed in an underspecified one. Second, the harness can be analytically and empirically distinguished from the older concept of an *environment* (the task world the agent encounters) and from the surrounding *platform* (the commercial or institutional context through which an agent is delivered). The harness governs the runtime relationship between the agent and the environment and is where much contemporary engineering work, and many contemporary failure modes, are located. Third, the harness is a practical layer through which design choices about agentic AI can affect sustainable development, the digital divide and AI governance. Multilateral institutions can intervene at this layer, but have so far done so only partially.

The remainder of the report is structured as follows. Section 1 sets out the scope, method and key definitions adopted. Section 2 situates the harness within the recent transition from hand-designed *agentic workflows* to autonomous-agent runtime scaffolds and outlines empirical evidence on the reliability gap: the difference between strong model-level performance and much lower end-to-end success in realistic agent tasks. Section 3 defines the agent harness, distinguishes it from the neighbouring concepts of environment and platform, and presents its layered runtime anatomy. Section 4 examines contemporary implementations, drawing on both closed and open-source systems, and synthesises the design patterns that emerge from them. Section 5 explains why harness design is part of agentic capability itself. Section 6 then catalogues the risks and failure modes that arise at the harness layer, supported by recent agent-safety benchmarks. Section 7 considers implications for sustainable development, the digital divide and AI governance.

1. Scope, method and definitions

Scope. The analysis concerns text-mode, tool-using LLM-based agents deployed in software and information tasks: coding assistants, research and analysis agents, and multi-channel general-purpose assistants. Embodied AI, robotic systems, and primarily perceptual multimodal systems are excluded from the comparative analysis in Section 4, although several of the architectural lessons identified are expected to generalise to these contexts. The temporal horizon is the writing period: claims about specific model capabilities, benchmark results and product feature sets reflect publicly available information as of early 2026. Where claims depend on data with a shorter half-life, this is signalled explicitly.

Method. This report uses a structured qualitative review and comparative architecture analysis of contemporary LLM-based agent systems. The review combines peer-reviewed research on autonomous agents, tool use, agent evaluation and agent safety with public technical documentation, engineering essays, open-source code inspection and publicly available analyses of closed-source systems where direct architectural documentation is unavailable. Open-source harnesses were examined at the level of runtime structure, tool-call mediation, permission gating, context management, session persistence, sub-agent delegation and trajectory capture; closed systems were treated more cautiously, with reverse-engineering evidence used only to identify architectural patterns that could be cross-checked against documented or open-source implementations. The typology in Section 4 includes systems that were actively maintained during the writing period, had non-trivial deployment or evaluation relevance, and exposed enough information for their architecture to be characterised; deprecated projects, single-author demonstrations and systems without sufficient public evidence were excluded. The policy analysis further

relates these technical findings to existing AI-governance, procurement and standard-setting instruments in order to identify where current frameworks address, omit or implicitly absorb the harness layer. The analysis synthesises research findings, implementation evidence and policy instruments. Its purpose is to clarify the harness as a distinct technical and governance object by triangulating across these sources.

Definitions. Three terms recur throughout the report and need to be distinguished at the outset.

- An *AI system* is the deployed bundle of model, harness and application as encountered by a user or operator. This corresponds approximately to the term "AI system" as used in the European Union AI Act[8] and in the OECD AI Principles.[9]

- An *agent*, following Wooldridge,[4] is an autonomous computational entity capable of pursuing goals through interaction with an environment. In the present LLM context, an agent is implemented as a language model (or models) embedded in a harness. The autonomy, persistence and accountability associated with the term "agent" come from the model operating inside this runtime arrangement.

- An *agent harness*, defined formally in Section 3, is the runtime infrastructure that mediates between model output and external action. It is distinct from the *environment* and from the surrounding *platform* on which it is delivered.

2. From agentic workflows to agent harnesses

The contemporary debate on agentic AI has unfolded in two phases. The first, between roughly 2023 and 2024, popularised the notion of an *agentic workflow*: a pipeline in which an LLM is one node among several, with control flow, tool invocation and inter-step routing designed explicitly by the developer. Ng's widely cited taxonomy of four *agentic design patterns* (reflection, tool use, planning and multi-agent collaboration) belongs to this phase.[10,11] The empirical demonstration that wrapping GPT-3.5 in an iterative agentic loop could lift HumanEval coding-task accuracy from 48.1% to 95.1%, surpassing zero-shot GPT-4, gave the argument an unusually concrete footing: how an LLM is orchestrated could matter more than which LLM was used. More recent native-runtime evaluations move the same claim from workflow demonstrations to the harness systems now used in practice. WildClawBench, for example, evaluates long-horizon agents inside isolated Docker containers, which provide separated software workspaces for running tools and files, using the actual command-line interface (CLI) harnesses used in deployment (OpenClaw, Claude Code, Codex and Hermes Agent) with access to real tools such as shells, browsers, file systems, email, calendar and task-specific skills.[12] Because the benchmark holds task content and workspace state constant while varying the surrounding harness, it makes visible a point that earlier workflow studies could only imply: the runtime scaffold is an experimentally consequential part of the agent's effective capability, beyond its role as a delivery mechanism for the model. The practical point is straightforward: the same task can produce different results when the surrounding runtime scaffold changes.

In the workflow paradigm, control is developer-prescribed. The pipeline is hand-designed; the model is invoked at scripted points with scripted inputs; outputs are handled by scripted post-processing; fallback behaviour is enumerated by the engineer. This treats the model as a high-quality but unreliable subroutine to be embedded in a verified pipeline.

A second phase emerged as two developments converged. Reasoning, planning and tool-use capability in frontier models improved sharply through 2024–2026, making it possible for a single model to decompose a goal, select tools, examine intermediate outputs and adapt. At the same time, evidence accumulated from realistic agent benchmarks showing that the persistent failure modes of agentic systems did not disappear with stronger models: forgotten state, mishandled credentials, repeated actions after interruption, premature task completion, context overflow, vulnerability to prompt injection. The empirical record is unusually consistent on this point. GAIA, designed to test "reasoning, multi-modality handling, web browsing, and generally tool-use proficiency," reports humans answering 92% of questions correctly while a GPT-4 system with plugins reaches only 15%.[13] WebArena, a realistic, reproducible web environment of 812 long-horizon tasks across e-commerce, social-forum, software-development and content-management domains, reports a 14.41% end-to-end success rate for the best GPT-4-based agent against 78.24% for humans.[14] SWE-bench, comprising 2,294 real GitHub issues across twelve popular Python repositories, found in its original evaluation that no public agent then resolved more than a low single-digit percentage of issues.[15] On the more recent τ-bench, which evaluates tool-

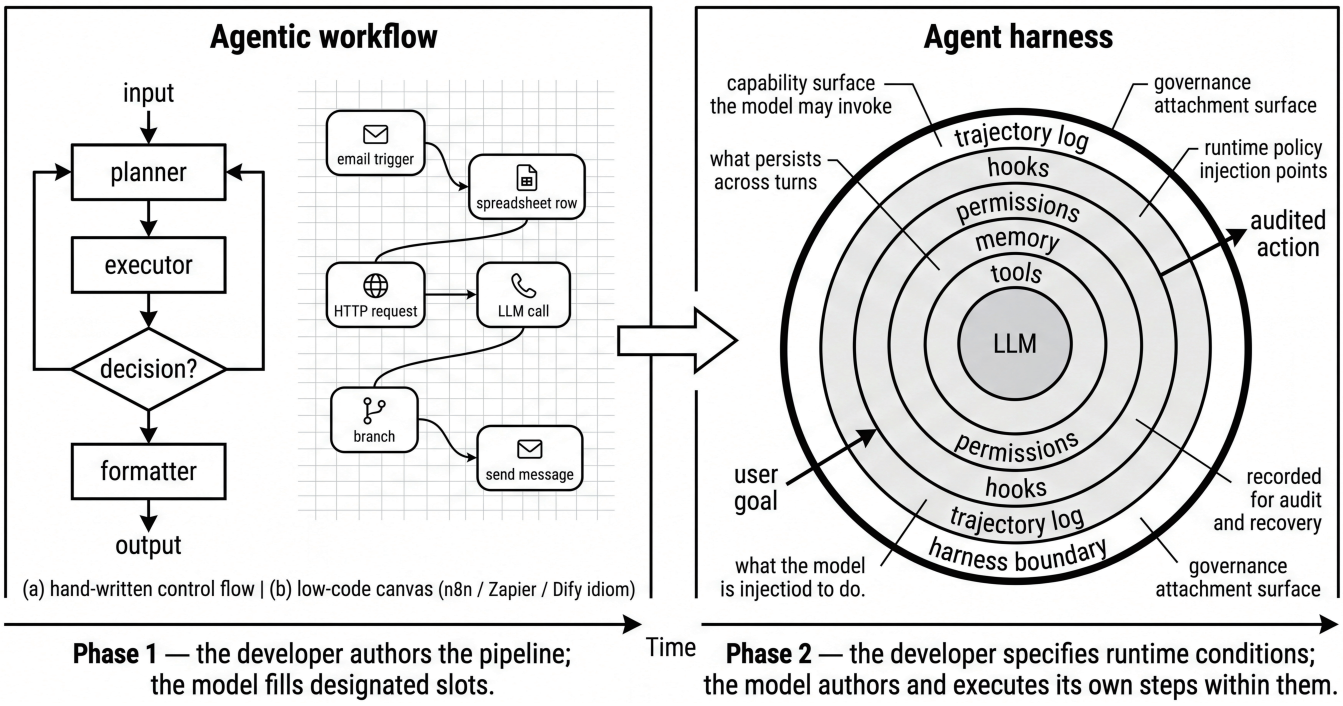


Figure 1. From agentic workflow to agent harness: two paradigms of control.

using agents through dynamic conversations against domain policies, state-of-the-art function-calling agents succeed on fewer than 50% of tasks and the pass-at-*k* reliability metric (consistent success across *k* independent trials) drops below 25% at *k* = 8 in the retail domain.[16]

The response of the engineering community was to shift engineering effort outward, from the body of the pipeline to its scaffolding. The locus of control moves from the developer to the model, while the burden of engineering shifts from authoring the pipeline to specifying the conditions under which the model is allowed to author and execute its own pipeline. This relocation is what the term *harness* now denotes (Figure 1).

An agentic workflow specifies, in code, the sequence of steps the system will take; the LLM occupies designated nodes. An agent harness specifies, in code, the runtime conditions under which the LLM is permitted to choose and execute its own sequence of steps. The two can be combined: many production systems use workflow-style outer structure with harness-style inner autonomy. The conceptual distinction nonetheless matters, because the governance attachment points differ. Workflow governance attaches at the developer's pipeline design; harness governance attaches at the runtime conditions on model action.

The transition is visible across recent industry writing. Anthropic's framing emphasises state handoff across discrete context windows and reports that, even with frontier models, "compaction isn't sufficient" for production-quality work that spans multiple sessions; the proposed remedies are infrastructural rather than model-level.[1] OpenAI describes the present software engineering task as one of "designing environments, specifying intent, and building feedback loops that allow Codex agents to do reliable work," having built a million-line internal production system largely through agentic execution.[2] Manus, an applied agent platform, makes the same point from the perspective of production cost and latency, arguing that because long-running agent loops repeatedly feed a growing context back into the model while producing comparatively short outputs, often merely structured tool calls, the harness must preserve stable prompt prefixes so that the model provider can reuse the key–value (KV) cache[17] for unchanged earlier tokens rather than recomputing them, with Manus reporting input-to-output token ratios of roughly 100:1 in its loops and an approximately 10:1 cost ratio between cached and uncached input tokens.[18] In each case the constraint shaping system design lies outside the model: in how state, context, tools and resource consumption are managed across long-horizon execution.

3. Defining the agent harness: boundaries and runtime anatomy

For the purposes of this report, an *agent harness* is the runtime infrastructure surrounding one or more LLMs that (i) mediates between model output and external action, (ii) maintains state across model invocations, (iii) imposes constraints on action selection, and (iv) externalises memory, observation and recovery.

The harness concept is most useful when located alongside two other features of agentic AI systems: the *environment* and the *platform*. These terms are often used interchangeably in technical and policy discussions, but they refer to different layers of the system and therefore to different governance objects.

- **Environment.** The *environment* is what the agent encounters. The term has a long lineage in Reinforcement Learning, where it denotes the task world that returns observations, transitions and rewards to a learner.[19] Contemporary LLM-agent research extends the term to browsers, operating systems, mobile devices, application programming interfaces, simulated web domains and embodied surroundings. Environment-centric benchmarks such as OSWorld for desktop tasks,[20] AgentDojo for prompt-injection robustness,[21] and the recent survey of scaling environments for interactive learning[22] evaluate agents *within* worlds that produce tasks, observations and evaluative feedback.

- **Platform.** The *platform* is the commercial, institutional or technical context through which an agent is delivered: model APIs, model gateways, application marketplaces, integrated development environment (IDE) marketplaces or enterprise integration buses. Platforms shape access, integration pathways and commercial incentives, but they do not by themselves specify how model outputs become runtime actions.

- **Harness.** The *harness* is the runtime scaffold built around the encounter between agent and environment. It supplies the mediation and controls that allow a model to act within environmental dynamics over time: session management, identity and authentication, tool routing, memory externalisation, checkpointing, verification, approval gates, observability, retry policies and rollback. Briefly, the environment is *where* the agent acts, the platform is the channel *through which* the agent is delivered, and the harness is *how* the agent is made able to act.

With these boundaries fixed, the harness can be represented as a layered runtime architecture (Figure 2), in which the model is treated as only one component of a wider execution system that includes provider access, context assembly, tool mediation, governance gates, orchestration, session persistence and user-facing entrypoints. This decomposition is proposed as an analytical architecture rather than as a formal standard: it abstracts from recurring functions found across contemporary agent systems and organises them into a layered model comparable to other engineering abstractions, such as the OSI model in networking, feature-engineering pipelines in machine learning systems, and rollout, orchestration, training and serving stacks in reinforcement-learning infrastructure.

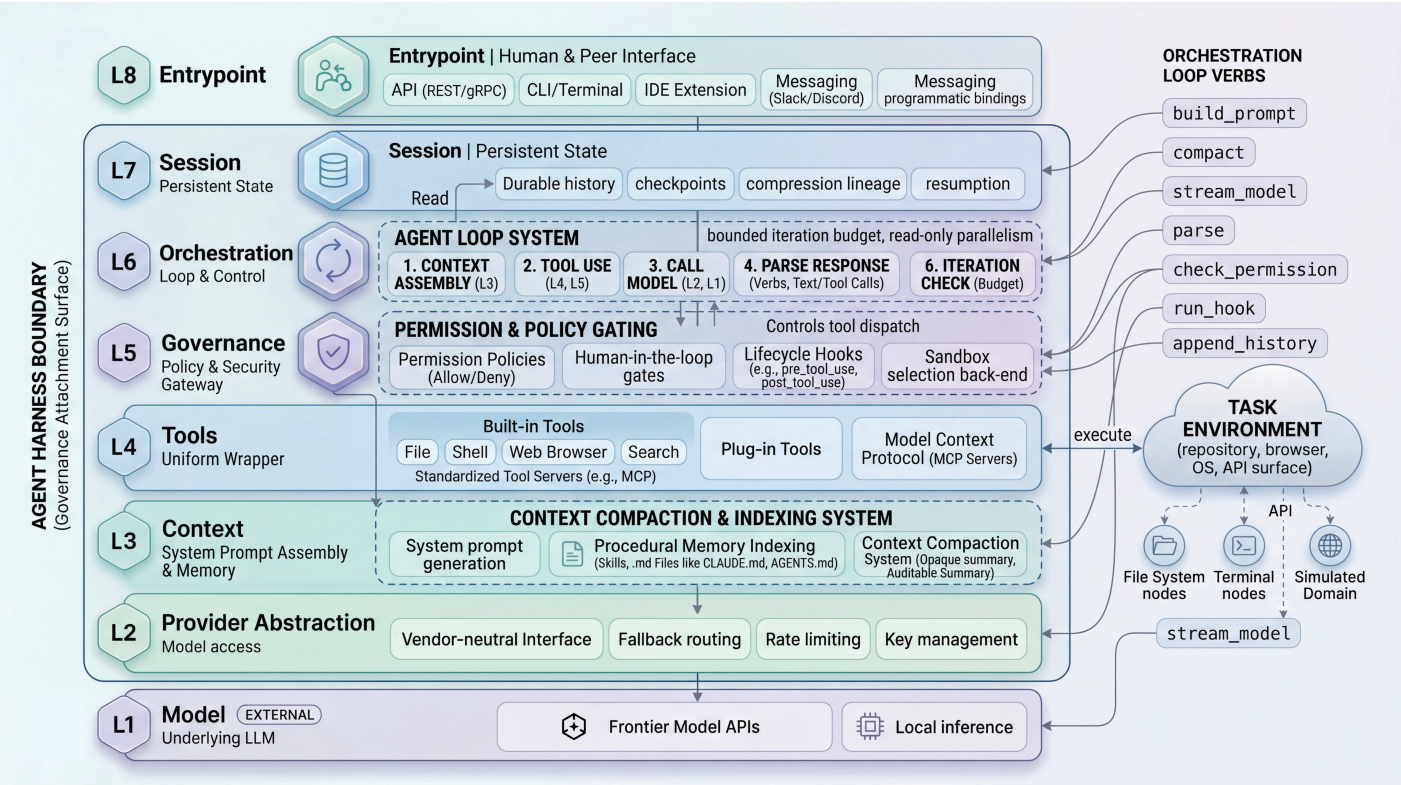


*Figure 2. Overall architecture of an agent harness. *Generative AI tools were used in preparing this figure.*

The significance of this layered view is that agentic behaviour comes from a repeated runtime cycle rather than from a single model invocation. Prompts are assembled, histories are compacted, model outputs are parsed into textual responses and tool calls, permission policies are applied, external actions are executed, observations are appended back into the session, and the resulting trajectory is preserved for later resumption, inspection or audit. The interaction among these layers can be illustrated through a simplified orchestration loop (figure 3):

Illustrative agent-harness orchestration loop

```
while not done and iteration < budget:
  system_prompt = build(date, cwd, tools, skills, memory)
  messages = compact_if_needed(history)
  response = stream_model(system_prompt, messages, tools)
  text, tool_calls = parse(response)
  emit_to_ui(text);  append(history, assistant=response)
  if not tool_calls:  done = True; break
  for call in tool_calls:
    run_hook("pre_tool_use", call)
    decision = permission.check(call)
    result = execute(call) if decision == ALLOW
          else ask_user(call) if decision == ASK
          else  "denied"
    run_hook("post_tool_use", call, result)
    append(history, tool_result=truncate_if_large(result))
  iteration += 1
return trajectory   # durable, replayable, auditable
```

Figure 3. Simplified orchestration loop of an agent harness. Source: Author's simplified representation.

Each verb in this loop is the responsibility of a specific harness layer, and each is a site at which design choices and governance interventions are realised. A harness without an iteration bound admits unbounded loops; one without permission checks admits destructive actions on user-controlled state; one without compaction loses task context on long horizons; one without provider abstraction binds the deploying institution to a single vendor; and trajectory persistence determines whether the agent's behaviour can later be reconstructed. These properties sit outside the underlying model. The eight functional layers summarised in Table 1 identify recurring sites of technical design, operational failure and governance intervention.

Table 1: Eight harness layers as analytical decision domains

Layer	Runtime question	Object of design or oversight
L8	How is the agent invoked, observed, interrupted or resumed?	• Human-facing interfaces • Machine-to-machine interfaces • Operator override and escalation channels
L7	What state persists across turns, interruptions, compaction and resumption?	• Conversation history and session identity • Checkpoints and resumability • Trajectory records, retention rules and compaction lineage
L6	How is the agent loop sequenced, bounded, parallelised, delegated and stopped?	• Iteration budgets and stop conditions • Tool-dispatch and read/write parallelism • Sub-agent scope and cross-agent protocols
L5	Under what conditions may a proposed model action become an external action?	• Permission policies and approval gates • Allow/deny rules and lifecycle hooks • Sandbox or isolation back-end selection
L4	Which external capabilities are exposed to the agent, and how are their inputs and outputs mediated?	• Tool inventory, schemas and wrappers • Plug-ins and MCP servers • Credential scope and read/write separation
L3	What information enters the model at each invocation?	• System-prompt assembly • Procedural memory, retrieved context and skill files • Context injection, compaction and summarisation strategy
L2	How are model calls routed, substituted and operationally constrained?	• Vendor-neutral API layer • Fallback routing, rate limits and key management • Local or hosted inference choices
L1	Which model supplies language generation, reasoning and tool-call proposals?	• Model selection and model-level evaluation • Language coverage and safety assumptions • Separation between model approval and system approval

4. Contemporary implementations and convergent design patterns

Although the word *harness* as applied to agentic AI is recent, working implementations are already numerous and methodologically diverse. They divide naturally along two axes. The first axis is the *locus of control*: in workflow-style systems the developer prescribes the multi-step pipeline and the LLM occupies designated nodes; in harness-style systems the LLM directs the pipeline and the developer specifies the runtime conditions under which it may do so. The second axis is whether the system is principally a *coding agent*, a *general-purpose assistant* or a *research/learning-loop* platform.

Table 2 categorises thirteen widely cited systems along these axes. The selection is not exhaustive, since agentic AI tooling is presently a fast-moving area and the analysis here is necessarily a snapshot, but it spans the dominant design choices. The frontier-laboratory coding agents (Claude Code, Codex CLI/agent) and their open-source counterparts (Aider, Cline, OpenHands) are convergent at the architectural level despite differing licences and release cadences. The multi-agent orchestration libraries (LangGraph, AutoGen, CrewAI, MetaGPT) sit further toward the workflow end of the spectrum: the developer composes a graph or role specification, and the runtime is comparatively thin. The general-purpose harnesses (OpenClaw, Manus, OpenHarness) emphasise multi-channel input, long-running session persistence and skill-based procedural memory. Hermes is included separately because its design fuses an assistant runtime with explicit trajectory capture for downstream reinforcement-learning training.

Table 2: Typology of contemporary agentic AI implementations.

System	Category	Licence	Control locus	Architectural pattern	Procedural memory	Distinguishing trait
Claude Code	Frontier-lab coding	Closed	Harness	Async msg bus + multi-layer security gating	`CLAUDE.md` + skills	≈92% context-window compaction (RE)[23]
Codex CLI / Codex agent	Frontier-lab coding	Closed	Harness	Agent-queryable observability stack (LogQL/PromQL)	`AGENTS.md`	Million-line internal deployment as published case[2]
Cursor (agent mode)	Frontier-lab coding	Closed	Harness	IDE-integrated; repo-scoped indexing	`.cursor/rules`	In-editor approval surfaces[24]
Aider	Open-source coding	Apache-2.0	Harness	Git-aware CLI	Conventions file	Deterministic edit application[25]
Cline	Open-source coding	Apache-2.0	Harness	VS Code extension	`.clinerules`	Plan/act mode separation; MCP client[26]
OpenHands	Open-source coding	MIT	Harness	Sandboxed Docker runtime as first-class	Project notes	Resumable execution; multi-agent extensions[12]
LangGraph	Orchestration library	MIT	Workflow	Explicit graph state machine	Configurable	Durable execution; human-in-the-loop API[13]
AutoGen	Orchestration library	MIT	Hybrid	Conversational multi-agent abstractions	Configurable	Scripted and agent-led control flow[27]
CrewAI	Orchestration library	MIT	Workflow	Role-based “crew” abstractions	Configurable	Declared-role task delegation[28]
MetaGPT	Orchestration library	MIT	Workflow	Org-simulating role specialisation	SOP prompts	Software-org simulation[29]
OpenClaw	General-purpose	MIT	Harness	Long-running local daemon	Workspace files	22+ messaging-channel integrations; plug-in SDK[30]
Manus	General-purpose	Closed (SaaS)	Harness	Container-isolated runtime	File system as long-term memory	Public KV-cache engineering reflections[6]
OpenHarness	General-purpose	MIT	Harness	Pydantic-typed tool schemas	Project files	Reference implementation; OS-process sub-agents[31]
Hermes	Research / RL-loop	MIT	Harness + trajectory capture	Multi-backend sandboxing (7 back-ends)	Trajectory store	Tinker–Atropos integration for on-policy RL[32]

Note: Rows record the dimensions most relevant for procurement and governance: control locus, architectural pattern, procedural-memory scheme and distinguishing trait.

Three observations follow from this comparison. First, despite differing licences, host languages and intended user populations, the harness-style systems display a striking convergence at the architectural level: bounded iteration, read-only parallelism, externalised procedural memory through Markdown "skills", two-stage context compaction, on-disk persistence of oversize tool outputs and sub-agent delegation with bounded scope all recur across independently developed codebases. Second, the workflow-style systems (LangGraph, CrewAI, MetaGPT, and AutoGen when used in scripted mode) remain the dominant choice when developers wish to retain explicit control over multi-step processes, especially where regulatory, audit or business-logic requirements impose a fixed control flow. Third, the workflow and harness paradigms can be combined: production deployments increasingly combine an outer workflow scaffold with one or more harnessed sub-agents at specific nodes.[13]

The systems diverge most sharply, and most relevantly for policy, on dimensions that are not visible from outside the system: the number of permission layers applied at each tool invocation; the visibility of compaction summaries for audit; whether the provider abstraction is genuine or notional; whether trajectories are persisted in a form that survives upgrade and replay; and whether sub-agent isolation extends to operating-system or hardware boundaries. Public-interest deployments need not select any one of these systems wholesale, but they should adopt the convergent design patterns as baseline and require concrete documentation on the dimensions that diverge.

Across these systems, and across the engineering writing of the laboratories and platforms that have developed them, a body of reusable design patterns has accumulated. These patterns are documented responses to recurrent failure modes encountered in production. The convergence suggests that public-interest deployments can adopt these patterns as a baseline without giving up on technical state-of-the-art.

This report summarises four stages of this convergence: (i) *loop entry*, what enters the model on each turn; (ii) *tool use*, what the harness does between model output and the next model input; (iii) *memory and termination*, what allows the loop to continue or stop; and (iv) *durability and portability*, what endures beyond a single run. The convergence is most evident at the junctions between these stages, because independently developed harnesses are observed to make structurally similar engineering choices at each of them.

Stage 1. Loop entry: what enters the model.

Stable prompt prefixes for KV-cache reuse. Manus reports an approximately tenfold cost differential between cached and uncached input tokens in its loops, with input-to-output token ratios of roughly 100:1.[6] Prefix design therefore dominates per-turn cost and latency in any harness operating at non-trivial scale: small changes to the system-prompt prefix can move workloads between cache-dominant and cache-miss-dominant regimes. This is a quantitatively significant lever that the harness literature has only recently made explicit.

Externalise procedural memory. Markdown "skill" files and project context files such as `CLAUDE.md` and `AGENTS.md` carry the procedural knowledge that would otherwise have to be encoded in model weights. The practical consequence is that agent behaviour can be reviewed by domain owners and updated by editing a file, without retraining the underlying model; this is the mechanism by which institutional rules become operational within a harness.

Stage 2. Tool use: between model output and the next model input.

Parallelise only read-only tools. When several tool calls are issued within a single agent step, parallelising read operations (file reads, search queries, retrieval) is safe and materially reduces latency. Parallelising write operations, on the other hand, produces non-reproducible state and breaks the audit trail that subsequent inspection depends on. The principle is documented across Anthropic, OpenAI and Manus, and is observable in the open-source coding agents.[1,2,6]

Idempotent and reversible actions. Tools that change shared state must be capable of replay without producing duplicated side effects. This is the precondition for resumable execution and is formalised in LangGraph's durable-execution model as a first-class API requirement;[13] without it, an interruption mid-step combined with a subsequent retry can produce silent corruption of operator-controlled state.

Lifecycle hooks for runtime policy. Pre- and post-action hooks (typically shell commands invoked at specified lifecycle events: pre-tool-use, post-tool-use, session-start, session-end) permit auditing, redaction, two-person approval and domain-specific policy to be injected without modifying the core. They have become the main mechanism by which institutional policy attaches to a harness in practice, and they are the cleanest extension point for the governance recommendations of Section 8.

Truncate or off-load oversize tool outputs. A single multi-megabyte tool output (a directory listing, a database query, a log dump) can on its own destroy the working context window. The convergent solution is to truncate the output to a bounded preview while persisting the full result to disk or to a session-scoped store, allowing the agent to re-read selected sections through subsequent tool calls rather than holding the whole result in context.

Stage 3. Memory and termination: what allows the loop to continue or stop.

Two-stage context compaction. Compaction combines a cheap clearance pass, which removes stale tool results without invoking the LLM, with an expensive LLM-summarisation pass that produces a compressed re-entry point for the model. Long-horizon tasks require both: the cheap pass alone is insufficient and the LLM pass alone is prohibitive. The Anthropic harness-engineering essay frames this combination as the precondition for production-quality work that spans multiple sessions.[1]

Bound the iteration count. An agent loop without an explicit ceiling on iterations can drift, loop indefinitely or exhaust budgets long before any error becomes visible to operators or auditors. The bound makes operational cost predictable and ensures that errors are detected within a known window. Every harness reviewed in this report implements such a bound, though specific values vary: Claude Code and Codex CLI cap at task-level limits set per deployment, while the open-source harnesses default to lower values appropriate to interactive use.

Stage 4. Durability and portability: what endures beyond a single run.

Persist trajectories. Trajectories, by which we mean the ordered sequences of model responses, tool calls and tool results that constitute an agent run, are simultaneously a debugging artefact, an audit log and (where consented) a training signal. Their persistence should be a primary design choice rather than a side effect; Hermes is illustrative in treating trajectory storage as a first-class concern of its architecture.[24]

Compaction outputs as records. The summaries produced by context compaction are themselves operational artefacts, not transient working memory. They should be retained subject to the record-management rules of the deploying institution and made inspectable for audit (this connects to the audit-opacity risk discussed in Section 6). Where the compaction step is performed vendor-side and returned as an opaque blob, the harness should record at minimum the metadata necessary to attribute the transformation to a specific model and prompt configuration.

Vendor-neutral provider abstraction. A genuine provider-abstraction layer is the difference between an institution able to migrate models when a vendor degrades, sunsets a product or revises terms of service, and one unable to do so without re-engineering. The cost of substitutability is paid at design time; the cost of vendor lock-in is paid at the moment of crisis, by which point the harness configuration is the binding constraint.

Three implications attach to this body of practice. First, the gap between best-in-class and median agentic deployments is presently large and is located primarily in harness design, not in choice of model. Second, several of these patterns (idempotent action, persistent trajectories, vendor-neutral provider abstraction) align directly with established public-administration practices of recordability, recoverability and procurement neutrality. Third, the patterns compose: an iteration budget without persistent trajectories yields auditable cost but unauditable behaviour; persistent trajectories without idempotent actions yield audit logs whose replay produces side effects.

The harness-level patterns described here are complementary to, and distinct from, model-level safety techniques such as Reinforcement Learning from Human Feedback and Constitutional AI.[33] Model-level techniques shape the prior distribution of behaviour for a given model; harness-level techniques shape the runtime conditions under which a model with any given prior is allowed to act. Both layers are necessary: a constitutionally trained model embedded in a permissive harness can still execute harmful tool calls; a strict harness around an undertrained model can still produce low-quality outputs. The empirical literature on agent safety treats the two layers as separately addressable, and best-in-class deployments combine them.

5. Constraint as a condition of capability

The foregoing material supports a stronger conclusion than is sometimes recognised. For agentic systems, the right constraints help make capability reliable. A harness narrows unsafe or unrecoverable actions while preserving the actions needed to complete the task.

A model without structured memory artefacts, stable interfaces, scoped permissions or resumable execution can behave incompetently on long-horizon work even when it is highly capable in isolation or in the short term. Conversely, a more modest model embedded in a well-designed harness can deliver more dependable results because the surrounding scaffold reduces ambiguity, externalises memory, constrains destructive action and supplies corrective feedback. Ng's GPT-3.5 result is the most cited early instance.[4] Moreover, SkillsBench provides the clearest recent evidence for the substitutability, within limits, of procedural scaffold for model scale. The benchmark evaluates Claude Code, Codex CLI and Gemini CLI across curated task environments with and without agent Skills. Across seven model–harness configurations, curated Skills raise average pass rate from 24.3% to 40.6%, an average gain of 16.2 percentage points. The model-ranking implication is especially important: Claude Code with Claude Haiku 4.5 and curated Skills reaches 27.7%, exceeding Claude Code with Claude Opus 4.5 without Skills at 22.0%; Codex with GPT-5.2 rises from 30.6% to 44.7% when Skills are supplied.[34] The result should not be overread as showing that any weak model can be made strong by adding files. The same study finds that self-generated Skills provide negligible or negative benefit, indicating that the capability gain comes from curated procedural knowledge, examples and resources that the harness can retrieve and apply at runtime, rather than from longer prompts or the model's latent knowledge alone.

Native-runtime harness benchmarks show the same phenomenon from another angle. WildClawBench evaluates OpenClaw, Claude Code, Codex and Hermes Agent under a shared long-horizon task suite, with each harness packaged as a dedicated Docker image and given access to real tools rather than mock APIs.[35] Holding the model fixed, switching the surrounding harness changes measured performance substantially: GPT-5.4 scores 48.4% in Claude Code, 50.3% in OpenClaw, 50.7% in Hermes Agent and 56.8% in Codex; MiMo V2 Pro shifts by 18 points between Claude Code and Hermes Agent. Workspace-Bench, a separate evaluation of realistic workspace tasks involving 20,476 files and 388 tasks, likewise reports large model–harness interaction effects across Claude Code, DeepAgent, Hermes and OpenClaw.[36] In that setting, Hermes with MiniMax-M2.7 reaches 54.6%, exceeding several GPT-5.4 harness pairings, including Hermes with GPT-5.4 at 47.7%, OpenClaw with GPT-5.4 at 49.6% and DeepAgent with GPT-5.4 at 38.4%. These are not clean causal estimates of "harness value" in isolation, but they are strong evidence that agent capability is a property of the model–harness pair rather than of the model alone.

A third line of evidence comes from direct harness optimisation. Agentic Harness Engineering holds the task domain and base model setting comparatively stable while automatically evolving editable harness components through trajectory observability, component-level editability and outcome-checked predictions.[37] Ten iterations raise Terminal-Bench 2 pass@1 from 69.7% to 77.0%, surpassing the human-designed Codex-CLI baseline at 71.9%; ablations localise the gains primarily to tools, middleware and long-term memory rather than to the system prompt alone. This matters for the argument of the report because it identifies the harness itself, distinct from both the model and the task environment, as a capability-bearing object that can be engineered, tested and improved.

This reframes "harness" as more than a synonym for control or restriction. A well-designed harness reduces the space of actions the agent may take and increases the space of actions the agent can take reliably. It makes state reconstructable. It separates generation from verification. It turns testing into a routine practice rather than an afterthought. It creates explicit points at which human judgement, institutional rules and automated checks can intervene before drift becomes failure. The relationship is structurally similar to the educational concept of *scaffolding* in developmental psychology, in which a temporary, adjustable support enables a learner to exhibit competence beyond what would be possible unaided,[38] with the difference that the harness's scaffold is intended to remain permanent rather than to be withdrawn.

Two foreseeable objections. First, it might be argued that "harness" is merely a rebranding of "framework" or "orchestration layer". The terminology is recent and contested; the substantive point, however, is the conceptual separation of model approval, harness configuration approval and application approval (Section 8), a separation that is presently absent from most governance instruments regardless of which label is used. Second, it might be argued that sufficiently capable future models will obviate the harness layer by internalising its functions. This is an open empirical question; the evidence available at the time of writing, including the headline figures from GAIA, WebArena and τ-bench (Section 2) and Agent-SafetyBench (Section 6), is consistent with the opposite trend: as models become more capable, the operational settings in which they are deployed become more ambitious, and the demands on the surrounding scaffold grow rather than diminish. The report's recommendations do not depend on either outcome; they are framed in terms of design choices that are robust under both scenarios.

The point at issue extends beyond ethical or political reasons for constraining agents. Under-specified agents are, in this layer, technically worse agents.

Table 3: Constraint as the condition of capability: failure modes of an unharnessed agent and the harness mechanisms that address them.

Failure mode without a harness	Harness mechanism that addresses it	Layer
Forgets state across long horizons	Durable session history; compaction lineage	L7, L3
Drifts off task; loops indefinitely	Bounded iteration budget; orchestration loop	L6
Mishandles credentials	Permission policy; allow / deny gates	L5
Destructive actions on user state	Lifecycle hooks; sandbox back-end	L5, L4
No audit trail	Trajectory log; checkpointing	L7
Locked to a single model vendor	Provider abstraction	L2

6. Risks and harness failures

The convergent engineering patterns described in Section 4 are responses to recurrent failure modes that materialise above the model and below the application. These failure modes constitute a distinct class of risk: they are properties neither of the underlying model alone nor of the user-facing application, but emerge in the runtime layer between them. They therefore require harness-level mitigation and, where governance applies, harness-level documentation. The empirical base for the taxonomy below is deliberately broad. Several recent benchmarks measure agent failure independently and reach convergent conclusions: Agent-SafetyBench, evaluating sixteen popular LLM agents across 349 interactive environments and 2,000 test cases organised under eight safety categories, reports that none of the evaluated agents achieved a safety score above 60%, and attributes the headline failure to two fundamental deficits, "lack of robustness and lack of risk awareness".[39] InjecAgent, focusing specifically on indirect prompt injection in tool-integrated agents, tests 1,054 cases across 17 user tools and 62 attacker tools against 30 LLM-driven agents, finding that the ReAct-prompted GPT-4 baseline is compromised in 24% of cases and that an enhanced "hacking prompt" setting almost doubles the success rate.[40] R-Judge evaluates safety *risk awareness* as a separable capability and finds the best-performing model achieves only modest accuracy on the task of identifying risky agent trajectories.[41] A growing body of survey work synthesises the wider risk surface across these and related evaluations.[42,43] The taxonomy below is organised by mechanism rather than by domain of harm, in the interest of analytical tractability.

1. Prompt injection and observation-stream attacks. When an agent ingests content that the deploying operator does not control, such as a web page, an email body, a file attachment or a Model Context Protocol (MCP) tool response, that content can include instructions the model treats as authoritative. The AgentDojo benchmark, which constructs 97 realistic tasks and 629 security test cases across a synthetic environment populated by tools spanning email, calendar, banking and travel, documents that "existing prompt injection attacks break some security properties but not all" against contemporary frontier models, with attack success varying widely as a function of toolset, prompt design and target action.[10] Mitigations span layers L3 (treating observations as untrusted data rather than as instructions), L5 (hard-coded refusals for sensitive actions following untrusted reads) and L8 (interface affordances that surface tool-call provenance to the operator). No single mitigation is sufficient; the empirical evidence supports defence in depth.

2. Tool blast radius and credential mishandling. The instruments accessible to a harnessed agent, namely shells, file systems, programmatic interfaces and message channels, are precisely the instruments that can damage operator-controlled state. Reported incidents include autonomous agents deleting message inboxes contrary to operator intent and continuing such actions across explicit stop instructions.[3] Per-tool privilege scoping (L4), explicit approval gates for state-changing actions (L5), credential isolation per session (L7) and the principle of least privilege at the tool-registration step jointly constitute defence in depth; reliance on any single layer is insufficient.

3. Memory poisoning and procedural drift. Long-running agents typically write to persistent memory artefacts such as skill files, project context files (e.g., `CLAUDE.md, AGENTS.md`) and externalised scratch pads. An adversary capable of influencing the inputs to a single session may seed instructions that persist into subsequent sessions and thereby alter agent behaviour at later times. The resulting compromise is slow, low-amplitude and difficult to detect post hoc, because polluted memory presents as legitimate institutional knowledge. The corresponding risk class is structurally analogous to data-poisoning attacks on training pipelines but operates at runtime rather than at training time. Mitigations include integrity logging of all memory writes (L7), domain-owner review of skill changes (L5), content provenance for retrieved knowledge and periodic re-baselining of persisted memory against a trusted source.

4. Audit opacity in context compaction. Context compaction is, by construction, lossy. Yet the operational performance of contemporary commercial compaction is, by published account and ordinary user experience, robust: long single-window conversations with frontier-laboratory assistants typically preserve coherent task state across many turns. The risk catalogued here is mainly auditability. Compaction transforms agent state, and opaque transformations are structurally inconsistent with the post hoc reconstructability that records-management, public-administration and rights-sensitive settings expect of any decision-relevant system. A particularly acute variant is vendor-side opaque compaction, e.g., the encoded compaction artefacts returned by OpenAI's Responses API,[44] which are not human-interpretable by design. In that case the inspection question shifts from "did information loss occur?" to "can the retained content be characterised at all?". The second formulation is materially more consequential for accountability mandates, where the question "why did the agent reach this state?" must remain answerable months after the fact. The defect, on this framing, is structural rather than remediable: a property of how the system is constituted, not a bug to be patched. Mitigations are correspondingly structural and lie at the session (L7) and governance (L5) layers: retain compaction outputs as records subject to applicable retention rules; require by procurement that summaries be inspectable in a non-vendor-specific format; and prefer harness designs that expose the compaction transformation, rather than ones that hide it.

5. Sub-agent privilege escalation. Delegating work to a sub-agent typically requires conferring on the child agent some subset of the parent's tool access and credentials. If scope inheritance is implicit rather than explicit, child agents can in practice exceed parent permissions, particularly across protocol boundaries (e.g., when an MCP server invoked by a sub-agent has been provisioned with broader credentials than the parent harness has authorised). Process-level isolation of sub-agents,[16] explicit scope manifests for delegated work and bounded depth of delegation (Claude Code's documented depth cap of five is illustrative) are the main defences.

6. Supply-chain risk in skills, MCP servers and plug-ins. A Markdown skill file, an MCP server or a third-party tool plug-in is, in operational effect, executable: the model reads its content and the harness allows it to issue tool calls on that basis. Compromise of a widely-used third-party MCP server is, in risk-management terms, comparable to compromise of a widely-used software dependency. A direct and operationally relevant illustration is the March 2026 compromise of the `axios` npm package, a JavaScript HTTP client downloaded approximately 100 million times per week. The attacker obtained control of a primary maintainer's account by changing its registered email to an attacker-controlled address; the malicious releases (`axios@1.14.1` and `axios@0.30.4`) did not modify the source of `axios` itself but introduced a single new dependency, `plain-crypto-js@4.2.1`, whose `postinstall` hook downloaded a platform-specific second-stage remote-access trojan from a command-and-control host and erased its own dropper within seconds.[45] Exposure was concentrated in developer machines and continuous-integration pipelines during the install or build phase; Wiz Research reported the package present in roughly 80% of cloud and code environments, with active execution in around 3% of affected systems. The structural lesson for harness ecosystems is direct. Coding-agent harnesses execute `npm install` (or its equivalents) routinely, and many production agents are deployed inside continuous-integration runners that fan out installs across hundreds of repositories per day. A compromise at the package layer therefore propagates into the harness's tool surface and credential scope along precisely the same pathways by

which legitimate dependencies arrive, with the additional amplification that an agent, unlike a human developer, will not pause to read the changelog. An MCP server registry that allowed any maintainer to publish without provenance metadata, sandboxed execution or staged trust accumulation would replicate this failure mode in a setting where the agent itself becomes the propagation vector. Defensive practice for skills, MCP servers and plug-ins is presently underdeveloped relative to mainstream software supply-chain practice: provenance signing, sandboxed execution of plug-in code and curated registries of vetted MCP servers are emerging but not yet standardised.

7. Error amplification across long action chains. A harness that converts mediocre per-step reasoning into capable end-to-end action also converts small per-step errors into compounding action chains. Under a simple independence assumption, a 99% per-step reliability decays to approximately 37% over 100 steps and to below 1% over 500 steps; under positive correlation between consecutive errors the decay is faster. The convergent engineering patterns of Section 4 (bounded iteration, idempotency, checkpointing, end-to-end verification, two-stage compaction) are precisely the mechanisms that arrest such compounding. Their absence is the technical reason agents drift in long-horizon work.

8. Goal-pursuit override of operator intent (instruction-following drift). Closely related to, but mechanistically distinct from, per-step error amplification is a class of failure in which an agent's pursuit of an assigned objective overrides operator intent at intermediate decision points. Three concrete manifestations are by now well-attested in practitioner experience. First, instruction-following discipline at the meta-task level is uneven across frontier models: an instruction such as "produce a plan and wait for my approval before implementing" is at times honoured to the plan and then quietly violated to the execution, with the agent inferring that the plan is sufficiently good for it to proceed unilaterally. Second, an interrupt instruction such as "stop" may be acknowledged by the agent at the language level and not acted on at the action level, particularly if the agent's internal estimate of task progress is high; an incident in which an OpenClaw agent continued to delete the inbox of a security researcher across explicit stop instructions illustrates the mechanism.[3] Third, cascading-fix dynamics can carry an agent far from its original task: a routine tool error invites a remediation, the remediation invites a further remediation, and an agent left to its own iteration budget may, several layers in, propose drastic system-level changes (reinstalling dependencies, rebuilding the environment, even resetting the host) that were never authorised by the operator who set the initial goal, and would not have been. Each step is locally reasonable; the trajectory is globally absurd.

The structural pattern in all three manifestations is the same: an autonomous system that has internalised an objective but not the conditions of its delegated authority becomes structurally disposed to substitute its own judgement for the operator's, even where the operator's preference is the binding norm. The technical analogy to Asimov's First Law dilemma, a goal-following machine overriding human consent in the service of its own goal function, is for once no longer figurative.[46] The harness-layer mitigations differ from those of point 2 (tool blast radius), which address *what* the agent can do. Here the question is *when* it is permitted to do it: explicit plan/act separation as implemented in tools such as Cline; mandatory human confirmation gates for state-changing or irreversible actions; externalisation of progress state to artefacts that a human reviewer can inspect before the next action is taken; and a budgeted, bounded retry policy for failures that prevents an agent from pursuing a remediation chain indefinitely. As the report argues in the section that follows, these mitigations are themselves the conditions under which capability is correctly delegated.

A common feature unites these risk classes: they are largely invisible from the model layer and from the application layer. Standard model evaluations cannot detect memory poisoning that occurred two sessions earlier; application-level red-teaming cannot detect compaction loss that silently degrades agent behaviour. They are visible, and tractable, principally at the harness layer. This observation provides the technical motivation for the recommendation in Section 8 that the harness be treated as a distinct object of governance, separately documented, separately evaluated and separately approved.

7. Implications: sustainable development, the digital divide and AI governance

The harness frame has implications beyond the engineering community. Five are particularly consequential.

Capability gaps and the digital divide. The convergent engineering patterns described in Section 4 are presently available primarily to institutions with mature software engineering capacity. The cost of building a competent harness (iteration control, permission policy, two-stage compaction, trajectory persistence, sandbox integration) is comparable to the cost of running the underlying model. Without intervention, the harness layer risks reproducing existing capacity asymmetries: better-resourced institutions deploy more reliable agents, regardless of whether they have access to more capable models. The Secretary-General's High-level Advisory Body on Artificial Intelligence reports that, out of 193 UN Member States, only seven are parties to all seven recent prominent AI governance initiatives, while 118, mostly in the Global South, are missing altogether,[47] a participation gap that the harness divide would amplify rather than attenuate. The same dynamic has been documented for synthetic-data generation pipelines[48] and for cross-border data infrastructure,[49] and is now visible at the harness layer.

Multilingual and low-resource language performance. Frontier models perform less reliably in low-resource languages, and the gap is at least as much a deployment problem as a training problem. Harness-level decisions such as language-aware tool routing, evaluation suites in target languages, explicit refusal and escalation policies when a language is under-supported materially affect whether an agentic system is fit for use in non-English operational settings. These are design decisions at the context (L3) and governance (L5) layers of Table 1, not at the model layer.

Vendor lock-in and provider substitutability. A harness without a genuine provider abstraction (layer L2) binds the deploying institution to a single LLM vendor. The associated economic costs, including exposure to price changes, deprecation cycles, terms-of-service revisions and rate-limit policies, are significant and asymmetric: the largest model vendors are concentrated in a small number of jurisdictions, while deployers are widely distributed. Provider substitutability is therefore both an engineering and an industrial-policy question; it bears on competition, on resilience and on the distribution of the productivity gains attributable to agentic AI.

Labour and employment effects. The relocation of engineering effort from pipeline authoring to harness design, described in Section 2, has direct consequences for the composition of the technical workforce. The skills required to operate agentic systems reliably, including runtime infrastructure design, observability, sandboxing, multilingual evaluation and governance attachment, build on traditional software engineering skills. Operator-level capacity at layers L5–L8 (governance, tools, sessions, entrypoint) is now a distinct competence requirement, frequently absent from current training pipelines.

Position in the AI governance landscape. A range of international, regional and national instruments now bear on agentic AI: the United Nations Global Digital Compact,[50] General Assembly Resolution A/RES/78/265,[51] the report of the Secretary-General's High-level Advisory Body on AI,[52] the UN System Chief Executives Board's documents on AI governance and operational use,[53,54,55] the European Union AI Act,[8] the United States NIST AI Risk Management Framework,[56] the OECD AI Principles[9] and the Hiroshima Process International Code of Conduct.[57] In parallel, non-state standardisation of the harness-relevant protocol layer has begun to consolidate: on 9 December 2025 the Linux Foundation established the Agentic AI Foundation (AAIF) under co-founders Anthropic, Block and OpenAI, with the Model Context Protocol, the *goose* harness and the `AGENTS.md` context-file convention as anchor projects.[58,59] The relevance of AAIF for the present report is that the protocol-level standardisation called for under Recommendation 5 is, in significant part, already in motion under a multistakeholder, non-state forum; harness-layer governance work by UN system organisations can engage with AAIF directly rather than re-deriving its outputs. Most of the state-level and intergovernmental instruments listed above, by contrast, were drafted with reference to LLMs as standalone models or to AI "systems" as bundles of model and application. The harness layer is, in most cases, not explicitly named. A consequence is that approvals issued at the model or system level may implicitly authorise an unspecified harness configuration around them. Aligning instrument language with the eight-layer view in Table 1, and in particular distinguishing model approval, harness configuration approval and application approval, would close a presently latent gap in the governance architecture.

The role of multilateral institutions. The implications above suggest specific work for multilateral institutions, in addition to their continuing norm-setting role.[60] Three contributions are particularly tractable. First, the articulation of reference architectures for public-interest harnesses,

such as documented blueprints for humanitarian case-management agents, multilingual citizen-service agents or internal knowledge agents, that prescribe defaults at the context, tools, governance, orchestration and session layers (L3–L7 of Table 1) and demonstrate compliance with existing instruments. Second, the pooling of evaluation capacity in low-resource languages and rights-sensitive scenarios, where individual institutional investment is unlikely to be efficient. Third, support for open-source harness components that smaller institutions can adopt, audit and adapt, reducing the costs of compliance with the governance recommendations below.

Illustration: a multilingual humanitarian intake agent. To make the foregoing analysis concrete, consider a hypothetical agent deployed by a humanitarian agency to triage incoming case reports submitted in multiple languages over messaging channels. The harness layers (cf. Table 1) translate directly into design decisions. *L8 (entrypoint):* which messaging channels are supported, and with what identity-verification and language-detection guarantees; refusal behaviour when the input language is not in the supported set. *L7 (session):* per-case persistence with redaction policies aligned with the relevant data-protection regime, and case-level resumption after interruption. *L6 (orchestration):* an iteration budget proportionate to case complexity, with mandatory human escalation when budget is exhausted without resolution. *L5 (governance):* pre- and post-action hooks invoking redaction and a two-person rule for any action that affects beneficiary status; documented refusal paths for sensitive categories (medical, child protection, gender-based violence). *L4 (tools):* a curated tool inventory limited to read-only retrieval from authorised institutional knowledge bases and write access only to a structured case file; no general internet access. *L3 (context):* system-prompt assembly that includes a language-specific behavioural rubric; trajectory-resident summaries that can be reviewed by case officers. *L2 (provider abstraction):* routing between two or more providers covering the target language portfolio, with documented fallback. None of these decisions is a property of the underlying model; each is a property of the harness, and each is a point at which institutional accountability can be realised.

8. Recommendations

The recommendations below are organised into two groups: recommended technical actions for institutions designing, procuring or deploying agentic AI systems, and recommended policy actions for regulators, standard-setting bodies, public institutions and multilateral organisations. They should be applied in a risk-tiered manner. For instance, a read-only internal knowledge agent does not require the same assurance burden as an agent that can write to databases, send messages, modify code, access sensitive personal data or act in rights-sensitive settings. The relevant risk factors include the degree of autonomy, the sensitivity of the data processed, the reversibility of actions, the scope of tool access, the presence of public-facing or rights-sensitive use cases, and whether the deployment operates across languages, jurisdictions or institutional boundaries.

Recommended technical actions

1. Document the harness as a first-class runtime configuration.

For each deployed agent, institutions should document the harness configuration alongside the underlying model and the user-facing application. This documentation should identify the orchestration loop, enabled tools, permission policy, context-management strategy, procedural-memory artefacts, session-retention rules, sandboxing back-end and model-provider routing. Without this layer of documentation, incident review and audit remain incomplete: an institution may know which model was used and which application was deployed, while lacking a record of the runtime conditions under which model outputs became external actions.

The eight-layer framework in Table 1 can serve as a default checklist for such documentation. It need not be adopted as a formal standard, but the separation it makes explicit should be preserved: model approval, harness configuration and application approval are different objects of design and review.

2. Adopt a baseline for reliable long-running execution.

Institutions deploying long-running or tool-using agents should treat several engineering patterns as baseline reliability requirements: bounded iteration budgets, read-only parallelism, controlled write operations, two-stage context compaction, persistence of oversize tool outputs, resumable sessions and trajectory retention. These patterns should not be treated as optional refinements added after deployment. They are the mechanisms that prevent agents from drifting across long horizons, losing state, repeating actions after interruption or exhausting resources without a visible failure point.

The specific implementation may vary by system and deployment context. A low-risk internal agent may require a lighter version of the baseline, while a high-impact agent with write access to institutional systems should satisfy it more fully. In either case, omissions should be documented as risk acceptances rather than left implicit.

3. Make every state-changing action interruptible, resumable and reversible.

Agentic systems should be designed so that state-changing actions can be stopped, resumed safely and, where possible, rolled back. This requires pre- and post-action hooks, checkpointed sessions, idempotent tools and explicit rollback semantics for actions that modify files, databases, code repositories, messages, accounts or public records. The practical test is simple: if an operator interrupts an agent halfway through a task, the system should not repeat destructive actions, lose track of what has already happened or continue acting without renewed authorisation.

This recommendation addresses a core asymmetry in agentic AI: agents can execute long chains of actions faster than human operators can inspect them. Harness design should therefore make interruption and recovery part of normal operation, not exceptional failure handling.

4. Apply least-privilege tool governance and explicit approval gates.

Agents should be granted only the tools and credentials necessary for the task at hand. Tool inventories should distinguish read-only tools from state-changing tools; high-impact actions should require explicit approval; and tool calls should be schema-validated before execution. Sensitive actions following untrusted inputs, such as web pages, email bodies, files or external tool responses, should be subject to hard-coded refusals or escalation rules rather than left to model judgement alone.

No single control is sufficient against prompt injection, credential mishandling or tool blast-radius risks. Effective harness governance combines least-privilege access, sandboxing, approval gates, lifecycle hooks, credential scoping and human override. These controls operate principally at the tools and governance layers (L4 and L5), and should be tested as part of deployment assurance.

5. Preserve portability through open protocols and provider abstraction.

Harnesses should be designed to avoid unnecessary dependence on a single model provider, tool protocol, editor integration or proprietary memory format. Provider-abstraction layers should make it possible to route model calls across vendors or local inference back-ends without redesigning the whole system. Tool-access and context-sharing protocols should, where feasible, follow open or widely adopted conventions.

Portability is a technical preference and a procurement and resilience issue. A system that cannot substitute providers is exposed to price changes, rate-limit changes, product discontinuation, jurisdictional constraints and unilateral terms-of-service revisions. Open protocols at the provider, context and tools layers (L2–L4) reduce these dependencies and support longer-term institutional control.

6. Evaluate the model–harness pair, not the model alone.

Agentic capability should be evaluated at the level of the model–harness pair. A benchmark result for a standalone model does not establish the reliability of a deployed agent, because the effective system also depends on tool wrappers, context assembly, memory design, permission policies, iteration budgets, sandboxing, compaction and recovery procedures. Conversely, a modest model embedded in a strong harness may outperform a more capable model embedded in a weak one on long-horizon tasks.

Evaluation should therefore include end-to-end agent runs, realistic tool environments, multilingual and domain-specific tasks where relevant, and failure analysis of trajectories. Model-level evaluation remains necessary, but it is insufficient for approving agentic systems that act in the world.

Recommended policy actions

7. Recognise the harness as a distinct object of governance.

AI governance instruments should explicitly distinguish the underlying model, the harness configuration and the user-facing application. Current policy language often refers broadly to "AI systems", but this can obscure the runtime layer that determines how model outputs become external actions. A model-level approval should not automatically authorise any harness configuration around that model, especially where the harness grants tool access, persistent memory, autonomous execution or write privileges.

Regulators, procurers and institutional review bodies should therefore ask two questions: "which model is being used?" and "under what runtime conditions is the model allowed to act?" This distinction is central to accountability for agentic AI.

8. Privilege legibility, recovery and substitutability in procurement criteria.

Procurement should not select agentic systems on benchmark performance alone. Institutions should weigh legibility, recoverability and substitutability alongside capability. A system that performs slightly better on a benchmark but provides no inspectable trajectories, no readable compaction summaries, no safe interruption mechanism and no realistic provider-substitution path may be a worse institutional choice than a less capable but more governable system.

This recommendation is especially relevant to public-interest deployments. The question is whether an agent can complete a task and whether the deploying institution can understand, supervise, interrupt, resume, audit and eventually replace it.

9. Treat skills, MCP servers, plug-ins and tool wrappers as supply-chain components.

Skills, MCP servers, plug-ins and tool wrappers should be governed as software supply-chain components. They should have provenance metadata, review processes, versioning, integrity checks and, where feasible, signed manifests. High-trust deployments should rely on curated registries or approved internal repositories rather than unrestricted third-party extensions. Where external components are used, they should be sandboxed and monitored.

This recommendation follows directly from the operational structure of agentic AI. A compromised tool server or malicious skill file can shape the agent's future actions just as a compromised software dependency can shape a conventional application. Supply-chain governance therefore extends naturally to the harness layer.

10. Develop multilingual and rights-sensitive evaluation environments as shared public goods.

Existing agent benchmarks remain heavily concentrated in English-language, software-oriented and commercially legible tasks. Public-interest deployments require evaluation environments that reflect multilingual communication, low-resource languages, humanitarian intake, public-service delivery, cross-jurisdictional document handling and rights-sensitive decision pathways. These environments should test model accuracy, tool use, escalation behaviour, refusal policy, memory handling, auditability and recovery.

Such evaluation infrastructure is costly to build and inefficient for individual institutions to duplicate. International organizations, research institutions and multilateral partners should support shared evaluation suites and sandboxes as public goods, especially for institutions in the Global South.

11. Build capacity for operator-level harness governance.

Capacity-building for AI literacy should address the practical work of supervising agentic systems, since many end-users will not train models or design foundation architectures. The ability to configure tools, review skills, approve actions, supervise sessions, inspect trajectories and manage escalation should therefore be a focus for responsible adoption.

12. Support comparative research on harness architectures and model–harness interaction.

The field needs sustained comparative research on how different harness configurations affect reliability, safety, cost and capability. Existing evidence from skills benchmarks, native-runtime evaluations, workspace benchmarks and harness-evolution studies suggests that harness design can materially alter agent performance, but the field still lacks mature controlled studies across domains, languages and deployment settings.

Publicly available analyses of closed systems can contribute to this research where they are legally and ethically obtained, clearly caveated and cross-checked against documented or open-source implementations. Open-source harnesses should be treated as critical research infrastructure, because they allow the field to inspect, reproduce and improve the runtime layer rather than treating agentic capability as a property of models alone.

13. Ensure that harness adoption narrows rather than widens the global AI divide.

Harness engineering is becoming a new capacity frontier in AI adoption. Institutions with strong software engineering teams can build reliable, auditable and portable agents; institutions without such capacity may be left with opaque hosted systems, weak procurement leverage and limited ability to inspect or adapt deployments. Without intervention, the harness layer may reproduce the same inequalities already visible in compute, data and model access.

Policies should therefore support open reference harnesses, shared sandboxes, multilingual evaluation suites, procurement templates and technical assistance for lower-resource institutions. The goal is to make advanced agentic AI available under conditions that institutions can understand, audit and adapt.

Conclusion

The term *harness* may or may not persist as the vocabulary of choice. It is analytically useful because it identifies the runtime layer that earlier terminology (*environment*, *platform*, *workflow*) only partially captured. Agentic capability is best understood as the joint product of model, environment and the runtime scaffold that mediates between them, rather than as a property of any one of these alone.

This has consequences for governance. The harness is presently the main site of design choices that determine whether agentic systems behave reliably, recover from failure, remain auditable and avoid lock-in. It is also, at present, the layer least explicitly addressed by existing instruments. The recommendations above are intended as a concrete step toward closing this gap, drawing on the convergent practice of open-source and frontier-laboratory engineering rather than on hypothetical futures.

For sustainable development specifically, the relevant question concerns the runtime conditions under which agents become legible, corrigible, interoperable and accountable in the institutions that will use them, rather than how quickly such agents can be set loose. On the present evidence the task is feasible, and good practice is increasingly clear. Agents do not first become capable and only afterwards become governable. They become capable through being organised, constrained and embedded in the right technical and institutional environments. In this layer, capability depends on boundaries.

References

1 Young, J. (2025, November 26). Effective harnesses for long-running agents. *Anthropic Engineering*. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents

2 Lopopolo, R. (2026, February 11). Harness engineering: Leveraging Codex in an agent-first world. *OpenAI*. https://openai.com/index/harness-engineering/

3 Böckeler, B. (2026, April 2). Harness engineering for coding agent users. *martinfowler.com*. https://martinfowler.com/articles/harness-engineering.html

4 Wooldridge, M. J. (2009). *An introduction to multiagent systems* (2nd ed.). John Wiley & Sons.

5 Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=WE_vluYUL-X

6 Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems, 36*. https://doi.org/10.48550/arXiv.2302.04761

7 Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J.-R. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science, 18*, Article 186345. https://doi.org/10.1007/s11704-024-40231-1

8 European Parliament and Council of the European Union. (2024). *Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No. 300/2008, (EU) No. 167/2013, (EU) No. 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act)*. Official Journal of the European Union, L 2024/1689. http://data.europa.eu/eli/reg/2024/1689/oj

9 Organisation for Economic Co-operation and Development. (2024). *Recommendation of the Council on Artificial Intelligence* (OECD/LEGAL/0449). OECD Legal Instruments. https://legalinstruments.oecd.org/en/instruments/OECD-LEGAL-0449

10 Ng, A. (2024, March 20). Four AI agent strategies that improve GPT-4 and GPT-3.5 performance. *The Batch*. DeepLearning.AI. https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance

11 Sequoia Capital. (2024, March). What's next for AI agentic workflows ft. Andrew Ng of AI Fund [Video]. *YouTube*. https://www.youtube.com/watch?v=sal78ACtGTc

12 Ding, S., Dai, X., Xing, L., Ding, S., Liu, Z., Yang, J., Yang, P., Zhang, Z., Wei, X., Fang, X., Ma, Y., Duan, H., Shao, J., Wang, J., Lin, D., Chen, K., & Zang, Y. (2026). *WildClawBench: A benchmark for real-world, long-horizon agent evaluation*. arXiv. https://doi.org/10.48550/arXiv.2605.10912

13 Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y., & Scialom, T. (2023). *GAIA: A benchmark for General AI Assistants*. arXiv. https://doi.org/10.48550/arXiv.2311.12983

14 Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., Alon, U., & Neubig, G. (2024). WebArena: A realistic web environment for building autonomous agents. *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=oKn9c6ytLx

15 Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2024). SWE-bench: Can language models resolve real-world GitHub issues? *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VTF8yNQM66

16 Yao, S., Shinn, N., Razavi, P., & Narasimhan, K. (2024). *τ-bench: A benchmark for tool-agent-user interaction in real-world domains*. arXiv. https://doi.org/10.48550/arXiv.2406.12045

17 Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 611-626). Association for Computing Machinery. https://doi.org/10.1145/3600006.3613165

18 Ji, Y. (2025, July 18). Context engineering for AI agents: Lessons from building Manus. *Manus Blog*. https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus

19 Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

20 Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., & Yu, T. (2024). *OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments*. arXiv. https://doi.org/10.48550/arXiv.2404.07972

21 Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., & Tramèr, F. (2024). AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *Advances in Neural Information Processing Systems, 37*, 82895-82920. https://openreview.net/forum?id=m1YYAQjO3w

22 Huang, Y., Li, S., Liu, M., Liu, W., Huang, S., Fan, Z., Chan, H. P., & Fung, Y. R. (2025). *Scaling environments for LLM agents in the era of learning from interaction: A survey*. arXiv. https://doi.org/10.48550/arXiv.2511.09586

23 Liu, J., Zhao, X., Shang, X., & Shen, Z. (2026). *Dive into Claude Code: The design space of today's and future AI agent systems*. arXiv. https://doi.org/10.48550/arXiv.2604.14228

24 Cursor. (n.d.). *Agent overview*. Retrieved July 6, 2026, from https://cursor.com/docs/agent/overview

25 Aider. (n.d.). *Aider documentation*. Retrieved July 6, 2026, from https://aider.chat/docs/

26 Cline. (n.d.). *Cline overview*. Retrieved July 6, 2026, from https://docs.cline.bot/cline-overview

27 Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). *AutoGen: Enabling next-gen LLM applications via multi-agent conversation*. arXiv. https://doi.org/10.48550/arXiv.2308.08155

28 CrewAI. (n.d.). *CrewAI documentation*. Retrieved July 6, 2026, from https://docs.crewai.com/

29 Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., & Schmidhuber, J. (2024). MetaGPT: Meta programming for a multi-agent collaborative framework. *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=VtmBAGCN7o

30 OpenClaw. (n.d.). *OpenClaw docs*. Retrieved July 6, 2026, from https://docs.openclaw.ai/

31 HKUDS. (n.d.). *OpenHarness: Open agent harness with a built-in AI agent marketplace* [Computer software]. GitHub. Retrieved July 6, 2026, from https://github.com/HKUDS/OpenHarness

32 Nous Research. (n.d.). *Hermes Agent documentation*. Retrieved July 6, 2026, from https://hermes-agent.nousresearch.com/docs/

33 Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). *Constitutional AI: Harmlessness from AI feedback*. arXiv. https://doi.org/10.48550/arXiv.2212.08073

34 Li, X., Liu, Y., Chen, W., You, B., Di, Z., He, Y., Zheng, S., Choe, K. W., Sun, J., Wang, S., Tao, C., Li, B., Zhao, X., Geng, H., Wu, X., Zhou, J., Chen, X., Xing, H., Li, Y., ... Song, D. (2026). *SkillsBench: Benchmarking how well agent skills work across diverse tasks*. arXiv. https://doi.org/10.48550/arXiv.2602.12670

35 Ding, S., Dai, X., Xing, L., Ding, S., Liu, Z., Yang, J., Yang, P., Zhang, Z., Wei, X., Fang, X., Ma, Y., Duan, H., Shao, J., Wang, J., Lin, D., Chen, K., & Zang, Y. (2026). *WildClawBench: A benchmark for real-world, long-horizon agent evaluation*. arXiv. https://doi.org/10.48550/arXiv.2605.10912

36 Tang, Z., Zhou, X., Liu, Y., Li, L., Wang, W., Huang, H., Zhou, J., Song, J., Yu, S., Wang, J., Zhou, Z., Zhou, H., Lv, Y., Li, J., Liu, J., Chen, R., Liu, C., Li, G., Kang, J., & Wu, F. (2026). *Workspace-Bench 1.0: Benchmarking AI agents on workspace tasks with large-scale file dependencies*. arXiv. https://doi.org/10.48550/arXiv.2605.03596

37 Lin, J., Liu, S., Pan, C., Lin, L., Dou, S., Huang, X., Yan, H., Han, Z., & Gui, T. (2026). *Agentic Harness Engineering: Observability-driven automatic evolution of coding-agent harnesses*. arXiv. https://doi.org/10.48550/arXiv.2604.25850

38 Wood, D., Bruner, J. S., & Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry, 17*(2), 89-100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

39 Zhang, Z., Cui, S., Lu, Y., Zhou, J., Yang, J., Wang, H., & Huang, M. (2024). *Agent-SafetyBench: Evaluating the safety of LLM agents*. arXiv. https://doi.org/10.48550/arXiv.2412.14470

40 Zhan, Q., Liang, Z., Ying, Z., & Kang, D. (2024). InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. *Findings of the Association for Computational Linguistics: ACL 2024*, 10471-10506. https://doi.org/10.18653/v1/2024.findings-acl.624

41 Yuan, T., He, Z., Dong, L., Wang, Y., Zhao, R., Xia, T., Xu, L., Zhou, B., Li, F., Zhang, Z., Wang, R., & Liu, G. (2024). R-Judge: Benchmarking safety risk awareness for LLM agents. *Findings of the Association for Computational Linguistics: EMNLP 2024*, 1467-1490. https://doi.org/10.48550/arXiv.2401.10019

42 Anwar, U., Saparov, A., Rando, J., Paleka, D., Turpin, M., Hase, P., Lubana, E. S., Jenner, E., Casper, S., Sourbut, O., Edelman, B. L., Zhang, Z., Günther, M., Korinek, A., Hernandez-Orallo, J., Hammond, L., Bigelow, E., Pan, A., Langosco, L., ... Krueger, D. (2024). *Foundational challenges in assuring alignment and safety of large language models*. arXiv. https://doi.org/10.48550/arXiv.2404.09932

43 Gan, Y., Yang, Y., Ma, Z., He, P., Zeng, R., Wang, Y., Li, Q., Zhou, C., Li, S., Wang, T., Gao, Y., Wu, Y., & Ji, S. (2024). *Navigating the risks: A survey of security, privacy, and ethics threats in LLM-based agents*. arXiv. https://doi.org/10.48550/arXiv.2411.09523

44 OpenAI. (n.d.). *Compaction*. OpenAI API documentation. Retrieved July 6, 2026, from https://developers.openai.com/api/docs/guides/compaction

45 Ng, C. (2026, April 1). When trust becomes a weapon: The Axios npm supply chain attack. *UNU Campus Computing Centre*. https://c3.unu.edu/blog/when-trust-becomes-a-weapon-the-axios-npm-supply-chain-attack

46 Liu, J. (2026, April 2). Why agentic AI needs boundaries before freedom. *UNU Macau Blog*. https://unu.edu/macau/blog-post/why-agentic-ai-needs-boundaries-freedom

47 United Nations. (2024, September 19). *UN Secretary-General's High-level Advisory Body on Artificial Intelligence releases proposals for global governance of AI* [Press release]. https://www.un.org/sites/un2.un.org/files/governing_ai_for_humanity_press_release.pdf

48 Marwala, T., Fournier-Tombs, E., & Stinckwich, S. (2023a). *The use of synthetic data to train AI models: Opportunities and risks for sustainable development* (UNU Technology Brief No. 1). United Nations University. https://unu.edu/publication/use-synthetic-data-train-ai-models-opportunities-and-risks-sustainable-development

49 Marwala, T., Fournier-Tombs, E., & Stinckwich, S. (2023b). *Regulating cross-border data flows: Harnessing safe data sharing for global and inclusive artificial intelligence* (UNU Technology Brief No. 3). United Nations University. https://unu.edu/publication/regulating-cross-border-data-flows-harnessing-safe-data-sharing-global-and-inclusive

50 United Nations. (2024). *Global Digital Compact*. https://www.un.org/global-digital-compact/sites/default/files/2024-09/Global%20Digital%20Compact%20-%20English_0.pdf

51 United Nations General Assembly. (2024). *Seizing the opportunities of safe, secure and trustworthy artificial intelligence systems for sustainable development* (A/RES/78/265). https://digitallibrary.un.org/record/4043244/files/A_RES_78_265-EN.pdf

52 United Nations Secretary-General's High-level Advisory Body on Artificial Intelligence. (2024). *Governing AI for humanity: Final report*. United Nations. https://www.un.org/sites/un2.un.org/files/governing_ai_for_humanity_final_report_en.pdf

53 United Nations System Chief Executives Board for Coordination. (2024a). *United Nations system white paper on AI governance*. https://unsceb.org/sites/default/files/2024-05/United%20Nations%20System%20White%20Paper%20on%20AI%20Governance.pdf

54 United Nations System Chief Executives Board for Coordination. (2024b). *Framework for a model policy on the responsible use of AI in UN system organizations* (CEB/2024/HLCM/28/Add.1/Rev.1). https://unsceb.org/sites/default/files/2024-11/Framework%20for%20a%20Model%20Policy%20on%20the%20%20Responsible%20Use%20of%20AI%20in%20UN%20System.pdf

55 United Nations System Chief Executives Board for Coordination. (2024c). *Report on the operational use of AI in the UN system*. https://unsceb.org/sites/default/files/2024-11/Report%20on%20the%20Operational%20Use%20of%20AI%20in%20the%20UN%20System_1.pdf

56 National Institute of Standards and Technology. (2023). *Artificial Intelligence Risk Management Framework (AI RMF 1.0)* (NIST AI 100-1). U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

57 Group of Seven. (2023, October 30). *Hiroshima Process International Code of Conduct for Organizations Developing Advanced AI Systems*. https://www.mofa.go.jp/files/100573473.pdf

58 Linux Foundation. (2025, December 9). Linux Foundation announces the formation of the Agentic AI Foundation (AAIF). *Linux Foundation*. https://www.linuxfoundation.org/press/linux-foundation-announces-the-formation-of-the-agentic-ai-foundation

59 Anthropic. (2025, December 9). *Donating the Model Context Protocol and establishing the Agentic AI Foundation*. https://www.anthropic.com/news/mcp-aaif

60 Liu, J., Chu, C., Zhao, Y., Aoki, G., & Xiao, Z. (2025). Agentic AI for sustainable development: Leveraging large language model-enhanced agent-based modeling for complex policy strategies. *Emerging Media, 3*(3), 401-413. https://doi.org/10.1177/27523543251365678