Harness Engineering for LLM Agents: A Survey of Harness Component Taxonomy, Evaluation, and Model–Harness Coevolution

Anonymous authors
Paper under double-blind review

Abstract

As large language model-driven autonomous agents are increasingly deployed in real-world long-horizon, open-environment tasks, foundation models expose systematic capability gaps in context retention, reliable tool invocation, persistent state management, and multi-step execution robustness. We understand the agent harness as the external execution support structure built around the model and treat it as a distinct performance lever that complements base model capability, positioning harness engineering as a growing area of research and engineering practice. Grounded in the scaffolding perspective from developmental psychology, we structure the field along three nested levels of analysis. At the structural level, we develop a unified taxonomy of harness components, mapping them to the specific capability gaps they compensate for and to their coupling with the core agent loop. At the fit level, we articulate a two-stage evaluation logic that distinguishes native model capability-gap diagnosis from assessments of compensation effectiveness and net benefit, and unpack the inherent multi-objective tradeoffs shaping harness design. At the dynamic level, we delineate the bidirectional coevolution mechanism between models and harnesses, explaining the shifting functional boundary where routine capability-bearing support migrates inward into model weights while constraint-bearing governance functions remain external. By synthesizing studies that are currently scattered across adjacent areas, we provide an organizing framework for understanding LLM agent harnesses in relation to the capability gaps they address. We further discuss open challenges and future directions in harness design, evaluation, and model–harness coevolution.

1 Introduction

When a task exceeds the independent capacity of a single cognitive subject, a capability gap emerges. Developmental psychology characterizes a central mechanism for bridging such gaps as scaffolding: external support structures enable a learner to accomplish tasks that would otherwise remain beyond unaided reach Wood et al. (1976). As competence increases, effective scaffolding is expected to fade and transfer responsibility to the learner, while higher task demands may require new forms of support Van de Pol et al. (2010); Vygotsky (1978). This cycle of construction, internalization, withdrawal, and reconstruction offers a useful lens for analyzing support structures that help complex systems move from fragile competence toward reliable performance.

Research on scaffolding can be organized around three progressively richer levels of analysis. At the structural level, studies ask what components constitute a scaffold and which capability gaps they compensate for Wood et al. (1976). At the fit level, they ask whether support is aligned with the actual learner need, and whether under-support or over-support produces a mismatch Van de Pol et al. (2010). At the dynamic level, they ask how support is faded, how responsibility is transferred as competence changes, which capabilities become internalized, and which remain distributed across the subject and external artifacts Van de Pol et al. (2010); Vygotsky (1978); Hollan et al. (2000).

The evolution of large language models (LLMs) and agentic systems exhibits a structurally analogous cycle. As foundation models improve in reasoning and generation, LLM-driven autonomous agents are deployed in software engineering, scientific work, information analysis, and other professional settings Wang et al. (2024); Xu & Peng (2025). These deployments extend beyond conversational interaction into long-running tasks with stronger reliability constraints. Under realistic conditions, several systematic capability gaps become salient. Models require mediated interfaces to turn reasoning into external actions, making tool use and action grounding central deployment problems Yao et al. (2023b); Qin et al. (2023); Qu et al. (2025). Attention and information localization degrade over long trajectories, causing important state to be forgotten or submerged in noise Zeng et al. (2026). The lack of persistent cross-session state constrains continual learning from prior experience Hu et al. (2026b); Du (2026). Multi-step execution amplifies error accumulation and state drift, gradually pushing trajectories away from the goal Wang et al. (2026c). Reliability improvements also introduce latency and cost tradeoffs that must be managed at the workflow level Yang & Zhu (2026). These are not isolated defects. They are systemic gaps in cognition, execution, and governance that become visible when agents move from controlled demonstrations into open-ended environments.

Engineering practice compensates for these gaps by building support structures outside the model. These supports manage context, expose tools, preserve state, organize workflows, provide execution environments, externalize verification and guardrails, and record observable traces. Around the agent loop, they are commonly described in engineering practice as an agent *harness* LangChain (2026); Lopopolo (2026). Holding the foundation model fixed, changes to the harness can substantially alter end-to-end performance, making the harness a performance lever orthogonal to model capability Ding et al. (2026b); Ndzomga (2026). As task complexity increases, the harness becomes a research and engineering object in its own right: *harness engineering* concerns how these external support structures are constructed, organized, evaluated, and governed Lin et al. (2026b).

Existing discussions of harness engineering remain fragmented. Context engineering, tool learning and function calling, memory mechanisms, interoperability protocols, safety governance, and evaluation methods have developed partially independent literatures and engineering communities Anthropic (2025c); Qu et al. (2025); Du (2026); Ehtesham et al. (2025); Su et al. (2026); Mohammadi et al. (2025). These literatures often describe which components exist. They give less attention to why the components emerge, how they match specific capability gaps, and how they are withdrawn or reorganized as model generations improve. Broad surveys of LLM agents cover many of these support structures, but they are typically organized around the model or the agent as the primary object. The external harness therefore remains a dispersed set of auxiliary modules Wang et al. (2024). Engineering posts on context and skills show how runtime artifacts are increasingly organized outside the model, but they do not yet amount to a unified academic taxonomy of harness engineering Anthropic (2025c;b). We therefore argue that harness engineering must answer three linked questions corresponding to the three scaffolding levels: what a harness consists of, whether it compensates for the right gap at acceptable cost, and which supports become model-side capabilities rather than external governance layers.

Accordingly, we develop a structured synthesis of harnesses along the three levels of scaffolding analysis. At the structural level, *Harness Component Taxonomy*, we use capability gaps as the organizing axis, propose a unified taxonomy of seven component families, and explain their boundaries and coupling relationships with respect to the agent loop. At the fit level, *Harness Evaluation*, we distinguish model capability-gap diagnosis from harness compensation assessment. The former assesses native capability limits; the latter measures compensation effectiveness and net benefit under multi-objective constraints such as reliability, cost, autonomy, safety, and long-horizon continuity. At the dynamic level, *Harness Evolution*, we treat evolution as a bidirectional process. Harnesses promote capability internalization by accumulating trajectories, feedback, and experience (Harness $\rightarrow$ Model). Improved model capabilities in turn drive harness optimization at the interface, component, workflow, and closed-loop levels (Model $\rightarrow$ Harness). This dynamic distinction also clarifies why routine capability-bearing support can move toward the model, whereas constraint-bearing support remains tied to external verification, permission control, and governance. Together, these dynamics continually reshape the model–harness boundary. Figure 1 summarizes the literature organization that supports the three analytical levels.

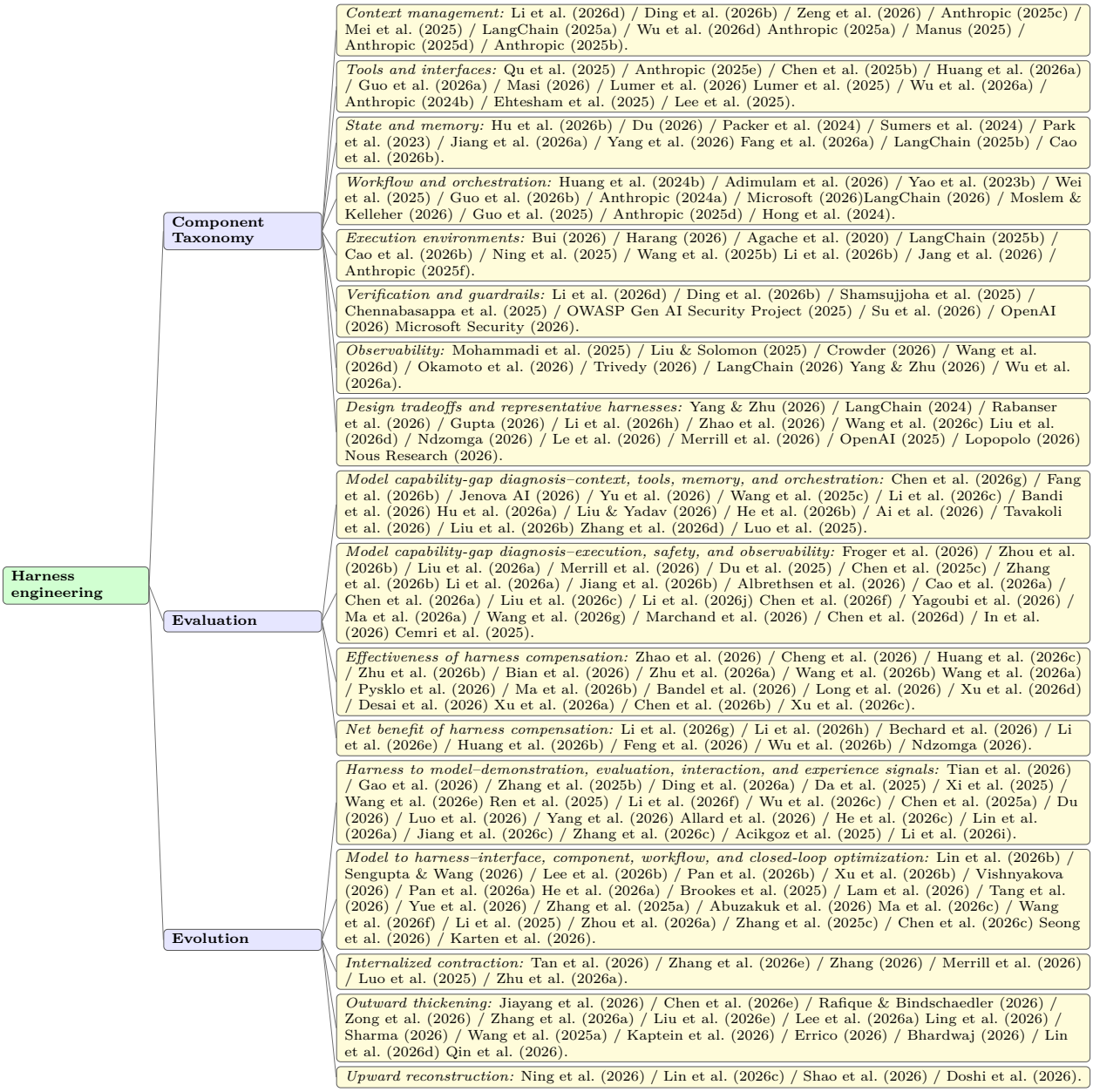

Figure 1: Tree-structured organization of the literature in the three analytical sections. The primary branches correspond to Harness Component Taxonomy, Harness Evaluation, and Harness Evolution, and the leaf nodes show the internal categories used to organize cited work.

2 Harness Foundations: From Model Capability Gaps to External Support

Building on the scaffolding lens introduced above, we first trace how external support structures have shifted across successive LLM capability boundaries. When models encounter a boundary in a particular capability, research and engineering practice first tend to construct targeted support outside the model to compensate for what the model cannot yet perform independently and reliably. When the corresponding behavior can be captured by stable training targets and repeatable feedback, these support patterns can move closer to model-side capability, and the former external support loses part of its role as a necessary condition. Improved model capability then expands deployment toward more complex and open-ended tasks, where new capability gaps become visible at a higher level. These developments reveal a cycle of gap exposure, external compensation, training internalization, and renewed gap exposure Wood et al. (1976); Zaharia et al. (2024).

To date, this pattern can be seen in two earlier boundary shifts in the evolution of LLMs, while a third shift is still unfolding. The first was the intent-alignment cycle. Pretraining gave models strong language-generation capacity, but did not ensure that their outputs would match user intent; few-shot prompting therefore served as an external alignment scaffold before reinforcement learning from human feedback and instruction tuning internalized the relevant behavior into model weights. The second was the reasoning-capability cycle. Even after models could follow instructions, they often omitted intermediate reasoning steps on complex tasks; chain-of-thought prompting became an external reasoning scaffold, and later specialized training made step-by-step reasoning a more endogenous model capability. The third and current cycle concerns execution capability. As models are applied to long-running tasks in real environments, gaps in tool use, state maintenance, environmental feedback processing, side-effect control, and failure recovery become concentrated. We refer to the resulting system-level external execution support as the harness, synthesizing recent academic surveys and official platform documentation Zaharia et al. (2024); Meng et al. (2026); Anthropic (2024b). The remainder of this section follows the temporal logic of these boundary shifts: it first reviews alignment and reasoning, then characterizes the still-open execution cycle, and finally formalizes the concepts of harness and harness engineering.

2.1 Intent Alignment Cycle

Pretrained language models that have not undergone explicit alignment training commonly exhibit a gap in intent following. Although they can generate fluent language, they do not reliably produce outputs that match user goals. Their responses may deviate in style or stance, and they may produce harmful, biased, or factually unreliable content. Merely scaling model size does not reliably remove these problems Ouyang et al. (2022); Brown et al. (2020); Bender et al. (2021); Weidinger et al. (2021).

Before alignment training became mature, few-shot prompting was a primary way to mitigate this gap. By embedding carefully selected demonstrations in the input prompt, developers could temporarily inject an expected behavioral pattern and steer the model toward the desired output form Brown et al. (2020); Liu et al. (2023). This intervention was external to the model itself. Its effectiveness depended heavily on the quality and coverage of the examples in a single prompt, and it could fail when the task format or wording shifted. It was also difficult to reuse reliably across users and tasks, making it insufficient as a scalable alignment mechanism.

Reinforcement learning from human feedback and later preference-optimization methods changed this arrangement. Through human preference annotation, reward-model construction, and policy optimization, the ability to follow human intent was trained into model behavior more systematically Ouyang et al. (2022); Christiano et al. (2023); Ziegler et al. (2020); Stiennon et al. (2022); Bai et al. (2022); Rafailov et al. (2024). InstructGPT was a landmark result: a 1.3-billion-parameter aligned model was preferred in human evaluations over the 175-billion-parameter GPT-3 baseline and over prompting-based baselines of the same model family Ouyang et al. (2022). This demonstrated that intent alignment had moved from an external compensatory technique toward an internalized model capability.

The first cycle therefore illustrates a clear boundary shift. Manually designed alignment demonstrations lost their central role as the primary mechanism for intent following once preference-based post-training made

instruction following a more stable model behavior. They remained useful in particular cases, but no longer served as the mandatory external structure for basic intent alignment.

2.2 Reasoning Capability Cycle

Intent alignment improved the fit between model outputs and user requests, but it did not by itself solve multi-step reasoning. Even when models understood instructions, they often defaulted to direct answers on arithmetic, symbolic manipulation, and multi-hop logical tasks, omitting intermediate steps and showing rapidly declining accuracy as complexity increased Wei et al. (2023); Kojima et al. (2023). This gap motivated external support during the reasoning stage.

Chain-of-thought prompting became the canonical external reasoning scaffold. By adding examples of step-by-step reasoning to the prompt, researchers significantly improved performance on complex tasks. Even a zero-shot trigger such as "Let's think step by step" could activate latent stepwise reasoning behavior Wei et al. (2023); Kojima et al. (2023). Subsequent methods such as self-consistency and Tree of Thoughts extended this scaffold into richer forms of sampling, voting, and path search Wang et al. (2023); Yao et al. (2023a). Surveys of chain-of-thought and prompting techniques further show that these methods developed into a broad engineering repertoire Chu et al. (2024); Schulhoff et al. (2025). In essence, however, these techniques remained external interventions at inference time: the reasoning procedure was injected through prompting rather than being a stable capability of the model.

Specialized reasoning training then began to move this capability boundary inward. Systems such as OpenAI's o1 models and DeepSeek-R1 used large-scale reinforcement learning to train models to produce and refine stepwise reasoning processes without relying on user-supplied demonstrations OpenAI et al. (2026); DeepSeek-AI et al. (2026). DeepSeek-R1-Zero is particularly relevant here because it showed that step-by-step reasoning behavior can emerge under reinforcement learning without supervised chain-of-thought data, suggesting that the structure is not simply imported from external exemplars but can arise endogenously under an appropriate training pressure DeepSeek-AI et al. (2026).

Correspondingly, carefully engineered chain-of-thought prompts provide much smaller marginal benefits for newer reasoning models, and overly prescriptive reasoning prompts may even interfere with strong models' internal reasoning processes Meincke et al. (2025). The second cycle shows a related but more model-dependent boundary movement. As reasoning-oriented training made stepwise problem solving more endogenous to the model, chain-of-thought prompting shifted from a central scaffold for eliciting reasoning to a task- and model-dependent intervention.

2.3 Execution Capability Cycle and the Emergence of the Harness

As intent alignment and reasoning capability were progressively internalized, models approached the limits of many closed-form task settings. The center of gravity in research and deployment shifted from whether a model could reason correctly to whether it could execute reliably in real environments. This shift had an important precursor in ReAct, which interleaved reasoning and action so that a model could call external tools and interact with an environment over multiple turns Yao et al. (2023b). It also contributed to the broader rise of LLM-based autonomous agents. Researchers and practitioners increasingly expected agents to complete goals in open-ended, long-horizon settings that require persistent interaction with the environment. In deployment, however, models exposed execution-level weaknesses: state is lost across multi-step interaction, tool calls may lack validation, errors cascade along execution chains, and agents often fail to recover after breakdowns Wang et al. (2024); Xi et al. (2023); Sumers et al. (2024); Cemri et al. (2025); Xie et al. (2026).

Benchmarks quantify this execution gap. Leading models have achieved only low end-to-end success rates on realistic web tasks, initially solved fewer than four percent of real GitHub issues in SWE-bench, and completed less than one third of tasks autonomously in simulated enterprise settings Zhou et al. (2024); Jimenez et al. (2024); Xu et al. (2025a). Tool-agent-user interaction benchmarks further show that pass rates across repeated runs can remain far below single-run success rates, revealing that reliability cannot be reduced to one-shot accuracy Yao et al. (2024). Recent work on agent reliability makes the same point

more generally: consistency, robustness, predictability, and bounded error severity are distinct operational dimensions that are obscured by a single success metric Rabanser et al. (2026). AgentBench and studies of self-correction likewise show that current LLMs still struggle to operate as reliable agents in complex interactive environments and cannot be assumed to repair their own failures without external feedback Liu et al. (2025); Huang et al. (2024a).

This divergence between single-run success and cross-run reliability marks a key difference between the execution cycle and the previous two cycles. Alignment and reasoning gaps could be narrowed substantially through scale and training. Execution reliability does not appear to converge monotonically with model size alone Rabanser et al. (2026). Moreover, many core requirements of execution settings, including permission control, side-effect constraints, and external-system connectivity, are valuable precisely because they remain independent of model weights and can be separately inspected, governed, and audited Su et al. (2026); Wang et al. (2025a); Kaptein et al. (2026); OWASP Gen AI Security Project (2025).

Engineering practice has therefore again turned first to support outside the model. Unlike the previous two cycles, however, the support structure is no longer limited to text-level prompt optimization. It includes task orchestration, state management, tool integration, safety constraints, observability, and failure recovery as system-level execution infrastructure Zaharia et al. (2024); Wu et al. (2023); Hong et al. (2024); Anthropic (2024b); Meng et al. (2026). This external structure, built around a language model and oriented toward reliable execution, is the harness. The systematic engineering practice around its design, construction, evaluation, and evolution is harness engineering.

The support functions inside the third cycle are not homogeneous. They can be divided into two classes with different evolutionary paths. The first class is capability-bearing support, including task orchestration, state tracking, tool-call decisions, and failure recovery. Tool-use training, native function calling, and model-side context management show how such functions move inward when the behavior admits stable training targets and repeatable feedback. The second class is constraint-bearing support, including safety guardrails, permission and identity management, side-effect control, and interfaces to external systems. The persistence of permission control, side-effect constraints, and audit requirements shows why these functions follow a different path: their value comes from separation from model weights, independent deployability, and auditability Su et al. (2026); Wang et al. (2025a); Kaptein et al. (2026); OWASP Gen AI Security Project (2025).

Early signs of inward migration are already visible for capability-bearing functions. Native function calling, self-supervised tool-use training, computer-use interfaces, model-driven context management, and API-oriented fine-tuning all suggest that parts of the harness that were previously external can become model capabilities OpenAI (2023); Schick et al. (2023); Anthropic (2024c; 2025c); Patil et al. (2023). Yet this migration is incomplete, and it does not imply that the harness will disappear. Current frontier systems still have significant reliability deficits Yao et al. (2024); Rabanser et al. (2026). At the same time, stronger models are entrusted with longer, more complex, and riskier tasks; METR reports that the length of tasks that frontier agents can complete at a fixed reliability threshold has been doubling on the order of months Kwa et al. (2026). Thus, even as some individual supports weaken, the total need for system-level execution support may grow.

Constraint-bearing functions follow this external logic most clearly. Theoretical and empirical work on hallucination argues that some limitations of LLMs cannot be completely eliminated, which explains why retrieval augmentation and output validation remain important even as model scale increases Gao et al. (2024); Xu et al. (2025b). These external structures should not be treated as temporary compensatory mechanisms. They are part of a governance and verification layer that remains outside the model because its purpose is to provide independent evidence and control.

The third cycle therefore remains open and does not point toward the disappearance of external support. Current systems reveal a shift in the harness rather than its removal: low-level capability support can contract, while governance, verification, and audit layers retain their external role. For this reason, the harness is both the current intermediate layer between model reasoning and reliable execution and a long-lived technical object whose form continues to evolve. The evolving model–harness boundary is the central theme of the following sections on Harness Component Taxonomy, evaluation, and evolution.

2.4 Definition and Conceptual Boundaries of the Harness

Based on the preceding analysis, a harness can be defined as a system-level external execution structure built around a language model and independent of model weights. Its core function is to transform the model's reasoning and generation capability into reliable and controllable task execution in a real environment. The harness sits between the model and the task environment: inwardly, it receives and organizes model outputs; outwardly, it connects to tools, environmental feedback, and execution substrates. Its core components include task orchestration, cross-step state management, tool integration and call validation, safety guardrails and permission constraints, output verification, and failure detection and recovery. It is therefore a first-class architectural object rather than a peripheral implementation detail Meng et al. (2026). Within this architecture, the model performs reasoning and generation, while the harness turns reasoning results into a controlled execution process.

The harness is not merely an accessory to the model. It is an independent dimension of performance optimization. Holding model weights fixed, changes to the harness can substantially improve an agent's end-to-end performance. For example, LangChain reports that harness-level changes such as prompts and middleware hooks moved the same coding model from roughly the top 30 to the top 5 on Terminal-Bench 2.0, with a score increase from 52.8 to 66.5 Trivedy (2026). This separability raises the harness from surrounding engineering to an object of independent research value Zaharia et al. (2024); Meng et al. (2026).

Three boundaries clarify the concept. First, a harness is broader than a prompt. A prompt is textual input to the model; the harness controls how model calls are organized, how context is assembled, how outputs are checked, and how results are passed to the environment. Prompt design is only one function within a harness. Second, a harness is distinct from an agent. A runnable agent can be decomposed as model plus harness: the model supplies core cognitive capability, while the harness supplies the execution framework that turns that capability into practical effect. The agent is the goal-oriented behavior that emerges from their combination, whereas the harness is the underlying support structure for that behavior LangChain (2026). Third, a harness differs from a traditional pipeline. A traditional pipeline is a predefined static workflow whose control logic is entirely specified by humans. A harness is built around model-mediated dynamic control. It must handle uncertain tool calls, state drift, error recovery, and partial delegation of control to the model rather than reducing execution to a single deterministic path Zaharia et al. (2024).

Harness engineering can therefore be defined as the systematic practice of designing, building, evaluating, and evolving harnesses so that language models can execute real tasks reliably, controllably, and maintainably. Its focus is not primarily to modify the model itself, but to construct the execution environment around the model, specify intent clearly, and build feedback loops that support reliable task execution Anthropic (2024a); OpenAI (2025); Microsoft Security (2026). Its research problems include component architecture, interaction protocols, the dynamic boundary between harness and model capability, and the strategies by which harnesses should evolve as model generations change.

We can now draw a working boundary among model, harness, and environment. The model is responsible for reasoning and content generation. The environment provides the task setting and real feedback. The harness mediates between them, translating model reasoning into controlled and verifiable execution. Within the harness, capability-bearing functions move inward when they become trainable as stable model behavior, while constraint-bearing functions remain outside when their value depends on separability, auditability, and governance Anthropic (2024b); Wang et al. (2025a); Kaptein et al. (2026); Anthropic (2025c). This boundary provides the basis for the remaining analysis. Harness Component Taxonomy decomposes harness component families according to the capability gaps they compensate for. Harness Evaluation begins from those gaps, diagnosing native capability limits before assessing the effectiveness, risks, and net benefit of harness compensation. Harness Evolution then analyzes how the model–harness boundary changes over time under the joint pressure of model improvement and system objectives.

3 Harness Component Taxonomy

An agent is grounded in a recurring loop: the model reads information, chooses an action, the action changes the environment, and the resulting feedback enters the next round of inference. This loop is the minimal

form of sustained agent behavior, but it is insufficient for complex, long-horizon, or high-risk work. The model still depends on external support for limited attention, a closed native action space, non-persistent state, unstable multi-step control, external execution, result verification, and process recording. We map these recurrent gaps to seven component families: context management, tools and interfaces, state and memory, workflow and orchestration, execution environments, verification and guardrails, and observability. A harness surrounds the loop with these compensating structures, allowing the model to act continuously and more reliably in a concrete environment. The resulting design problem is anatomical as well as strategic. Anatomically, different components compensate for different gaps in the agent loop. Strategically, task settings determine which gaps dominate, which components must thicken, and which supports can remain lightweight. This section therefore first decomposes the harness into component families, then examines how design objectives reshape component emphasis, and finally compares representative implementations as different configurations of the same underlying loop. Figure 2 gives an overview of the component families and their associated capability gaps.

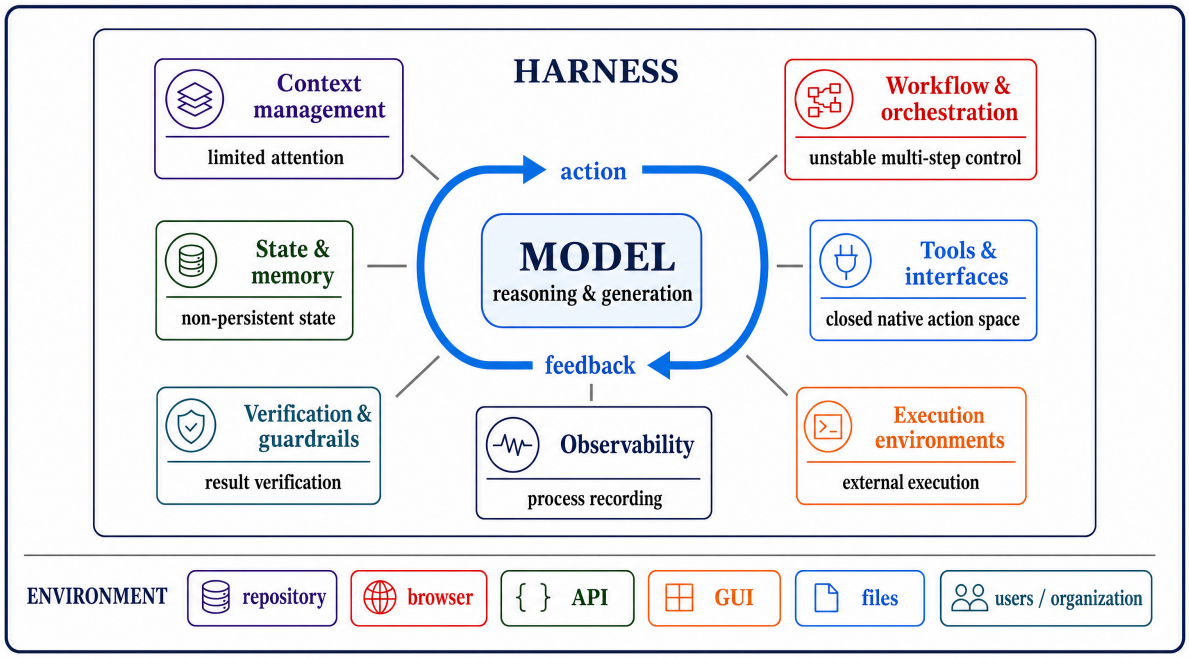

Figure 2: The agent harness as external support structures around the core agent loop. At the center, the model performs reasoning and generation within the recurring *action–feedback* loop: it chooses an action that changes the environment, and the resulting feedback enters the next round of inference. Surrounding the loop are the seven harness component families, each annotated with the recurrent capability gap it compensates for: context management (limited attention), workflow and orchestration (unstable multi-step control), state and memory (non-persistent state), tools and interfaces (closed native action space), verification and guardrails (result verification), observability (process recording), and execution environments (external execution). The bottom layer denotes the task environment the harness mediates access to, including repositories, browsers, APIs, GUIs, files, and users or organizations.

3.1 Context Management

Agent trajectories accumulate information as they proceed. Tool results, dialogue history, drafts, failed attempts, and intermediate outputs all compete for the same context window. Recent evaluations of dynamic agents and long-horizon real-world tasks show that, as trajectories stretch across many tool calls, long execution times, and million-token contexts, models suffer from more than degraded information localization. Planning, memory, and tool management also fail in compound ways. AgencyBench scenarios require roughly 90 tool calls and one million tokens on average, while WildClawBench tasks require approximately

eight minutes of execution and more than 20 tool calls, making these settings closer to agent operating conditions than earlier static long-context benchmarks Li et al. (2026d); Ding et al. (2026b). Once the model drifts in redundant context, later steps become harder to recover Zeng et al. (2026); Li et al. (2026d). A larger context window is therefore not automatically a better one. Context management treats the window as a scarce resource: it asks what the model should see at the current step, where temporarily irrelevant information should go, and how complex tasks can avoid cross-contamination. Around these questions, context management components fall into selection and retrieval, compression and summarization, externalization and offloading, and isolation with progressive disclosure.

Selection and retrieval start from a central constraint: only part of the available material is relevant to the next step. They identify the evidence needed now, rather than inserting all potentially useful material at once. Just-in-time retrieval uses the task goal, current question, and tool state to retrieve necessary information, while leaving other content outside the window as paths, queries, links, or indexes Anthropic (2025c); Mei et al. (2025). If selection is too broad, redundant material dilutes model attention. If it is too narrow, key premises may be omitted. These components therefore work with state and memory components: memory preserves retrievable material, while context management decides which part to retrieve at each step.

Compression and summarization become necessary after useful information has already passed through the window. Some of that information is too important to discard, but too bulky to keep verbatim. Auto-compaction, structured summaries, budget-aware compression, and task-state rewriting can turn long histories into shorter records of goals, decisions, evidence, and pending work LangChain (2025a); Wu et al. (2026d). Their central risk is information loss. Budget-aware context management models compression as a sequential decision problem under an explicit context budget, and long-horizon evaluations show that erroneous compression or omitted premises can later become compound planning, memory, and tool-management failures Wu et al. (2026d); Li et al. (2026d). Safer compression is therefore more than shortening. It preserves recoverable pointers, structured state, and verifiable evidence so that information moved out of the window can be retrieved again when needed.

Externalization and offloading keep useful material available without keeping it in the prompt. Structured notes, draft files, offloaded tool outputs, and file-system caches move large information blocks out of the resident context while retaining references to them Anthropic (2025c;a); Manus (2025). This is useful for information that remains valuable but should not consume attention budget at the current step. The file system plays a double role: it provides an almost unbounded external workspace for context management while also serving execution environments and state-and-memory mechanisms. Externalization is not forgetting. It moves information from linguistic context into a location that tools can revisit.

Isolation and progressive disclosure matter when one task contains multiple branches, assumptions, or capabilities. Isolation gives subagents, subtasks, or separate workspaces their own contexts, preventing failed attempts, irrelevant evidence, or local assumptions from polluting the main task window Anthropic (2025d). Progressive disclosure stores capability descriptions, resources, and operational details in layers and loads them only when required; Agent Skills are a representative example Anthropic (2025b;a). Together, these mechanisms determine when the model should see more information. Isolation controls boundaries between tasks, while progressive disclosure controls how information unfolds inside a single capability. Both depend on orchestration and control flow, because control logic typically decides whether to launch a subtask and when to load a skill.

Context management turns the model's limited attention budget into a schedulable system resource. Selection and retrieval decide what enters the window; compression and summarization reduce historical burden; externalization and offloading preserve recoverable pointers; and isolation with progressive disclosure controls information boundaries across tasks and capabilities. These components do not add model knowledge or replace long-term memory. Their distinctive role is to keep the current reasoning state clean, relevant, and recoverable when long tasks would otherwise be submerged by redundant history, irrelevant tool outputs, and failed branches. The corresponding risk is loss of continuity: if context is compressed, isolated, or offloaded too aggressively, key premises disappear from the active reasoning path.

3.2 Tools and Interfaces

Acting in the world requires more than text generation. To read external information, modify files, call services, or trigger real actions, a harness must translate linguistic intent into executable calls through tools and interfaces. The design problem has three parts: which actions the model may use, how actions are called reliably, and how action capabilities are discovered, organized, and governed at scale. The relevant components include tool-call contracts, agent-computer interface design, tool selection and organization, and interoperability protocols. Together, they define the action space. Execution environments run these actions, while verification and guardrails constrain their side effects Qu et al. (2025); Anthropic (2025e).

Tool-call contracts give an exposed action a precise boundary. They specify whether the action can be called, with what parameters, and how the returned result enters later reasoning. Function calling has moved from a specialized tool-learning capability toward a native feature of mainstream models. The problem has therefore shifted from whether the model can call tools to whether it can call them stably, accurately, and explainably Qu et al. (2025); Anthropic (2025e); Chen et al. (2025b). Vague contracts increase errors in parameter types, enum values, optional-field semantics, and return interpretation. Diagnostic work on multi-agent tool invocation also shows that parameter handling and schema-constraint conflicts are common failure sources Huang et al. (2026a). Tool contracts clarify action boundaries by compressing natural-language intent into a constrained parameter space that the execution environment can run.

Agent-computer interface design addresses a different failure point: a tool may exist, but still be difficult for the model to use correctly. Tool names, descriptions, parameter explanations, and return formats enter the model's decision process, so they operate as control signals rather than ordinary documentation Anthropic (2025e). Interface quality depends on functional completeness, but also on whether actions are sufficiently atomic, names reduce ambiguity, and returns contain only information needed for later reasoning. Recent work on rewriting tool descriptions shows that, at the scale of more than 150 candidate tools, rewritten descriptions reduce accuracy degradation and improve query-level success Guo et al. (2026a). A 2026 study of terminal-style agent interfaces similarly reduces effective human-agent collaboration to representational compatibility, transparency of action, and low barriers for human participation Masi (2026). The role of ACI is therefore not to add more tools. It is to reduce misunderstanding and misuse of the actions already available while making those actions easier for humans to supervise Guo et al. (2026a); Masi (2026).

Tool selection and organization become more difficult as the action space grows. A small tool set can be placed directly in context, but tens or hundreds of tools consume attention budget, weaken selection signals, and increase the likelihood of incorrect choice Lumer et al. (2026). Retrieval-based tool selection recalls tools dynamically according to the current task. Methods that jointly model tool capabilities and agent capabilities can identify appropriate actions more precisely than coarse-grained retrieval Lumer et al. (2025). Beyond selection, call timing is a separate problem: some calls fill information gaps, while others merely add cost, latency, or noise. A harness must therefore judge whether a call is necessary, useful, and affordable Wu et al. (2026a). Skills represent another organizational pattern. They package instructions, resources, and operating modes as capabilities loaded on demand, rather than keeping the full tool set resident in context Anthropic (2025b;a).

Interoperability protocols enter when tools come from many systems rather than one local integration. As agents need access to code repositories, databases, enterprise applications, and external services, one-off integrations become difficult to maintain. Protocols such as MCP provide a uniform interface between AI applications and external data sources, tools, and services, and sit alongside ACP, A2A, ANP, and related interoperability efforts Anthropic (2024b); Ehtesham et al. (2025). These protocols reduce integration cost, but they also foreground tool supply chains, security boundaries, and permission governance. If an external server's tool descriptions, permission scope, or execution capabilities are untrusted, they can influence tool selection and expand side effects Lee et al. (2025). Interoperability is valuable not only because it connects more systems, but also because it exposes connection relationships, capability boundaries, and governance responsibilities more clearly.

The action space of an agent is therefore engineered, not merely appended. Tool-call contracts narrow action boundaries, ACI improves action comprehensibility, tool selection and organization control large-scale capability exposure, and interoperability protocols reduce cross-system integration cost while introducing

governance boundaries. Their distinctive role is to turn linguistic intent into executable action contracts. More expressive tools expand the agent's reachable environment, but more tools, broader permissions, and vaguer interfaces also increase misuse and privilege risk. Tools and interfaces therefore define the agent's capability exposure as much as they expand its task coverage.

3.3 State and Memory

Agent work often outlives a single prompt. After a session reset, prior facts, preferences, decision rationales, and lessons from failed attempts disappear from the visible context. State and memory components persist such information outside model parameters and outside the current context window. Their functions are to record current task progress, save prior facts and experience, and provide reusable evidence for similar tasks Hu et al. (2026b); Du (2026). The relevant components include working memory, episodic and semantic memory, procedural memory, memory management, and engineering primitives such as file systems and version-control systems. Their boundary with context management is functional: context management allocates attention inside the current window, while memory maintains persistent state outside it. The two are connected through write, manage, and read loops Anthropic (2025c); Packer et al. (2024); Du (2026).

Working memory holds the state needed for the next few decisions. It records goals, constraints, the current plan, tool results, intermediate drafts, and unfinished todos. This information is closely coupled to the context window because the next step often depends on it directly Du (2026); Sumers et al. (2024). Too little working memory causes later steps to lose prior reasoning. Too much consumes attention budget and accelerates context decay. A safer pattern keeps only the state required for the current decision in the window, writes the rest to external storage, and retains pointers for recovery Packer et al. (2024); Du (2026). Working memory is therefore the boundary layer between state-and-memory components and context management.

Episodic and semantic memory become relevant when past information should survive beyond the current task state. Episodic memory records temporally ordered interactions, tool returns, user feedback, and task trajectories, allowing an agent to revisit prior decisions in later steps or sessions. Semantic memory stores relatively stable facts, domain knowledge, user preferences, and entity relations, so the system does not rediscover the same information on every task Hu et al. (2026b); Du (2026); Park et al. (2023). Together, they determine how evidence from the past is retrieved. Vector stores are suitable for semantic-similarity recall, while knowledge graphs better express entity relations, multi-hop reasoning, and provenance, though at higher write and maintenance cost Jiang et al. (2026a); Yang et al. (2026). The key is not storing more information, but retrieving information that is relevant, trustworthy, and not overly redundant.

Procedural memory is needed when the system should remember how to perform a class of tasks. It preserves reusable steps, skills, conventions, repair strategies, and task scripts. This helps similar tasks avoid starting from scratch, rather than storing the raw record of a particular interaction Hu et al. (2026b); Fang et al. (2026a). Procedural memory may be distilled from successful trajectories, failed-attempt reflection, or human-curated skill instructions, and later used as callable operational knowledge Fang et al. (2026a); Jiang et al. (2026a). It is complementary to Skills. Skills are runtime capability packages loaded on demand, while procedural memory is an accumulated store of cross-session practice. Without quality filtering, procedural memory can also fossilize bad practices, so it must be updated jointly with verification outcomes and human feedback.

Memory management is the control layer that keeps stored experience usable. It decides what to write, how to organize it, when to retrieve it, and when to forget it. Writing must judge whether information is worth long-term retention instead of turning all logs into noise. Management must consolidate, archive, or forget repeated, stale, or conflicting information. Reading must recall the entries most relevant to the current goal, not reinsert the entire memory store into the context window Hu et al. (2026b); Du (2026). The main risks are under-writing, over-writing, and poor retrieval. These respectively create memory blind spots, retrieval pollution, and context inflation Du (2026); Jiang et al. (2026a). Memory-system quality therefore depends not only on the storage substrate, but also on whether write, manage, and read policies fit the task risk and context budget.

File systems and version control are often the simplest memory substrate for engineering tasks. Many coding agents do not rely on specialized memory databases. They use files, directory structures, logs, and intermediate artifacts as carriers of cross-step state LangChain (2025b); Cao et al. (2026b). This approach is readable, debuggable, and naturally connected to the execution environment. Artifacts left by an agent in the file system become both context for the next round of work and evidence for verification and audit LangChain (2025b); Cao et al. (2026b). Its limits are equally clear: the file system does not natively provide semantic search, conflict resolution, or cross-task generalization, so retrieval, indexing, or tiered storage are still needed at scale LangChain (2025b); Yang et al. (2026).

State becomes useful only when it remains retrievable and trustworthy. Working memory preserves continuity within the current task; episodic and semantic memory provide past evidence and stable knowledge; procedural memory stores reusable practice; memory management controls write and read quality; and file systems plus version control provide auditable engineering substrates. The distinctive gap here is persistence: without state and memory, the agent repeatedly loses task progress, prior evidence, and accumulated practice. Richer memory can support longer-term continuity, but unconstrained writing, management, and retrieval can turn memory into noise, bias, and stale information. Memory is therefore a trust problem as much as a storage problem.

3.4 Workflow and Orchestration

Complex tasks rarely finish in one model call. They unfold across observation, action, branching, and recovery, so isolated model calls must be organized into a process that can keep moving. Workflow and orchestration components provide that process. They decide the next action, decompose tasks, assign models or agents, and organize recovery after failure. The corresponding components include reasoning-action loops, planning and task decomposition, deterministic workflows and hooks, model routing, and subagents and handoffs Huang et al. (2024b); Adimulam et al. (2026). These components depend on context management and verification: context management determines what each step can see, and verification and guardrails determine whether each step can continue.

Reasoning-action loops begin with the need to make progress before the whole answer is known. ReAct interleaves reasoning traces and actions, allowing the model to update later actions after observing environment feedback instead of producing a complete answer in one shot Yao et al. (2023b). Such loops record the relationship between thinking, acting, and observing, so the agent can adjust after tool returns, execution failures, or state changes. Their value lies in decomposing long tasks into short steps that can be observed, interrupted, and verified. As loops grow longer, error, context inflation, and cost accumulate. The loop must therefore be designed together with context compression, verification checks, and stopping conditions Yao et al. (2023b); Huang et al. (2024b).

Planning and task decomposition become necessary when a goal is too large to execute directly. Planning turns an abstract objective into ordered subgoals, and task decomposition further breaks complex tasks into locally executable and locally verifiable subtasks Huang et al. (2024b); Wei et al. (2025). Plan selection weighs tradeoffs among candidate paths, reflection updates plans after failure, and memory-augmented planning incorporates prior experience Huang et al. (2024b); Wei et al. (2025). Long-horizon evaluations in 2026 move this problem from static reasoning puzzles into real tool environments: AgencyBench requires agents to process queries, deliverables, and scoring rules across many real scenarios, while AgentEscapeBench requires agents to infer, execute, and revise novel tool procedures under explicit dependency graphs Li et al. (2026d); Guo et al. (2026b). The conclusion is not that more search is always better. Long-horizon tasks require exploration, dependency tracking, local verification, and termination conditions within the same control flow. Without these constraints, extra steps become token, latency, and error-accumulation costs.

Deterministic workflows and hooks are useful when parts of the process should not depend on free-form model judgment. A deterministic workflow uses a state machine, directed acyclic graph, or explicit step table to specify preconditions, successor actions, and failure branches, making the process easier to inspect before execution and debug afterward Anthropic (2024a); Microsoft (2026). Hooks attach deterministic logic at model calls, tool calls, or state transitions to perform parameter checks, permission decisions, format validation, retries, or compensating actions LangChain (2026); Microsoft (2026). Their role is not to replace

model reasoning, but to provide rails. Stable, repetitive, or high-risk parts are handled by rules, while open and judgment-heavy parts remain with the model.

Model routing matters because not every step needs the same model. In one agent process, intent classification, formatting, deep reasoning, code generation, and long-context analysis may demand different model capabilities. Sending everything to the same model wastes cost or under-provisions capability. Difficulty-aware routing decides whether to upgrade to a stronger model according to request complexity, while capability-aware routing chooses an executor according to task type and model strengths Moslem & Kelleher (2026); Guo et al. (2025). The value of routing is to make model choice a runtime decision that trades off quality, cost, and latency. If routing targets specialized agents rather than models, it enters the subagent and handoff layer.

Subagents and handoffs appear when one context or one role cannot carry the whole task. Subagents can have independent goals, contexts, and tool sets for parallel retrieval, local exploration, role specialization, or high-risk branch isolation Anthropic (2025d). Multi-agent orchestration must usually solve task allocation, result aggregation, and state transfer among agents Anthropic (2025d); Adimulam et al. (2026). Centralized topologies use one orchestrator to assign and collect work; decentralized topologies let agents negotiate peer to peer; hierarchical topologies constrain responsibilities and output forms through superior-subordinate structures Adimulam et al. (2026); Hong et al. (2024). The key handoff tradeoff is context-transfer granularity. Passing raw history is more complete, but costly and contaminating; passing structured summaries is cheaper, but may lose critical evidence Anthropic (2025d).

Workflow and orchestration give the agent a way to advance, stop, and recover. Reasoning-action loops provide the execution rhythm, planning and decomposition provide goal structure, deterministic workflows and hooks provide inspectable control boundaries, model routing schedules capability and cost, and subagents and handoffs provide context isolation and parallel work allocation. The distinctive gap is multi-step control: isolated model calls do not by themselves preserve progress, assign responsibility, or decide when recovery is needed. Orchestration must therefore work with execution feedback, pass criteria from verification and guardrails, and trace evidence from observability.

3.5 Execution Environments

Many agent tasks become meaningful only when an action runs or changes external state. The model can generate commands, code, or operational intent, but completion often requires code execution, file changes, web access, or modified system state. Execution environments translate these intents into observable environmental changes and return runtime results, logs, screenshots, and file diffs to later reasoning and verification. They answer where execution happens, how execution state is preserved, how real interfaces are accessed, and how runtime conditions are prepared. The relevant components include sandboxes, file-system abstractions, browser and computer-use interfaces, and preinstalled runtimes and toolchains. These components receive action contracts defined by tools and interface protocols and provide external feedback signals to verification and guardrails.

Sandboxes are needed because generated actions may have real side effects. Agent-generated code and commands are difficult to audit fully before execution, so a sandbox must manage both isolation strength and resource constraints. Isolation limits contact with the host system, credentials, and network resources. Resource constraints limit CPU, memory, I/O, network egress, and session lifetime Bui (2026); Harang (2026). Containers, user-space kernels, and microVMs represent different tradeoffs between isolation strength and startup cost. Firecracker-style microVMs provide stronger boundaries for high-risk, multi-tenant, or massively parallel execution Agache et al. (2020); Harang (2026). The purpose of a sandbox is not to make execution risk-free. It is to confine model actions to boundaries that are reclaimable, auditable, and interruptible.

File-system abstractions matter once execution leaves artifacts behind. Agents can use the file system to save intermediate artifacts, logs, and generated files, preserving traceable state across execution LangChain (2025b); Cao et al. (2026b). This structure serves three needs at once. It gives context management an external carrier outside the window, gives state and memory a persistent workspace, and gives verification and guardrails comparable runtime artifacts. Recent work shows that coding agents can use terminal tools

and directory navigation to organize large text corpora as operable file structures, outperforming semantic-retrieval baselines on some long-context processing tasks Cao et al. (2026b). The file system is therefore not merely storage. It is an environmental interface shared by execution, memory, and verification.

Browser and computer-use interfaces are needed when the relevant system has no clean API. A browser lets an agent access web pages, read the DOM or accessibility tree, submit forms, and observe page feedback. Computer use further covers desktop GUIs, file managers, terminals, and other applications without stable APIs Ning et al. (2025); Wang et al. (2025b). These environments expand the action space and introduce stronger grounding requirements: the system must map natural-language intent to screen elements, page state, and multi-step interaction sequences. WindowsWorld and Odysseys shift evaluation from single-site or single-application tasks to cross-application, cross-site, and long-horizon real workflows. WindowsWorld reports poor success on multi-application GUI tasks, while Odysseys shows that even the strongest systems reach only 44.5 percent perfect success on long-horizon web tasks Li et al. (2026b); Jang et al. (2026). The key issue is therefore not opening an interface. It is converting interface state into reliable next actions across multiple steps, interfaces, and state dependencies.

Preinstalled runtimes and toolchains shape what the agent can try immediately. Language runtimes, package managers, build tools, test frameworks, version-control tools, and common command-line utilities jointly determine whether an agent can complete the generate-run-test-repair loop Anthropic (2025f); Bui (2026). Insufficient toolchain coverage forces agents to install dependencies on the fly, adding network latency, version conflict, and supply-chain risk. Excessive coverage increases image size, maintenance cost, and attack surface. Long-horizon practice often separates environment initialization from sustained execution. Initialization prepares dependencies, directory structure, and base tools; later execution makes incremental progress within that environment and leaves artifacts, logs, and diffs for the next round Anthropic (2025f).

Execution gives model output external consequences. Sandboxes provide execution boundaries, file systems provide cross-step workspaces, browser and computer-use interfaces provide access to real interfaces, and preinstalled runtimes and toolchains provide closed-loop capability. The distinctive gap is grounded action: without an execution environment, tool calls remain abstract action contracts rather than observable state changes. Yet execution expands both the agent's capability boundary and its side-effect boundary, so greater execution authority must be paired with permission gates, verification loops, and observability.

3.6 Verification and Guardrails

Agent trajectories need feedback about both quality and permission. Without such feedback, models can produce incorrect judgments with high confidence, and single-step errors can propagate through multi-step trajectories. Recent long-horizon evaluations show that real agent tasks cannot be judged only by final answers. WildClawBench checks artifacts, side effects in environment state, and semantic correctness; AgencyBench uses executable scoring scripts and rubrics to map deliverables to comparable scores Li et al. (2026d); Ding et al. (2026b). If critical judgment remains entirely inside the model, the system struggles to distinguish completed work, partial work, plausible-looking errors, and cases where it should not have proceeded. Verification and guardrails move these judgments into the surrounding process. They check whether results are reliable, determine whether actions are allowed, and decide whether high-risk steps need human confirmation. The corresponding components include result verification, deterministic checks, action guardrails, layered defense, and human-in-the-loop control. Execution environments provide tests, logs, and tool returns; observability provides traces and evidence; verification and guardrails use them to pass, retry, block, or escalate.

Result verification starts from a recurring uncertainty: the agent's answer may look complete before the work is correct. In long-horizon tasks, final-answer scores can hide whether the intermediate path truly completed work, omitted side effects, or only partially satisfied the request. Recent evaluations therefore split verification into executable scripts, environment-state audits, artifact checks, and necessary semantic judgment Li et al. (2026d); Ding et al. (2026b). A more stable pattern shifts feedback toward the environment: code interpreters, search engines, rule checkers, test suites, file diffs, and execution logs provide verifiable signals. Result verification does not prove that the full task goal is complete. It separates a portion of formalizable error from model judgment and places error localization closer to the real execution result.

Deterministic checks apply when part of correctness can be expressed as a rule. Linting, type checking, contract assertions, test-suite hooks, structured-output validation, executable scoring scripts, and environment-state audits respectively check formatting, type constraints, interface contracts, runtime results, downstream consumability, and action side effects Li et al. (2026d); Ding et al. (2026b). Their value is not to replace semantic judgment, but to provide low-cost, reproducible negative feedback to multi-step agents. The closer a check is to the real environment, the more specific and expensive it becomes. Engineering practice therefore needs verification step limits, early-exit conditions, and failure fallback strategies, so that an agent does not consume unbounded tokens, calls, and runtime while pursuing local pass rates.

Action guardrails apply before or during an action, where the question is authorization rather than correctness. Guardrails move safety and policy constraints out of prompts and into runtime checks around model calls and tool execution Shamsujjoha et al. (2025); Chennabasappa et al. (2025). Input guardrails block prompt injection, sensitive data leakage, and out-of-domain requests. Output guardrails check hallucination, harmful content, factual consistency, and structured format. Tool-call gates further check tool allowlists, parameter ranges, permission boundaries, idempotence, and reversibility Shamsujjoha et al. (2025); OWASP Gen AI Security Project (2025); Chennabasappa et al. (2025). Code-based guardrails compile some policies into executable rules, bringing policy checks closer to deterministic infrastructure Chennabasappa et al. (2025). Their shared function is to restrict free model action to boundaries that are authorized, auditable, and recoverable.

Layered defense is needed because no single checkpoint sees the whole attack surface. Agent attack surfaces include inputs, model outputs, tool calls, external protocols, and execution environments; a single filter cannot cover them all Su et al. (2026); Shamsujjoha et al. (2025). Layered guardrails place checkpoints at different locations, so if one layer is bypassed, a later layer can still check the new context, action parameters, or execution result Shamsujjoha et al. (2025); Chennabasappa et al. (2025). This structure does not remove risk, because rule quality, changing attack patterns, and expanding tool sets leave blind spots Su et al. (2026); Chennabasappa et al. (2025). Its value is to reduce the probability of single-point failure and turn safety judgment from a one-time prompt constraint into a continuous execution constraint.

Human-in-the-loop control enters where responsibility cannot be delegated fully to automated checks. HITL is not merely a confirmation button. It preserves human judgment at irreversible, destructive, or compliance-sensitive steps OpenAI (2026). Approval gates can pause a flow and wait for human approval, asynchronous supervision can send high-risk actions to a review queue, and tiered intervention can decide by risk score whether to continue automatically, request review, or escalate OpenAI (2026). The effectiveness of human supervision depends on the context supplied by observability. If approvers see only isolated requests without the task goal, tool trajectory, and risk explanation, approval degenerates into a formal click-through OpenAI (2026); Microsoft Security (2026). HITL therefore needs trace records, evidence displays, and organizational rules to carry real responsibility boundaries.

Reliable agent behavior depends on judgments that can be checked outside the model. Result verification supplies correctness signals, deterministic checks provide reproducible local constraints, action guardrails constrain side effects, layered defense reduces single-point failure, and HITL preserves human responsibility for high-risk decisions. The distinctive gap is external judgment: the model may propose or justify an action, but the harness must decide whether the result is correct, authorized, reversible, or worth escalating. The more judgment is externalized, the easier the system becomes to correct, audit, and govern, but latency, cost, and engineering complexity also increase. A harness combines feedforward and feedback to steer the system toward a target state; verification and guardrails supply the negative-feedback function. Whether this feedback works depends on external signals from the execution environment and evidence structures from observability.

3.7 Observability

Failures are difficult to diagnose when only the final output is visible. If an agent's intermediate decisions are not recorded, hallucinations, loops, tool misuse, and cost anomalies are difficult to trace to concrete steps. Observability turns execution into inspectable evidence: it assesses output quality, locates where failures occur, and measures whether resource consumption is acceptable Mohammadi et al. (2025); Liu & Solomon

(2025); Crowder (2026). It includes tracing, evaluation harnesses, and cost and latency instrumentation. These records support verification, guardrails, and human review.

Tracing gives the execution process a timeline. It records model calls, tool calls, retrieval results, environment feedback, subagent handoffs, and final output. The value of structured traces is not saving more logs. It is preserving temporal relationships, causal relationships, and evidence provenance, so failure attribution can move from the whole agent to concrete locations such as planning, tools, context transfer, or evidence use Liu & Solomon (2025); Wang et al. (2026d); Crowder (2026). OpenTelemetry's GenAI semantic conventions bring model calls, tool execution, and agent orchestration into a common tracing framework, and tools such as Langfuse, LangSmith, and Arize Phoenix provide engineering support around trace recording, retrieval, and visualization Crowder (2026). Tracing turns otherwise invisible intermediate processes into comparable, reviewable, and reproducible process evidence.

Evaluation harnesses add judgment to the recorded process. Local evaluation checks whether an individual step is acceptable: whether the chosen tool is reasonable, parameters satisfy interface constraints, output format is consumable downstream, and the step advances its subgoal Wang et al. (2026d); Trivedy (2026); LangChain (2026). Whole-trajectory evaluation checks whether a trajectory is coherent, economical, and evidence-grounded: whether tool-call order satisfies dependencies, whether steps are disconnected, and whether the final answer is based on traceable evidence Wang et al. (2026d); LangChain (2026). These evaluations expose problems hidden by final success rate. An agent may reach a correct answer through a redundant, brittle, or irreproducible path. Another may fail, but leave enough evidence to localize the flaw to tool definitions, context transfer, or control-flow design Wang et al. (2026d); Trivedy (2026).

Cost and latency instrumentation records whether the process is economically sustainable. The total cost of a multi-step agent is usually accumulated across several expensive steps. If the system records only the final output, it is difficult to determine whether cost came from long prompts, redundant tool calls, inefficient model routing, or repeated verification Liu & Solomon (2025); Crowder (2026). Stepwise records of input tokens, output tokens, cache hits, time to first token, full generation latency, and failed retries help researchers compare the real cost of context compression, model routing, and tool design Liu & Solomon (2025); Crowder (2026). Cost and latency are therefore not merely post-deployment operations metrics; they are design dimensions of the harness itself Yang & Zhu (2026); Wu et al. (2026a).

An observable trajectory is easier to compare, debug, and review. Tracing supplies process records, evaluation harnesses provide quality judgments, and cost and latency instrumentation supply resource constraints. The distinctive gap is process evidence: without traces and instrumentation, verification and guardrails can only judge local results, and framework comparisons remain stuck at final success rates. With sufficient evidence, harness problems can be localized to components, steps, and interfaces, providing more concrete inputs for later repair Wang et al. (2026d); Trivedy (2026).

4 Harness Design Tradeoffs Under Multiple Objectives

In practice, harness design is constrained by several goals at the same time. Production agent systems must balance **reliability**, meaning correct output, stable behavior, and controllable error; **cost and latency**, meaning acceptable per-run expense and response time; **autonomy and task coverage**, meaning the range of tasks the agent can advance independently; **safety and side-effect control**, meaning predictable and reversible effects on the external environment; and **long-horizon continuity**, meaning preservation of state and experience across steps and sessions Yang & Zhu (2026); LangChain (2024). A survey of more than 1,300 practitioners reports that 32 percent list output quality as a top barrier to production deployment. Cost ranks below quality, but still shapes call frequency, budget allocation, and sustainable operation at scale LangChain (2024). These objectives constrain each other. Reliability depends on verification steps, which consume latency and cost. Autonomy expands the action space and makes safety control harder. Tool coverage supports long-horizon tasks, but each call adds cost.

These tensions cannot be removed by adding every component at maximum strength. Yang and Zhu's 2026 mathematical analysis of sequential agentic workflows shows that, under simultaneous latency and cost caps, maximizing reliability is a constrained convex optimization problem. The three objectives have an irreducible

Pareto frontier: relaxing latency constraints allows more reasoning tokens for reliability, while tightening cost budgets lowers the reliability ceiling Yang & Zhu (2026). Autonomy and safety are similarly in tension. The larger the action radius and the broader the permissions, the harder it is to statically audit generated actions before execution. Risks such as tool misuse, memory corruption, and cross-agent exploitation also increase Su et al. (2026). Autonomy is not a passive by-product of model capability. It is an architectural parameter that must be set deliberately, and higher autonomy requires thicker safety support Su et al. (2026); Harang (2026). Tool coverage and cost efficiency also conflict. Tool-use decision work in 2026 decomposes whether to call a tool into necessity, utility, and affordability, noting that redundant calls can increase cost, latency, and noise Wu et al. (2026a). Yet in long-horizon tasks that depend on tool coverage to complete task branches, excessive tool pruning causes task failure rather than a small quality decrease Yang & Zhu (2026); Cao et al. (2026b). Harness design is therefore a tradeoff problem under multiple objectives. The priority of objectives determines which capability gap is amplified, and external support thickens where the amplified gap lies.

Reliability becomes difficult when small errors accumulate across a trajectory. Under this priority, the harness must strengthen verification and guardrails, observability, and deterministic workflows inside the orchestration layer. Reliability-oriented harnesses emphasize process constraints: they place verification points before and after critical steps, preserve fallback paths after failure, and record trace evidence for error localization Rabanser et al. (2026); Gupta (2026). Long-horizon benchmarks in 2026 have expanded reliability evaluation from final pass rate to combinations of executable scripts, rubrics, environment-state audits, and semantic checks, showing that correctness judgment must be decomposed into externally verifiable signals Li et al. (2026d); Ding et al. (2026b). Structured tracing records execution as evidence with causal relationships, allowing failure attribution to descend to concrete steps and tool calls rather than remaining at the level of overall success rate Liu & Solomon (2025); Wang et al. (2026d). Deterministic control flow moves repetitive, stable, or high-risk steps out of model judgment and implements inspectable execution constraints through rules and hooks Anthropic (2024a); Microsoft (2026).

Cost and latency pressure begins with repeated small expenses. Context grows, tool calls accumulate, and model calls recur across the trajectory. Under this priority, the harness relies more heavily on context management, tool pruning inside tools and interfaces, and model routing inside workflow and orchestration. Context compression is not only a way to reduce input tokens each round. It dynamically decides which information to retain, compress, or externalize under the remaining context budget, while prioritizing structured state and recoverable pointers LangChain (2025a); Wu et al. (2026d). Tool exposure and call timing must be filtered by necessity, utility, and affordability to reduce the context, latency, and noise cost of redundant tool definitions and ineffective calls Wu et al. (2026a). Model routing sends easier subtasks to lighter models, reducing the proportion of expensive model calls while preserving quality Moslem & Kelleher (2026); Okamoto et al. (2026). Under a fixed reliability target, allocating token budget according to each step's marginal reliability gain, using a water-filling strategy, can outperform uniform allocation Yang & Zhu (2026). However, cost optimization has boundaries. Once compression or pruning removes information or actions required for completion, the cost appears as task failure rather than a small score decrease Yang & Zhu (2026); Cao et al. (2026b).

Autonomy is limited when the agent cannot reach the actions or environments required by the task. Under this priority, support shifts toward tools and interfaces, execution environments, and workflow and orchestration. Tool interfaces and ACI design expand the reachable action space; rewritten tool descriptions can reduce accuracy degradation by 29.23 percent when more than 150 candidate tools are available Guo et al. (2026a), and terminal-interface research emphasizes that interfaces must support both agent operation and human supervision Masi (2026). Execution environments provide code execution, file operations, and browser access, turning model intent into observable environmental change Bui (2026); Harang (2026). Workflow and orchestration organize single-step progress into task decomposition, subagent delegation, error recovery, and termination judgment Huang et al. (2024b); Adimulam et al. (2026). General AgentBench and WildClawBench show that cross-skill, cross-tool, real-runtime tasks significantly amplify degradation relative to specialized short-horizon evaluations, and that a single model can shift substantially across harnesses Ding et al. (2026b); Li et al. (2026h). The bottleneck of autonomy expansion is not the number of components alone. It is whether action interfaces are clear, execution feedback is timely, and runtime boundaries are controlled. Autonomy expansion still has hard constraints: a broader action radius makes side effects harder

to enumerate in advance, so the same objective also requires permission management and action-boundary constraints to keep exploration recoverable Harang (2026); Su et al. (2026).

Safety pressure rises as actions become more consequential. Under this priority, support emphasizes permission gates, layered defense, and HITL within verification and guardrails, together with observability for the trace context needed in human approval. As agent autonomy increases, the attack surface expands from single prompts to tool-call chains, cross-session memory, and multi-agent interaction. Traditional chatbot guardrails do not cover self-prompting across turns, cross-tool linking, or persistent memory risks Su et al. (2026). Permission gates and tool allowlists constrain reachable operations to a least-privilege set OWASP Gen AI Security Project (2025). Layered guardrails distribute checkpoints through execution so that if one layer is bypassed, later layers can still block, reducing single-point failure Shamsujjoha et al. (2025); Chennabasappa et al. (2025). Human approval gates preserve human judgment at irreversible or compliance-sensitive steps, but their effectiveness depends on context from observability. If approvers see only isolated requests without task trajectory and risk explanation, approval becomes formal confirmation OpenAI (2026); Microsoft Security (2026). Stronger safety support increases verification latency and constrains throughput, so safety configuration must be calibrated to concrete risk levels and compliance requirements rather than uniformly stacked on every step.

Long-horizon work fails when earlier state no longer guides later decisions. Under this priority, the harness must strengthen state and memory, together with recoverable structures inside context management. The bottleneck is not storage capacity, but whether causal relationships are preserved. Existing memory systems mainly fail by losing causal and objective information and by accumulating errors introduced by lossy similarity-based retrieval Zhao et al. (2026). Large-scale trajectory analysis of cross-domain long-horizon tasks also shows that planning- and memory-related failures dominate, with consistent degradation patterns across strong model families Wang et al. (2026c). Persistent memory systems store cross-session episodic information, semantic knowledge, and operational experience as retrievable external state, so agents need not rediscover the same information in every task Hu et al. (2026b); Du (2026); Jiang et al. (2026a). File systems provide persistent workspaces; cross-step artifacts become both context for the next round and evidence for verification and audit LangChain (2025b); Cao et al. (2026b). Recoverable session structures divide long tasks into incremental phases with explicit artifacts and state, allowing interrupted sessions to resume from checkpoints Anthropic (2025f); Liu et al. (2026d).

Different objectives therefore thicken different parts of the harness. Reliability strengthens verification, guardrails, and observability. Cost strengthens context management, tool pruning, and model routing. Autonomy strengthens tool interfaces, execution environments, and orchestration. Safety strengthens permission gates, layered defense, and human approval. Long-horizon continuity strengthens state, memory, and recoverable workspaces. Objectives can be combined, but their support requirements often constrain one another: thicker safety support adds latency, broader autonomy expands side-effect boundaries, and lower cost can weaken tool coverage. This structural constraint cannot be eliminated by engineering; it is an intrinsic property of harness design Yang & Zhu (2026). Harness tradeoffs are therefore configuration decisions. Under a given task risk level, resource constraint, and capability boundary, designers decide which gaps external support should carry and which judgments remain with the model. Evaluation evidence can then guide later reconfiguration as objective priorities and model capabilities change.

5 Representative Harness Implementations

Representative systems show how the same observe-act-feedback loop can be configured in substantially different ways. In current agent research and engineering practice, the model often generates a next action from current context. The action changes external state through a tool or execution environment, and the new observation enters the next decision round until the task completes or a stop condition fires Yao et al. (2023b); Anthropic (2024a); Bui (2026). Variation appears most clearly along task horizon, execution boundary, permission radius, state persistence, and governance burden. Repository-level repair scenarios represented by SWE-agent have been further refined by late-2025 and 2026 evaluations of long-horizon software evolution and terminal work. Once similar agents move from isolated issue repair to cross-file evolution, real command-line operation, and multi-step test loops, the bottleneck shifts. Success no longer

depends only on whether the final patch passes; it also depends on interfaces, environments, test feedback, and state preservation Le et al. (2026); Merrill et al. (2026). Codex places tasks into independent cloud sandboxes, completing execution, feedback, and updates inside isolation boundaries OpenAI (2025). Source analysis of Claude Code shows that its core loop calls the model, runs tools, and repeats. Much of the code is concentrated in surrounding permission, compaction, and extension mechanisms Liu et al. (2026d). Hermes Agent uses a resident main agent that operates over time. It accepts tasks, accumulates experience, and distills reusable skills through continuing perception-action-feedback Nous Research (2026). These systems share the same abstract control pattern, but their concrete designs reflect the priority requirements of their task settings.

Repository repair gives the agent a concrete success condition. In SWE-agent-style settings, each run has a goal that can be verified by a test suite, so the primary engineering objective is task completion rate and reliable execution closure. This objective presses the gap into the interface layer and the feedback layer. Success depends directly on whether the agent can locate relevant code, apply an appropriate modification, run tests, and judge success. SWE-agent-like support therefore concentrates on tools and interfaces and verification and guardrails. Newer coding-agent evaluations show that single-issue repair does not cover long-horizon software evolution. SWE-EVO requires agents to make multi-step changes across an average of 21 files under high-level requirements, and the strongest models solve only about a quarter of instances, far below their performance on SWE-bench Verified Le et al. (2026). Terminal-Bench moves evaluation into realistic command-line tasks, emphasizing terminal-native abilities such as file manipulation, system administration, debugging, and research-code reproduction Merrill et al. (2026). The key support in single-issue repair is therefore compact interfaces, immediate test feedback, and executable verification, rather than complex long-term memory. Once the task shifts toward cross-version evolution or terminal-native long-horizon work, state management, environment preparation, and workflow support must thicken as well.

Cloud execution changes the design problem. In Codex-style settings, users submit tasks as requests, and multiple tasks proceed concurrently in isolated environments. The primary engineering objectives are isolation safety and cost control. This setting presses the gap into execution boundaries and context: model-generated code must run without affecting other tasks or production systems, while the token cost of each task must remain controlled. Codex therefore concentrates support in execution environments and context management. Each task runs in an independent environment preloaded with the target repository, making isolation a technical property rather than only a policy document OpenAI (2025). A second issue is repository organization under finite context budget. Concise repository instructions in `AGENTS.md` reduce exploratory context cost by giving the agent stable local guidance before it spends tokens rediscovering project conventions Lopopolo (2026). Codex is therefore not designed around long-term memory accumulation. It is designed to let each request complete inside a clear, reclaimable execution boundary.

Local terminal work exposes the agent to a longer and less isolated development setting. In Claude Code-style use, a user may work on the same codebase within one or multiple sessions, so the primary engineering objectives are context continuity and controllable permissions. This setting presses the gap at both ends. Long sessions accumulate history, tool returns, and intermediate drafts, requiring active compression and persistence to preserve continuity. Local workspace actions directly affect the developer's environment, making side effects less recoverable than in sandboxed settings and requiring earlier permission judgment. Claude Code therefore concentrates support in context management and verification and guardrails. Its five-layer compaction pipeline, append-oriented session storage, and subagent summaries address long-session context growth. Seven permission modes, an ML classifier, and hooks attach deterministic checks around tool calls. Approximately 98.4 percent of the code lives in infrastructure around the main loop Liu et al. (2026d). Compared with SWE-agent, Claude Code also carries heavier workflow and orchestration support. MCP, plugins, Skills, and subagent delegation provide more flexible tool extension and capability loading because long-session tasks require the agent to introduce domain-specific tools and capabilities on demand Liu et al. (2026d).

Resident agents raise a different problem: the system keeps operating after any single task ends. In Hermes Agent-style settings, the primary engineering objectives are cross-session persistence and skill reuse. This setting presses the gap into memory: cross-session state must be retrievable after session reset, and recurring task patterns must be recognized and solidified into reusable skills. Otherwise, the agent cannot become

more efficient as it runs longer. Hermes therefore concentrates support in state and memory and workflow and orchestration. Persistent memory uses structured files for environment facts, learned conventions, and user preferences, injecting them into the system prompt at session start so cross-session baseline state does not disappear with resets Nous Research (2026). A self-improving skill loop extracts reusable patterns after tasks and writes them into structured skill files, allowing similar tasks to load them later rather than reason from scratch Nous Research (2026). Short-lived subagents handle local execution and isolated exploration, while the main agent coordinates, accumulates, and evolves across longer time scales Nous Research (2026). Compared with the other three systems, Hermes also has a stronger need for observability. Continuous operation requires tracking which skills were created and remain effective, which triggered errors, and which should be retired. Observability supplies the evidence base for skill evolution.

The comparison shows how task setting changes component emphasis. Repository-level repair presses the gap into interfaces and feedback, concentrating support in ACI and verification. Cloud parallelism presses the gap into execution isolation and cost, concentrating support in sandboxes and context management. Long-session development presses the gap into window continuity and permissions, concentrating support in compaction and guardrails. Resident autonomy presses the gap into cross-session memory and experience reuse, concentrating support in persistent state and skill evolution. The longer the task duration and the broader the execution permission, the more external support must shift from compact interfaces and verification toward recoverable, reusable memory and evolutionary structure. Where a gap is not prominent, support can remain lightweight to avoid unnecessary complexity Anthropic (2024a); Bui (2026).

6 Harness Evaluation: Diagnosing Model Capability Gaps and Assessing Harness Compensation

The core question in evaluating a harness is whether it can compensate for the capability gaps of a base model. We organize harness evaluation into two complementary dimensions. The first is model capability-gap diagnosis, which assesses the model's native capability limits and identifies where the base model is deficient on its own. The second is harness compensation assessment, which examines whether the harness can effectively fill the corresponding gaps and improve overall agent performance.

6.1 Model Capability-Gap Diagnosis: Assessing Native Capability Limits

Before assessing compensation, we first need to characterize the native capability limits of the base model when it operates without harness support. This section reviews benchmark families that expose gaps across the seven core harness components.

6.1.1 Context and Attention Gaps

Recent long-context evaluations reveal that large models remain limited on long-horizon and long-document tasks. LongBench Pro builds a bilingual benchmark from natural long-form English and Chinese documents with 11 primary tasks and 25 secondary tasks Chen et al. (2026g). Its design varies document length, language setting, dependency range, and reasoning difficulty. The results reinforce an important distinction: the ability to accept long inputs is not equivalent to stable long-document understanding. Task length, dependency span, and reasoning complexity all materially affect performance.

AgentLongBench embeds long-context evaluation into simulated agent-environment rollouts, stressing information retention, state tracking, and dynamic information synthesis over extended trajectories Fang et al. (2026b). The evidence suggests that a model may complete isolated long-text retrieval tasks, yet still struggle to maintain a stable executable state across sustained interaction. A separate orchestration benchmark for 100k-token workflows further shows that frontier models remain prone to next-step decision errors and context-management failures under extreme context pressure Jenova AI (2026). Together, these studies indicate that closed-loop capabilities for state selection, key-content retention, context management, and execution recovery remain underdeveloped.

6.1.2 Tool-Use and Interface Gaps

Tool-use benchmarks consistently expose weaknesses in real user settings, cross-tool coordination, standardized interfaces, and long-horizon multi-tool workflows. WildToolBench evaluates compositional tool-call tasks derived from real user interactions, including implicit intent and dynamic instruction shifts Yu et al. (2026). Its results show that strong performance on standardized single-step tool calls does not generalize to realistic tool-use conditions. As user interaction becomes more complex, tool-selection errors, intent-tracking drift, and multi-tool failures increase.

Protocol-oriented benchmarks make the same limitation visible at larger scale. MCP-Bench evaluates agents against 28 real MCP servers and 250 tool categories, emphasizing cross-tool coordination, precise parameterization, input-output adaptation, and multi-step chain reasoning Wang et al. (2025c). The results suggest that standardized protocol interfaces improve basic tool availability, but models still struggle to compose workflows across servers and satisfy compounded parameter constraints. Toolathlon extends this picture across 32 software applications and more than 600 tool functions, showing that realistic multi-application workflows remain difficult as execution depth and tool diversity increase Li et al. (2026c). MCP-Atlas further broadens the evaluation with 36 real MCP servers and 220 tools, where tool discovery, multi-server workflow composition, parameterization, and error recovery remain frequent failure points Bandi et al. (2026). Overall, single-step calls are often more tractable, whereas real tool use still exposes substantial deficits in interface adaptation and cross-tool orchestration.

6.1.3 State and Memory Gaps

Memory evaluations center on incremental writing, multi-session retention, conflict resolution, continual learning during service time, and memory-enabled decision making. MemoryAgentBench reformulates static long-context data into incremental multi-turn interaction and tests accurate retrieval, test-time learning, long-range understanding, and conflict resolution Hu et al. (2026a). The evidence indicates that models can handle basic pointwise retrieval, but memory stability degrades sharply under more realistic conditions. Performance falls when information arrives continuously, content updates dynamically, or contradictory facts must be merged.

STATE-Bench shifts the emphasis from retrieval accuracy to whether memory mechanisms improve real task execution, including policy validation, user-state calls, tool coordination, and repeated-error avoidance Liu & Yadav (2026). MemoryArena evaluates interdependent multi-session tasks and links memory write, retention, reading, and live decision making in a single loop He et al. (2026b). Other recent benchmarks extend the same pattern. MemoryBench studies continual learning from accumulated user feedback, BEAM examines long-term memory over 100K- to 10M-token conversations, and MemGUI-Bench stresses cross-temporal and cross-spatial memory retention in mobile GUI agents Ai et al. (2026); Tavakoli et al. (2026); Liu et al. (2026b). The common conclusion is that current models still lack robust external state management, especially for write, update, conflict resolution, and memory-driven downstream decision making.

6.1.4 Workflow and Orchestration Gaps

Long-horizon orchestration benchmarks test whether models can maintain a global plan, actively acquire information, respect constraints, track progress, and suppress error accumulation. DeepPlanning evaluates agentic planning under verifiable constraints, including multi-day travel planning and multi-product shopping, where proactive information acquisition and global constrained optimization are both required Zhang et al. (2026d). Odysseys demonstrates that binary pass/fail scoring is insufficient for long-horizon web tasks because it hides partial completion and intermediate execution failures Jang et al. (2026). UltraHorizon further shows that trajectories spanning hundreds of tool calls and more than 200k tokens still expose premature termination, early strategy lock-in, and sustained degradation Luo et al. (2025).

These results suggest that orchestration failures cluster around global plan maintenance, progress tracking, and error recovery under delayed feedback and long execution chains. In such settings, models remain unable to sustain coherent control flow over the full task horizon.

6.1.5 Execution Environment Gaps

Execution-environment benchmarks assess whether text-based decisions can be converted into precise actions in dynamic, verifiable settings. Gaia2 evaluates agents in asynchronous environments that evolve independently of the agent's actions, paired with explicit action verification Froger et al. (2026). FeatureBench extends code-agent evaluation to complex feature development in a constructed and verifiable environment Zhou et al. (2026b). DynamicGUIBench studies dynamic GUI settings in which substantial interface changes occur between actions, stressing visual grounding and action selection under partial observability Liu et al. (2026a). Terminal-Bench 2.0 shows that realistic command-line automation remains far from reliable deployment standards Merrill et al. (2026). Work on deep research and browsing agents pushes the same evaluation logic toward professional, multi-stage workflows with explicit evidence and verification requirements Du et al. (2025); Chen et al. (2025c); Zhang et al. (2026b).

The common conclusion is that models remain unreliable in dynamic, high-interference execution environments. They struggle to verify outcomes continuously and to adapt to environmental state changes, leaving a clear gap in grounded execution.

6.1.6 Verification and Guardrail Gaps

Safety and guardrail evaluations focus on long-horizon attacks, cross-turn intent drift, prompt injection, visual injection, sandbox escape, and other irreversible risks. AgentDyn highlights that harmful intent can be introduced through third-party data and environmental content, not only through explicit user prompts Li et al. (2026a). AgentLAB shows that defenses designed for single-turn interaction do not reliably mitigate long-horizon attacks such as intent hijacking, tool-chain attacks, task injection, and memory poisoning Jiang et al. (2026b). DeepContext further demonstrates that incremental intent drift across turns can bypass static guardrails Albrethsen et al. (2026).

VPI-Bench evaluates visual prompt injection against computer-use agents, showing that embedded malicious instructions in the interface can meaningfully steer behavior Cao et al. (2026a). GhostEI-Bench and AgentHazard extend the security surface to mobile and on-device environments, where misleading UI content and third-party interaction can degrade or misdirect agent behavior Chen et al. (2026a); Liu et al. (2026c). Other benchmarks expand the failure map across execution isolation, mid-trajectory safety, privacy leakage, temporal evasion, and sandbox escape Li et al. (2026j); Chen et al. (2026f); Yagoubi et al. (2026); Ma et al. (2026a); Wang et al. (2026g); Marchand et al. (2026). Overall, current models still lack autonomous correction and dynamic risk control in long-horizon, action-capable settings.

6.1.7 Observability Gaps

Observability evaluation asks whether an agent can produce complete, traceable, and attributable evidence after a failure. TraceElephant shows that full traces can improve attribution accuracy by up to 76 percent relative to partial observations Chen et al. (2026d). MP-Bench argues that failures in multi-agent systems often admit multiple plausible attributions because of complex inter-agent dependencies and ambiguous trajectories In et al. (2026). MAST provides a large failure taxonomy over multi-agent traces and demonstrates that raw logs alone are not enough for diagnosis Cemri et al. (2025). Together, these results show why performance scores alone do not explain why a task failed.

These findings support a broader conclusion: base models and native agent systems still do not generate sufficiently complete process evidence for debugging, root-cause analysis, and regression testing.

6.2 Harness Compensation Assessment: Effectiveness and Net Benefit

Once the native gaps are identified, the next question is whether external harness components compensate for them. This requires two tests: whether compensation is effective, and whether the resulting gain justifies the added system cost.

6.2.1 Effectiveness of Harness Compensation

The evidence suggests that external components can convert some unstable model capabilities into deployable system behavior, but the improvement is usually local rather than complete.

State and memory support directly targets the continuity failures exposed by long-horizon and multi-session evaluations. AMA-Bench shows that memory-system design itself becomes a bottleneck in long-horizon agentic applications, rather than a solved add-on to base-model capability Zhao et al. (2026). LifeBench finds that even top systems achieve only modest accuracy on long-horizon multi-source memory tasks Cheng et al. (2026). M3Exam, H2HMem, RealMem, and MemoryArena further show degradation when models must retain, reconstruct, or reuse information across sessions, parties, and modalities Huang et al. (2026c); Zhu et al. (2026b); Bian et al. (2026); He et al. (2026b). Explicit memory can therefore expand continuity mechanisms, but it does not automatically produce reliable project-level persistence.

Tool and interface support expands the agent's reachable action space, but the evaluation evidence shows that reachability does not guarantee reliable composition. MCP-Atlas broadens the evaluated action space through real MCP servers, but tool selection, parameter configuration, and task understanding remain the dominant failure sources Bandi et al. (2026). FinMCP-Bench shows that single-tool tasks can be handled reasonably well, while multi-tool and multi-turn settings still degrade noticeably Zhu et al. (2026a). MCP-Persona highlights underexploration of tool functions in personalized applications Wang et al. (2026b). GTA-2, Agent-Diff, and the Data Agent Benchmark similarly indicate that richer tool interfaces make more tasks operationally reachable, but do not remove planning errors, state mismatches, or exploration pathologies Wang et al. (2026a); Pysklo et al. (2026); Ma et al. (2026b).

Execution environments and orchestration make longer workflows operational, but they also reveal where state maintenance, protocol compliance, and multi-stage verification remain fragile. General AgentBench shows that performance can drop substantially once systems move from domain-specific settings to general, heterogeneous environments, while General Agent Evaluation complicates this picture by showing that general agents can match strongly customized domain agents on several benchmarks Li et al. (2026h); Bandel et al. (2026). Long-horizon benchmarks then expose where degradation occurs when tasks require sustained interaction. Web, GUI, terminal, software-engineering, data-science, and research-task evaluations all point to recurring bottlenecks in state maintenance, protocol compliance, and multi-stage verification Long et al. (2026); Ding et al. (2026b); Jang et al. (2026); Froger et al. (2026); Liu et al. (2026a); Merrill et al. (2026); Xu et al. (2026d); Desai et al. (2026); Xu et al. (2026a); Chen et al. (2026b); Xu et al. (2026c). The harness can move the failure point, but not eliminate it.

Verification, guardrail, and observability mechanisms shift part of the reliability problem from model judgment to external evidence and control. Long-horizon safety benchmarks show that defenses and isolation are necessary in agent settings with tools, memory, and execution side effects Li et al. (2026a); Jiang et al. (2026b); Li et al. (2026j); Chen et al. (2026f); Yagoubi et al. (2026); Ma et al. (2026a); Wang et al. (2026g); Marchand et al. (2026). At the same time, these mechanisms introduce new surfaces, monitoring burdens, and failure modes. TraceElephant and MP-Bench confirm that richer traces improve debugging and failure attribution, but this is fundamentally a diagnostic benefit rather than a guarantee of fewer failures Chen et al. (2026d); In et al. (2026).

Overall, external harness components provide measurable support. They usually transform model gaps into system-level management problems, rather than eliminating the gaps entirely.

6.2.2 Net Benefit of Harness Compensation

Whether compensation is justified is a stricter question. A harness is warranted only if its gains in reliability, safety, observability, autonomy, or long-horizon continuity outweigh the added complexity, latency, token cost, and operational burden.

At the level of overall architecture, SkillsBench shows that skill augmentation is not a guaranteed gain; its benefit depends strongly on task and environment structure Li et al. (2026g). General AgentBench and General Agent Evaluation likewise indicate that the apparent advantage of a complex harness often shrinks when the evaluation moves into heterogeneous settings Li et al. (2026h); Bandel et al. (2026). In tool-

heavy enterprise tasks, Terminal Agents Suffice for Enterprise Automation suggests that direct terminal/API interaction can outperform heavier tool-augmented stacks in both effectiveness and efficiency Bechard et al. (2026).

The same tradeoff appears in research-oriented workflows. DeepResearch Bench II, MMDeepResearch-Bench, IDRBench, DeepResearch-9K, and BrowseComp-V3 evaluate agents under deeper evidence chains, multimodal grounding, interaction, and verifiable browsing requirements Li et al. (2026e); Huang et al. (2026b); Feng et al. (2026); Wu et al. (2026b); Zhang et al. (2026b). These settings make quality assessment more realistic, but they also increase turns, tokens, and verification cost. Efficient Benchmarking of AI Agents underscores a related point: evaluating advanced agents is itself expensive, so cost must be part of the assessment Ndzomga (2026). This leads to a stricter criterion. A harness is justified only when its additional reliability, safety, observability, autonomy, or continuity benefits consistently outweigh the overhead it introduces.

6.3 Summary

Model capability-gap diagnosis and harness compensation assessment are two tightly linked stages in harness evaluation. Model capability-gap diagnosis decomposes the model's limitations into specific, measurable deficits and gives harness design a concrete target. Harness compensation assessment extends the unit of analysis from the model to the full agent system, testing not only whether external components fill the gap, but also whether the resulting system remains economical and operationally viable.

Together, these two lines of evidence point to the same direction: agent evaluation must move beyond single-number task success and toward a system-level view that balances intrinsic capability, external compensation, deployment cost, and operational risk.

7 Harness Evolution

Although a harness systematically compensates for gaps in a base model's native capabilities, the boundary between the two is not fixed. It is continuously reconstructed as the system evolves. On one side, execution traces, feedback, and historical experience generated by the harness become structured training signals. Through these signals, some externally supported capabilities move closer to model-side behavior. On the other side, improvements in native model capability narrow some of the original gaps. This can make parts of the compensation layer redundant and push the harness to remove obsolete functions, reorganize its architecture, and move toward higher-level system governance. These two forces form a closed loop of coevolution. This section analyzes that pattern in three steps: first, harness-driven model improvement through runtime signals; second, model-driven harness optimization; and third, the evolving model–harness boundary. Figure 3 illustrates this bidirectional coevolution loop and the changing boundary it produces.

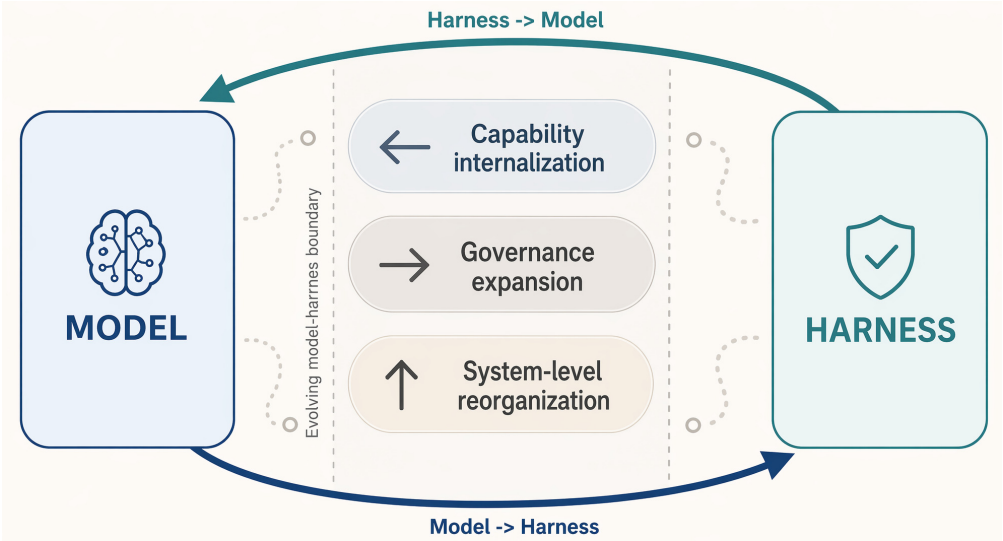

Figure 3: The bidirectional coevolution loop between models and harnesses and its evolving boundary. The upper arc (*Harness* $\rightarrow$ *Model*) denotes harness-driven model improvement, in which runtime trajectories, feedback, and accumulated experience supported by the harness become training signals that move externally supported capabilities toward model-side behavior. The lower arc (*Model* $\rightarrow$ *Harness*) denotes model-driven harness optimization, in which improved native capability drives redesign of the harness. Between the two, the evolving model–harness boundary follows three parallel paths: capability internalization, in which stable, standardized, low-level support migrates inward toward the model; governance expansion, in which the harness thickens its state-management, tool-coordination, security, and governance functions as task complexity, environmental openness, and risk increase; and system-level reorganization, in which scattered and temporary support mechanisms are consolidated into standardized, auditable, and reversible runtime structures. The boundary is therefore layered and dynamic rather than fixed: routine capability-bearing support moves toward the model, while constraint-bearing governance remains external.

7.1 Harness-Driven Model Improvement

The runtime process supported by a harness produces signals through which external support can become model-side capability. During deployment, the harness mediates tool use, environmental perception, state interaction, and error correction. Once these processes are recorded, validated, filtered, and abstracted, they become learning signals for post-training and test-time adaptation. Four signal types are especially important. Demonstration signals provide reproducible execution trajectories and answer how an agent should act effectively. Evaluation signals quantify task quality and define successful outcomes. Interaction signals create dynamic environments for trial and adaptation. Experience signals preserve historical behavior and distill reusable execution patterns.

Demonstration signals come from standardized recording, synthetic construction, and selective filtering of agent trajectories. A harness can integrate target parsing, reasoning, tool selection, and environmental feedback across multiple tool calls into structured reasoning-action, tool-use, or multi-turn interaction traces. These traces support supervised fine-tuning and imitation learning for standardized behavior. Existing work offers several trajectory-construction strategies. ASTRA synthesizes structurally grounded trajectories from tool-call topologies for supervised fine-tuning and reinforcement learning Tian et al. (2026). EigenData generates tool-grounded dialogue data with hierarchical multi-agent data engines while dynamically optimizing prompts and workflows Gao et al. (2026). Tool-R1 converts multi-step tool use into Python-code trajectories with variable dependencies and reuses high-quality samples through a dynamic sample queue Zhang et al. (2025b). DynaWeb combines expert trajectories, imagined rollouts, and real online interaction, thereby broadening the training-sample sources for web agents Ding et al. (2026a).

The value of a demonstration signal depends mainly on the structure and completeness of the trajectory. A trace that contains only the final result supports only outcome-expression learning. A complete trace that covers intermediate actions, real-time feedback, and error repair can teach the model process control for long-horizon tasks. ASTRA, EigenData, and Tool-R1 all indicate that clear boundaries among actions, states, and feedback help turn external execution behavior into standardized model action patterns Tian et al. (2026); Gao et al. (2026); Zhang et al. (2025b).

Evaluation signals consist of verifiers, detection modules, unit tests, reward models, and environmental feedback. They define success criteria, compliant processes, and valid invocation rules, thereby giving model optimization a discriminative target. In practice, EigenData attaches automated checkers to individual samples and converts tool-use tasks into verifier-driven reinforcement learning problems Gao et al. (2026). ASTRA builds code-executable and rule-verifiable simulation environments that constrain training by both completion and efficiency Tian et al. (2026). Agent-RLVR uses unit tests to verify software-engineering trajectories and adds guidance mechanisms for failed retries Da et al. (2025). AgentGym-RL supports multi-turn reinforcement learning in diverse realistic environments, allowing models to optimize directly from environmental rewards Xi et al. (2025).

The design of the evaluation system directly determines the target and convergence direction of model optimization. Different tasks require different validation logic. Code tasks depend on test cases and repository state; tool-use tasks emphasize API accuracy and business effects; web-interaction tasks can be evaluated through page state and trajectory completeness. To stabilize evaluation, Agent World Model, GTM, and LiteResearcher construct simulation environments for code-driven, tool-simulation, and lightweight retrieval settings Wang et al. (2026e); Ren et al. (2025); Li et al. (2026f). Long-horizon reinforcement-learning work further integrates reward shaping, data composition, algorithm choice, and environmental stability into the evaluation design space Wu et al. (2026c). Only an evaluation system with clear boundaries and stable feedback can prevent model capability from drifting away from real task requirements.

Interaction signals are produced by standardized environments that are explorable, fault tolerant, and reproducible. Long-horizon agent tasks involve sequential decisions and partial observation: the model must act under incomplete environmental knowledge, receive feedback, and update its policy. AgentGym-RL decouples the environment, agent, and training module, and uses staged training to balance exploration and exploitation, allowing the model to learn short-interaction behavior before adapting to long-horizon tasks Xi et al. (2025). Agent World Model constructs thousands of SQL-database-backed standardized tool-interaction environments, giving training settings with stable state transitions and high-quality observations Wang et al. (2026e). Other work models interactive tasks as partially observable Markov decision processes and trains agents directly in target environments Chen et al. (2025a), while long-horizon tool-use studies show that environmental stability is a central variable in reinforcement-learning performance Wu et al. (2026c).

Interaction signals are valuable because they allow the model to discover strategies through autonomous trial and error, compensating for the limited coverage of demonstration data. DynaWeb's mixed training over expert traces, imagined rollouts, and online interaction shows that web-agent capability cannot be improved by static samples alone Ding et al. (2026a). AgentGym-RL's staged training mitigates long-horizon instability Xi et al. (2025). Agent-RLVR's teacher-like guidance gives failed trajectories directions for improvement and error attribution, reducing exploration difficulty in complex settings Da et al. (2025). Interaction signals therefore let the model move beyond reproducing existing trajectories and complete an exploration, failure, and correction loop inside a controlled environment.

Experience signals arise from the retention, reflection, compression, and reuse of historical execution trajectories. Trajectories become transferable capabilities only after they are stored, organized, and processed. Existing work summarizes this mechanism from several angles. Agent memory forms a write-manage-read loop that couples perception and action and turns a stateless generator into an adaptive agent Du (2026). Memory evolution can be divided into storage, reflection, and experience, showing a progression from trace retention to experience abstraction Luo et al. (2026). Graph-based memory represents entity relations, hierarchical semantics, and temporal dependencies for multi-turn dialogue, complex planning, and neurosymbolic reasoning Yang et al. (2026). Experiential reflective learning distills trajectories and feedback into general

heuristics that help models reuse experience and avoid repeated errors Allard et al. (2026). In this signal system, the harness manages retention, filtering, association, deletion, and reuse.

Experience signals differ from demonstration signals in their level of abstraction. Demonstration signals focus on the concrete action sequence in a single task, whereas experience signals distill reusable patterns across task families. SIRI integrates scattered experience into discoverable, verifiable, and internalizable skill units He et al. (2026c). MUSE-Autoskill optimizes the full life cycle of skills and enables modular management of external experience Lin et al. (2026a). A systematization of knowledge on agentic skills divides the skill life cycle into discovery, practice, distillation, storage, composition, evaluation, and update, treating skills as long-term experience carriers and harness components Jiang et al. (2026c). Agentic context engineering treats context as an evolving playbook that supports retrieval, compression, isolation, and reuse of experience Zhang et al. (2026c). Complementary work identifies additional routes. Test-time self-improvement can update the model from newly generated task-adjacent data, while just-in-time reinforcement learning can use experience at runtime without gradient updates Acikgoz et al. (2025); Li et al. (2026i). Experience signals make the harness a buffer layer for capability internalization: experience is first stored externally as memory or skills, then progressively feeds model improvement through retrieval augmentation, distillation, test-time adaptation, and related mechanisms.

Together, demonstration, evaluation, interaction, and experience signals constitute the main channels through which a harness transfers external capability toward the model. They are usually combined in supervised fine-tuning, reinforcement learning, retrieval augmentation, and continual learning. The value of the harness is therefore not limited to improving deployment-time performance. It also reveals a pathway for selecting, retaining, abstracting, and retraining from runtime experience. External support that is fixed, repetitive, and structurally stable is easiest to internalize. Open-ended, state-dependent, safety-critical long-horizon tasks still require the harness to provide environmental support, compliance verification, memory management, and risk governance.

7.2 Model-Driven Harness Optimization

The second half of the model–harness coevolution loop runs from model improvement back to harness reconstruction. As model capabilities change, the harness must adjust the assumptions it makes about model behavior, the interfaces through which the model acts, the components that remain necessary, and the governance mechanisms that constrain adaptation. In this sense, model-driven harness optimization refers to harness redesign under changing model capability boundaries. Drawing together recent work on harness adaptation, configuration search, meta-harness optimization, and retrospective self-improvement, we analyze this process at four levels: the interface layer, component layer, workflow layer, and closed-loop layer Lin et al. (2026b); Sengupta & Wang (2026); Lee et al. (2026b); Pan et al. (2026b).

Interface-layer optimization focuses on the interaction paradigm between the model and external systems. It is a central mechanism for changing agent performance when model weights are fixed. Runtime harness adaptation shows that changing the interface alone can substantially affect behavior Xu et al. (2026b). Human-oriented API documentation often does not fit automated model use, so tool interfaces must be rewritten around model cognition Guo et al. (2026a). Meta-Harness shows that historical traces and scores can be used to optimize multiple harness-level choices, including prompts and other interaction specifications Lee et al. (2026b). Context engineering extends this view by treating write, selection, compression, and isolation as design variables and by defining quality criteria such as relevance, sufficiency, isolation, economy, and provenance LangChain (2025a); Anthropic (2025c); Vishnyakova (2026). Additional engineering discussions frame the harness layer itself as a reportable and optimizable scientific object Pan et al. (2026a); He et al. (2026a).

Component-layer optimization decomposes the harness into modules with independent functions and clear responsibilities. Agentic harness engineering treats observable harness artifacts as adaptation surfaces, including prompts, tools, middleware, skills, and configuration-like choices Lin et al. (2026b). HARBOR formulates configuration optimization as constrained noisy Bayesian optimization over a mixed-variable space, jointly considering cost and benefit Sengupta & Wang (2026). ARTEMIS uses semantically aware genetic operators for joint optimization of multidimensional agent configurations Brookes et al. (2025). Surveys and

systematizations further clarify that modern agent systems include reusable skills, memory-management mechanisms, and other iterative components that can be governed or optimized at the harness level Jiang et al. (2026c); Lam et al. (2026); Tang et al. (2026). Effective component optimization depends on observability and rollback boundaries, so that local changes can be evaluated, attributed, and reused independently.

Workflow-layer optimization focuses on the coordination logic among components. Workflow optimization surveys abstract agent workflows as agent computation graphs and distinguish static templates, pre-generated architectures, and dynamically revised runtime graphs Yue et al. (2026). AFlow searches for effective workflows through code modification and execution feedback Zhang et al. (2025a). Meta-tool optimization mines recurring tool-call sequences from historical traces and turns them into meta-tools Abuzakuk et al. (2026). JudgeFlow uses block-level judging to assign responsibility scores to workflow modules and localize failures Ma et al. (2026c). MASPO further optimizes multi-agent workflows from system-wide prompts Wang et al. (2026f).

As tasks become more complex, fixed workflows become insufficient and the workflow architecture itself becomes a primary optimization object. Recent work extends this level toward modular coordination and system-level search. AgentFlow coordinates planner, executor, verifier, and generator modules through evolving memory and optimizes the planner inside the multi-turn loop Li et al. (2025). MASS searches over block-level prompts, workflow topology, and workflow-level prompts in stages, reducing the cost of multi-agent design search Zhou et al. (2026a). MAPRO formulates multi-agent prompt optimization as maximum a posteriori inference and solves it through language-guided belief propagation Zhang et al. (2025c). Workflow optimization surveys provide a unified taxonomy by distinguishing methods that fix a reusable scaffold before deployment from those that select, generate, or revise workflows before or during execution Yue et al. (2026).

Closed-loop optimization upgrades the harness from a single-run support mechanism into a continuously self-improving system. Meta-Harness allows agents to read historical configurations, scores, and traces and propose harness updates Lee et al. (2026b). Agentic harness engineering uses state observations to make evolution detectable, attributable, and reversible Lin et al. (2026b). Retrospective harness optimization uses only past trajectories, self-preference, and consistency checks to update the system without human labels Pan et al. (2026b). Additional studies repair harness flaws from failed trajectories Chen et al. (2026c), while continual-harness frameworks emphasize diagnosis, modification, and evaluation during online adaptation Seong et al. (2026); Karten et al. (2026).

The key advance at this level is that the harness becomes a meta-system capable of recording data, diagnosing failures, changing architecture, and constraining update direction through effect evaluation. In this form, closed-loop optimization is not only an efficiency technique; it also requires update admission, rollback boundaries, and validation criteria so that self-revision remains attributable and governable.

The four levels form a progression from interface, to functional module, to execution architecture, to autonomous evolution. Stronger models can help diagnose system failures and suggest architectural changes, but evolving memory, self-evolving agents, tool use, and autonomous behavior also introduce governance risks such as misevolution and safety vulnerabilities Lam et al. (2026); Shao et al. (2026); Doshi et al. (2026); Bhardwaj (2026). Observability, permission control, version rollback, and effect validation are therefore required mechanisms for adaptive harnesses, not optional additions.

The main result of model-driven harness reconstruction is not the weakening of the harness. Rather, the harness evolves from a deployment-time support layer into a more complex and more strictly governed adaptive system. The model–harness boundary changes, but system-level governance becomes more important.

7.3 The Evolving Model-Harness Boundary

The coevolution of model and harness continuously adjusts their functional boundary. This evolution follows three parallel paths: capability internalization, governance expansion, and system-level reorganization.

7.3.1 Capability Internalization

Capability internalization means that stable and standardized low-level support capabilities move closer to the model, reducing some burden on the harness. Capabilities with clear action spaces, stable feedback, and repeatable sampling are easiest to consolidate through post-training. AgentGym-RL uses multi-environment interaction and staged training to move models from short interactions toward long-horizon decision making Xi et al. (2025). Hindsight credit assignment decomposes sparse terminal rewards into finer process supervision for long-horizon agents Tan et al. (2026). In-distribution optimization combines automatic process-reward labeling with policy learning, enabling faster iteration of multi-turn execution behavior Zhang et al. (2026e).

Stable training environments and reward design are preconditions for internalization. Micro-level execution patterns that once depended on hand-written prompts or fixed rules can increasingly be optimized as model policies when credit assignment, reward shaping, and environment design are well specified Zhang (2026); Wu et al. (2026c). Local internalization, however, does not imply that the whole harness becomes weaker. Benchmarks for command-line, extremely long-horizon, and constraint-heavy planning tasks show that cross-context and cross-tool execution remains difficult for current agents Merrill et al. (2026); Luo et al. (2025); Zhang et al. (2026d). In such settings, environment simulation and standardized tool protocols provide external structure for evaluating and operating complex toolchains Wang et al. (2026b); Zhu et al. (2026a). Internalization therefore applies mainly to low-level local routines. In complex tasks, the harness remains the main carrier that links model capability and reliable execution.

7.3.2 Governance Expansion

Governance expansion means that as task complexity, environmental openness, and risk level increase, the harness expands its functional boundary in state management, tool coordination, security, and governance.

In memory and state management, the bottleneck of long-horizon tasks is not only context length, but also structured control over history, experience, and execution state. AMA-Bench argues that suboptimal memory-system design, rather than base model capability alone, can become a major bottleneck in agentic applications Zhao et al. (2026). AMemGym separately shows that long-horizon assistant memory requires interactive evaluation rather than only static recall tests Jiayang et al. (2026). Multimodal-memory benchmarks further indicate that realistic user-agent interactions require memory mechanisms beyond text-only static recall Huang et al. (2026c). Long-term memory management can therefore be understood partly as execution-state management: the harness provides the external layer for preserving, validating, and writing back state Du (2026); Chen et al. (2026e); Rafique & Bindschaedler (2026).

In tool-ecosystem security, protocols such as MCP standardize tool access but do not eliminate risks inside the invocation process. Open tool ecosystems create large cross-stage attack surfaces Zong et al. (2026); Zhang et al. (2026a). Tool-document poisoning and runtime manipulation of tool identity can lead to tool hijacking Liu et al. (2026e); Lee et al. (2026a). Because these risks arise at the boundary among model outputs, tool metadata, permissions, and runtime state, they motivate security controls outside model optimization alone, including trust boundaries, privilege control, and state governance in the harness Ling et al. (2026).

In system governance, stronger models with broader action spaces increase the need for permission control, process auditing, and failure recovery. Governance shifts from single-output control to path-level control. ContextCov, AgentSpec, runtime governance, AARM, Agent Behavioral Contracts, and SafeHarness provide executable constraints, runtime enforcement, action-boundary management, formal contracts, and life-cycle security for agent deployment Sharma (2026); Wang et al. (2025a); Kaptein et al. (2026); Errico (2026); Bhardwaj (2026); Lin et al. (2026d). Static rules cannot cover such complex scenarios by themselves; the harness must supply an independent governance layer. In embodied and policy-constrained settings, governance also separates model cognition from execution oversight through policy checking, capability admission, monitoring, rollback, and human override Qin et al. (2026).

7.3.3 System-Level Reorganization

System-level reorganization means that scattered and temporary low-level support mechanisms are reorganized into standardized, auditable, and reversible high-level mechanisms. The result is not a matter of functional addition or deletion, but a more disciplined system layer.

Natural-language agent harnesses externalize control logic into editable artifacts, moving design from implicit code conventions to explicit system architecture Pan et al. (2026a). Code-as-harness work treats sandboxes, verification mechanisms, and permission boundaries as core foundations, making external support an engineerable system artifact Ning et al. (2026). Agentic harness engineering and retrospective harness optimization further show how components can evolve automatically and in closed loops without human labels Lin et al. (2026b); Pan et al. (2026b). These developments suggest a shift from scattered functional patching toward more explicit and auditable runtime-system construction.

In this process, the model can participate in failure diagnosis, pattern induction, and optimization proposal generation. Studies of harness benefit, misevolution, and evolving memory suggest that controllable evolution depends on explicit mechanisms for update admission, effect evaluation, rollback or recovery, and governance authority at the harness level Lin et al. (2026c); Shao et al. (2026); Lam et al. (2026).

Across these three paths, model and harness form a dynamic layered boundary rather than a fixed separation. Recent systems show that the model increasingly adapts to and consolidates localized, repetitive, and stable abilities, such as routine tool invocation and local plan repair. At the same time, harnesses carry long-cycle, cross-context, and high-risk functions, such as state management across contexts, permission control, audit trails, and long-term memory evolution. These functions are persistent, complex, and risk sensitive, making them difficult to fully internalize into model parameters.

The evolving model–harness boundary is therefore not a path toward mutual replacement. It follows a layered logic: low-level execution capability moves toward model internalization, while high-level governance responsibility concentrates in the harness. System-level reorganization makes this shifting boundary auditable, reversible, and governable. Together, these changes reveal how agent systems become easier to train, more reliable to operate, and easier to govern.

8 Open Challenges and Future Directions

Building on the preceding synthesis, we identify six future directions for harness engineering. These directions extend the paper's core view of harnesses as external support structures: harnesses should become explicit research objects, comparable design artifacts, gap-indexed evaluation targets, training-signal generators, governable adaptive systems, and components of an automated model–harness coevolution loop.

8.1 Harness as a First-Class Scientific Object

Harness engineering should be studied as an independent object of inquiry rather than as a secondary implementation detail of model work. A useful research program would define stable abstractions, terminology, and design principles for component composition, just as model research standardized architectures and training objectives. The goal is to move from ad hoc engineering tricks to reusable knowledge about how support structures map to capability gaps Lin et al. (2026b); Pan et al. (2026a).

8.2 Making Harness Design Explicit

Much of today's harness logic still lives in controller code, prompt fragments, tool wrappers, and runtime conventions. This makes systems hard to compare, ablate, and transfer. A more explicit representation of the harness should surface stage structure, role separation, artifact contracts, state fields, failure categories, and verification thresholds so that the same design can be inspected across different systems Pan et al. (2026a); Lin et al. (2026b).

8.3 Evaluating Harness Compensation

End-to-end success metrics conflate base-model ability and harness support. Future evaluation should therefore index results by capability gap and component family, and should ask which support reduced which residual failure. Counterfactual evaluation, component-level attribution, and structured error taxonomies would make it possible to judge not only whether a system works, but also whether the harness is appropriately sized for the gap it is meant to cover Yang & Zhu (2026); Wang et al. (2026d); Lin et al. (2026b).

8.4 Designing Harnesses for Training

Harnesses should also be designed as sources of training signal, not only as inference-time scaffolds. By shaping trajectories, feedback, and reward channels, a harness determines which behaviors are internalized and which remain external. The challenge is to identify support structures that produce transferable execution capability rather than narrow prompt-specific behavior Tian et al. (2026); Da et al. (2025); Xi et al. (2025).

8.5 Self-Optimizing but Governable Harnesses

Harnesses will increasingly optimize prompts, tools, workflows, memory, and skills on their own. That makes governance part of the design problem rather than an afterthought. Future systems need update admission, effect evaluation, rollback, and permission boundaries so that self-modification remains auditable and reversible. In this view, capability-bearing components may evolve inward, but constraint-bearing components must remain subject to external control Lee et al. (2026b); Pan et al. (2026b); Lin et al. (2026c); Shao et al. (2026); Lam et al. (2026).

8.6 Toward Automated Model–Harness Coevolution

The deeper goal is an automated coevolution loop in which the system can decide which gaps should be internalized by the model and which should remain compensated by the harness. That requires the model–harness boundary itself to become observable, comparable, and recoverable. When this boundary moves, the system should be able to reconfigure interfaces, preserve evidence, and roll back unsafe changes without losing continuity. Such a loop would connect explicit design, gap-aware evaluation, training-time signal design, and governance into a single adaptive process Lin et al. (2026b;c); Bhardwaj (2026); Kaptein et al. (2026).

9 Conclusion

We have framed harness engineering as a problem of allocating capability, compensation, and governance in LLM agent systems. Rather than treating tools, memory, guardrails, and other modules as isolated additions, this perspective relates them to the capability gaps they address, the tradeoffs they introduce, and the reasons some forms of support remain external even as models improve. Across the three levels developed in this survey, Harness Component Taxonomy organizes gap-indexed component families, Harness Evaluation distinguishes native capability-gap diagnosis from compensation effectiveness and net benefit assessment, and Harness Evolution explains why model improvement reorganizes external support rather than simply eliminating it. Routine capability-bearing support may move toward the model when it becomes trainable as stable behavior, whereas constraint-bearing support remains external because it provides independent verification, permission control, and governance. As agents operate over longer horizons with broader tool access and higher autonomy, harness design therefore becomes a first-class engineering problem concerned with where capability should reside, how external support should be configured, and how the model–harness boundary should evolve.

References

Sami Abuzakuk, Anne-Marie Kermarrec, Rishi Sharma, Rasmus Moorits Veski, and Martijn de Vos. Optimizing agentic workflows using meta-tools, 2026. URL https://arxiv.org/abs/2601.22037.

Emre Can Acikgoz, Cheng Qian, Heng Ji, Dilek Hakkani-Tür, and Gokhan Tur. Self-improving llm agents at test-time, 2025. URL https://arxiv.org/abs/2510.07841.

Apoorva Adimulam, Rajesh Gupta, and Sumit Kumar. The orchestration of multi-agent systems: Architectures, protocols, and enterprise adoption, 2026. URL https://arxiv.org/abs/2601.13671.

Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*, pp. 419–434, 2020.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memorybench: A benchmark for memory and continual learning in llm systems, 2026. URL https://arxiv.org/abs/2510.17281.

Justin Albrethsen, Yash Datta, Kunal Kumar, and Sharath Rajasekar. Deepcontext: Stateful real-time detection of multi-turn adversarial intent drift in llms, 2026. URL https://arxiv.org/abs/2602.16935.

Marc-Antoine Allard, Arnaud Teinturier, Victor Xing, and Gautier Viaud. Experiential reflective learning for self-improving llm agents, 2026. URL https://arxiv.org/abs/2603.24639.

Anthropic. Building effective ai agents, 2024a. URL https://www.anthropic.com/research/building-effective-agents. Official Anthropic research note.

Anthropic. Model context protocol, 2024b. URL https://modelcontextprotocol.io/docs/getting-started/intro. Official MCP overview.

Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku, 2024c. URL https://www.anthropic.com/news/3-5-models-and-computer-use. Anthropic news, October 22, 2024.

Anthropic. Introducing advanced tool use on the claude developer platform, 2025a. URL https://www.anthropic.com/engineering/advanced-tool-use. Official Anthropic engineering post.

Anthropic. Equipping agents for the real world with agent skills, 2025b. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills. Official Anthropic engineering post.

Anthropic. Effective context engineering for ai agents, 2025c. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents. Official Anthropic engineering post.

Anthropic. How we built our multi-agent research system, 2025d. URL https://www.anthropic.com/engineering/multi-agent-research-system. Official Anthropic engineering post.

Anthropic. Writing effective tools for agents – with agents, 2025e. URL https://www.anthropic.com/engineering/writing-tools-for-agents. Official Anthropic engineering post.

Anthropic. Effective harnesses for long-running agents, 2025f. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Official Anthropic engineering post, published November 26, 2025.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback, 2022. URL https://arxiv.org/abs/2204.05862.

Elron Bandel, Asaf Yehudai, Lilach Eden, Yehoshua Sagron, Yotam Perlitz, Elad Venezian, Natalia Razinkov, Natan Ergas, Shlomit Shachor Ifergan, Segev Shlomov, Michal Jacovi, Leshem Choshen, Liat Ein-Dor, Yoav Katz, and Michal Shmueli-Scheuer. General agent evaluation, 2026. URL https://arxiv.org/abs/2602.22953.

Chaithanya Bandi, Razvan-Gabriel Dumitru, Ben Hertzberg, Divyansh Agarwal, Geobio Boo, Tejas Polakam, Sami Hassaan, Jeff Da, HiJae Kim, Vipul Gupta, Manasi Sharma, Andrew Park, Martin Dimakis, Ernesto Gabriel Hernandez Montoya, Dan Rambado, Ivan Salazar, Rafael Cruz, Mohammad-Hossein Rezaei, Chetan Rane, Ben Levin, Daniel Yue Zhang, Brad Kenstler, and Bing Liu. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers, 2026. URL https://arxiv.org/abs/2602.00933.

Patrice Bechard, Orlando Marquez Ayala, Emily Chen, Jordan Skelton, Sagar Davasam, Srinivas Sunkara, Vikas Yadav, and Sai Rajeswar. Terminal agents suffice for enterprise automation, 2026. URL https://arxiv.org/abs/2604.00073.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pp. 610–623, 2021.

Varun Pratap Bhardwaj. Agent behavioral contracts: Formal specification and runtime enforcement for reliable autonomous ai agents, 2026. URL https://arxiv.org/abs/2602.22302.

Haonan Bian, Zhiyuan Yao, Sen Hu, Zishan Xu, Shaolei Zhang, Yifu Guo, Ziliang Yang, Xueran Han, Huacan Wang, and Ronghao Chen. Realmem: Benchmarking llms in real-world memory-driven interaction, 2026. URL https://arxiv.org/abs/2601.06966.

Paul Brookes, Vardan Voskanyan, Rafail Giavrimis, Matthew Truscott, Mina Ilieva, Chrystalla Pavlou, Alexandru Staicu, Manal Adham, Will Evers-Hood, Jingzhi Gong, Kejia Zhang, Matvey Fedoseev, Vishal Sharma, Roman Bauer, Zheng Wang, Hema Nair, Wei Jie, Tianhua Xu, Aurora Constantin, Leslie Kanthan, and Michail Basios. Evolving excellence: Automated optimization of llm-based agents, 2025. URL https://arxiv.org/abs/2512.09108.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL https://arxiv.org/abs/2005.14165.

Nghi D. Q. Bui. Building effective ai coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned, 2026. URL https://arxiv.org/abs/2603.05344.

Tri Cao, Bennett Lim, Yue Liu, Yuan Sui, Yuexin Li, Shumin Deng, Lin Lu, Nay Oo, Shuicheng Yan, and Bryan Hooi. Vpi-bench: Visual prompt injection attacks for computer-use agents, 2026a. URL https://arxiv.org/abs/2506.02456.

Weili Cao, Xunjian Yin, Bhuwan Dhingra, and Shuyan Zhou. Coding agents are effective long-context processors, 2026b. URL https://arxiv.org/abs/2603.20432.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail?, 2025. URL https://arxiv.org/abs/2503.13657.

Chiyu Chen, Xinhao Song, Yunkai Chai, Yang Yao, Haodong Zhao, Lijun Li, Jie Li, Yan Teng, Gongshen Liu, and Yingchun Wang. Ghostei-bench: Do mobile agents resilience to environmental injection in dynamic on-device environments?, 2026a. URL https://arxiv.org/abs/2510.20333.

Haolin Chen, Deon Metelski, Leon Qi, Tao Xia, Joonyul Lee, Steve Brown, Kevin Riley, Frank Wang, T. Y. Alvin Liu, Hank Capps MD, Zeyu Tang, Xiangchen Song, Lingjing Kong, Fan Feng, Tianyi Zeng, Zhiwei Liu, Zixian Ma, Hang Jiang, Fangli Geng, Yuan Yuan, Chenyu You, Qingsong Wen, Hua Wei, Yanjie Fu, Yue Zhao, Carl Yang, Biwei Huang, Kun Zhang, Caiming Xiong, Sanmi Koyejo, Eric P. Xing, Philip S. Yu, and Weiran Yao. Chi-bench: Can ai agents automate end-to-end, long-horizon, policy-rich healthcare workflows?, 2026b. URL https://arxiv.org/abs/2605.16679.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. Reinforcement learning for long-horizon interactive llm agents, 2025a. URL https://arxiv.org/abs/2502.01600.

Mengzhuo Chen, Junjie Wang, Zhe Liu, Yawen Wang, Haiming Zheng, and Qing Wang. From failed trajectories to reliable llm agents: Diagnosing and repairing harness flaws, 2026c. URL https://arxiv.org/abs/2606.06324.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems, 2026d. URL https://arxiv.org/abs/2604.22708.

Yaoqi Chen, Haibin Lai, Yuru Feng, Chuyu Han, Qianxi Zhang, Baotong Lu, Menghao Li, Xinjiang Wang, Zhirui Wang, Shusen Xu, Zengzhong Li, Zewen Jin, Hao Wu, Cheng Li, and Qi Chen. Beyond semantic organization: Memory as execution state management for long-horizon agents, 2026e. URL https://arxiv.org/abs/2606.06090.

Yen-Shan Chen, Sian-Yao Huang, Cheng-Lin Yang, and Yun-Nung Chen. Tracesafe: A systematic assessment of llm guardrails on multi-step tool-calling trajectories, 2026f. URL https://arxiv.org/abs/2604.07223.

Yi-Chang Chen, Po-Chun Hsu, Chan-Jan Hsu, and Da-shan Shiu. Enhancing function-calling capabilities in llms: Strategies for prompt formats, data integration, and multilingual translation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pp. 99–111, 2025b.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent, 2025c. URL https://arxiv.org/abs/2508.06600.

Ziyang Chen, Xing Wu, Junlong Jia, Chaochen Gao, Qi Fu, Debing Zhang, and Songlin Hu. Longbench pro: A more realistic and comprehensive bilingual long-context evaluation benchmark, 2026g. URL https://arxiv.org/abs/2601.02872.

Zihao Cheng, Weixin Wang, Yu Zhao, Ziyang Ren, Jiaxuan Chen, Ruiyang Xu, Shuai Huang, Yang Chen, Guowei Li, Mengshi Wang, Yi Xie, Ren Zhu, Zeren Jiang, Keda Lu, Yihong Li, Xiaoliang Wang, Liwei Liu, and Cam-Tu Nguyen. Lifebench: A benchmark for long-horizon multi-source memory, 2026. URL https://arxiv.org/abs/2603.03781.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. Llamafirewall: An open source guardrail system for building secure ai agents, 2025. URL https://arxiv.org/abs/2505.03574.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023. URL https://arxiv.org/abs/1706.03741.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1173–1203, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.65. URL https://aclanthology.org/2024.acl-long.65/.

Sam Crowder. How to debug and evaluate ai agents with observability, 2026. URL https://www.langchain.com/blog/agent-observability-powers-agent-evaluation. LangChain blog.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. Agent-rlvr: Training software engineering agents via guidance and environment rewards, 2025. URL https://arxiv.org/abs/2506.11425.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2026. URL https://arxiv.org/abs/2501.12948.

Rishi Desai, Jesse Hu, Joan Cabezas, Neel Harsola, Pratyush Shukla, Roey Ben Chaim, Adnan El Assadi, Omkaar Mukund Kamath, Fenil Faldu, Prannay Hebbar, Jiankai Sun, Yiyuan Li, Pramod Srinivasan, Ishan Gupta, Christopher Settles, Daniel Wang, Derek Chen, Pranav Raja, Albert Liu, Marek uppa, Nevasini Sasikumar, Luyang Kong, Erik Quintanilla, Xiangyi Li, Ivan Bercovich, and Steven Dillmann. Swe-marathon: Can agents autonomously complete ultra-long-horizon software work?, 2026. URL https://arxiv.org/abs/2606.07682.

Hang Ding, Peidong Liu, Junqiao Wang, Ziwei Ji, Meng Cao, Rongzhao Zhang, Lynn Ai, Eric Yang, Tianyu Shi, and Lei Yu. Dynaweb: Model-based reinforcement learning of web agents, 2026a. URL https://arxiv.org/abs/2601.22149.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Yang JingYi, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, Yubo Ma, Haodong Duan, Jing Shao, Jiaqi Wang, Dahua Lin, Kai Chen, and Yuhang Zang. Wildclawbench: A benchmark for real-world, long-horizon agent evaluation, 2026b. URL https://arxiv.org/abs/2605.10912.

Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. Towards verifiably safe tool use for llm agents, 2026. URL https://arxiv.org/abs/2601.08012.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench: A comprehensive benchmark for deep research agents, 2025. URL https://arxiv.org/abs/2506.11763.

Pengfei Du. Memory for autonomous llm agents:mechanisms, evaluation, and emerging frontiers, 2026. URL https://arxiv.org/abs/2603.07670.

Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. A survey of agent interoperability protocols: Model context protocol (mcp), agent communication protocol (acp), agent-to-agent protocol (a2a), and agent network protocol (anp), 2025. URL https://arxiv.org/abs/2505.02279.

Herman Errico. Autonomous action runtime management(aarm):a system specification for securing ai-driven actions at runtime, 2026. URL https://arxiv.org/abs/2602.09433.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026a.

Shicheng Fang, Yuxin Wang, Xiaoran Liu, Jiahao Lu, Chuanyuan Tan, Xinchi Chen, Yining Zheng, Xuanjing Huang, and Xipeng Qiu. Agentlongbench: A controllable long benchmark for long-contexts agents via environment rollouts, 2026b. URL https://arxiv.org/abs/2601.20730.

Yingchaojie Feng, Qiang Huang, Xiaoya Xie, Zhaorui Yang, Jun Yu, Wei Chen, and Anthony K. H. Tung. One interaction is worth a thousand guesses: Benchmarking the interactive capabilities of deep research agents, 2026. URL https://arxiv.org/abs/2601.06676.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking llm agents on dynamic and asynchronous environments, 2026. URL https://arxiv.org/abs/2602.11964.

Jiaxuan Gao, Jiaao Chen, Chuyi He, Shusheng Xu, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents, 2026. URL https://arxiv.org/abs/2601.22607.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey, 2024. URL https://arxiv.org/abs/2312.10997.

Ruocheng Guo, Kaiwen Dong, Xiang Gao, and Kamalika Das. Learning to rewrite tool descriptions for reliable llm-agent tool use, 2026a. URL https://arxiv.org/abs/2602.20426.

Xiyu Guo, Shan Wang, Chunfang Ji, Xuefeng Zhao, Wenhao Xi, Yaoyao Liu, Qinglan Li, Chao Deng, and Junlan Feng. Towards generalized routing: Model and agent orchestration for adaptive and efficient inference, 2025. URL https://arxiv.org/abs/2509.07571.

Zhengkang Guo, Yiyang Li, Lin Qiu, Xiaohua Wang, Jingwen Xv, Dongyu Ru, Xiaoyu Li, Xiaoqing Zheng, Xuezhi Cao, and Xunliang Cai. Agentescapebench: Evaluating out-of-domain tool-grounded reasoning in llm agents, 2026b. URL https://arxiv.org/abs/2605.07926.

Aayush Gupta. Reliabilitybench: Evaluating llm agent reliability under production-like stress conditions, 2026. URL https://arxiv.org/abs/2601.06112.

Rich Harang. Practical security guidance for sandboxing agentic workflows and managing execution risk, 2026. URL https://developer.nvidia.com/blog/practical-security-guidance-for-sandboxing-agentic-workflows-and-managing-execution-risk. NVIDIA Technical Blog, published January 30, 2026.

Chaoyue He, Xin Zhou, Di Wang, Hong Xu, Wei Liu, and Chunyan Miao. Harness engineering for language agents: The harness layer as control, agency, and runtime, 2026a. Preprints.

Zexue He, Yu Wang, Churan Zhi, Yuanzhe Hu, Tzu-Ping Chen, Lang Yin, Ze Chen, Tong Arthur Wu, Siru Ouyang, Zihan Wang, Jiaxin Pei, Julian McAuley, Yejin Choi, and Alex Pentland. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks, 2026b. URL https://arxiv.org/abs/2602.16313.

Zhongyu He, Yuanfan Li, Fei Huang, Tianyu Chen, Siyuan Chen, Xingyang Li, Meng Hsuan Yu, Xiangrong Liu, Leyi Wei, Lu Pan, Ke Zeng, and Xunliang Cai. Siri: Self-internalizing reinforcement learning with intrinsic skills for llm agent training, 2026c. URL https://arxiv.org/abs/2606.02355.

James Hollan, Edwin Hutchins, and David Kirsh. Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Trans. Comput.-Hum. Interact.*, 7(2):174196, June 2000. ISSN 1073-0516. doi: 10.1145/353485.353487. URL https://doi.org/10.1145/353485.353487.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions, 2026a. URL https://arxiv.org/abs/2507.05257.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026b. URL https://arxiv.org/abs/2512.13564.

Donghao Huang, Gauri Malwe, and Zhaoxia Wang. When agents fail to act: A diagnostic framework for tool invocation reliability in multi-agent llm systems, 2026a. URL https://arxiv.org/abs/2601.16280.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024a. URL https://arxiv.org/abs/2310.01798.

Peizhou Huang, Zixuan Zhong, Zhongwei Wan, Donghao Zhou, Samiul Alam, Xin Wang, Zexin Li, Zhihao Dou, Li Zhu, Jing Xiong, Chaofan Tao, Yan Xu, Dimitrios Dimitriadis, Tuo Zhang, and Mi Zhang. Mmdeepresearch-bench: A benchmark for multimodal deep research agents, 2026b. URL https://arxiv.org/abs/2601.12346.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey, 2024b. URL https://arxiv.org/abs/2402.02716.

Zhengjun Huang, Wenxuan Liu, Zhoujin Tian, Wei Chen, Junle Chen, Yuqian Wu, Fangyuan Zhang, Qintian Guo, and Xiaofang Zhou. M^3exam: Benchmarking multimodal memory for realistic user-agent interactions, 2026c. URL https://arxiv.org/abs/2606.07402.

Yeonjun In, Mehrab Tanjim, Jayakumar Subramanian, Sungchul Kim, Uttaran Bhattacharya, Wonjoong Kim, Sangwu Park, Somdeb Sarkhel, and Chanyoung Park. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation, 2026. URL https://arxiv.org/abs/2603.25001.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks, 2026. URL https://arxiv.org/abs/2604.24964.

Jenova AI. Long-context agentic orchestration benchmark, 2026. URL https://www.jenova.ai/en/resources/jenova-ai-long-context-agentic-orchestration-benchmark-february-2026. Jenova.ai benchmark page.

Dongming Jiang, Yi Li, Songtao Wei, Jinxin Yang, Ayushi Kishore, Alysa Zhao, Dingyi Kang, Xu Hu, Feng Chen, Qiannan Li, and Bingzhe Li. Anatomy of agentic memory: Taxonomy and empirical analysis of evaluation and system limitations, 2026a. URL https://arxiv.org/abs/2602.19320.

Tanqiu Jiang, Yuhui Wang, Jiacheng Liang, and Ting Wang. Agentlab: Benchmarking llm agents against long-horizon attacks, 2026b. URL https://arxiv.org/abs/2602.16901.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills – beyond tool use in llm agents, 2026c. URL https://arxiv.org/abs/2602.20867.

Cheng Jiayang, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations, 2026. URL https://arxiv.org/abs/2603.01966.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Maurits Kaptein, Vassilis-Javed Khan, and Andriy Podstavnychy. Runtime governance for ai agents: Policies on paths, 2026. URL https://arxiv.org/abs/2603.16586.

Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents, 2026. URL https://arxiv.org/abs/2605.09998.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023. URL https://arxiv.org/abs/2205.11916.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring ai ability to complete long software tasks, 2026. URL https://arxiv.org/abs/2503.14499.

Chingkwun Lam, Jiaxin Li, Lingfei Zhang, and Kuo Zhao. Governing evolving memory in llm agents: Risks, mechanisms, and the stability and safety governed memory (ssgm) framework, 2026. URL https://arxiv.org/abs/2603.11768.

LangChain. Langchain state of ai agents report: 2024 trends, 2024. URL https://www.langchain.com/state-of-agent-engineering. LangChain report.

LangChain. Context engineering in agents, 2025a. URL https://docs.langchain.com/oss/python/langchain/context-engineering. LangChain docs page.

LangChain. How agents can use filesystems for context engineering, 2025b. URL https://www.langchain.com/blog/how-agents-can-use-filesystems-for-context-engineering. LangChain blog.

LangChain. The anatomy of an agent harness, 2026. URL https://www.langchain.com/blog/the-anatomy-of-an-agent-harness. LangChain blog.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. Swe-evo: Benchmarking coding agents in long-horizon software evolution scenarios, 2026. URL https://arxiv.org/abs/2512.18470.

Lin-Fa Lee, Yi-Yu Chang, Chia-Mu Yu, and Kuo-Hui Yeh. Webmcp tool surface poisoning: Runtime manipulation attacks on llm agents, 2026a. URL https://arxiv.org/abs/2606.06387.

Yonghwa Lee, Wonseok Choi, and Donghyun Nam. Supply chain threats in the mcp ecosystem: Attack vectors and mitigation strategies. In *International Workshop on Security*, pp. 329–349. Springer, 2025.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses, 2026b. URL https://arxiv.org/abs/2603.28052.

Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, Yevgeniy Vorobeychik, and Chaowei Xiao. Agentdyn: Are your agent security defenses deployable in real-world dynamic environments?, 2026a. URL https://arxiv.org/abs/2602.03117.

Jinchao Li, Yunxin Li, Chenrui Zhao, Zhenran Xu, Baotian Hu, and Min Zhang. Windowsworld: A process-centric benchmark of autonomous gui agents in professional cross-application environments, 2026b. URL https://arxiv.org/abs/2604.27776.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution, 2026c. URL https://arxiv.org/abs/2510.25726.

Keyu Li, Junhao Shi, Yang Xiao, Mohan Jiang, Jie Sun, Yunze Wu, Dayuan Fu, Shijie Xia, Xiaojie Cai, Tianze Xu, et al. Agencybench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7422–7440, 2026d.

Ruizhe Li, Mingxuan Du, Benfeng Xu, Chiwei Zhu, Xiaorui Wang, and Zhendong Mao. Deepresearch bench ii: Diagnosing deep research agents via rubrics from expert report, 2026e. URL https://arxiv.org/abs/2601.08536.

Wanli Li, Bince Qu, Bo Pan, Jianyu Zhang, Zheng Liu, Pan Zhang, Wei Chen, and Bo Zhang. Literesearcher: A scalable agentic rl training framework for deep research agent, 2026f. URL https://arxiv.org/abs/2604.17931.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026g. URL https://arxiv.org/abs/2602.12670.

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. Benchmark test-time scaling of general llm agents, 2026h. URL https://arxiv.org/abs/2602.18998.

Yibo Li, Zijie Lin, Ailin Deng, Xuan Zhang, Yufei He, Shuo Ji, Tri Cao, and Bryan Hooi. Just-in-time reinforcement learning: Continual learning in llm agents without gradient updates, 2026i. URL https://arxiv.org/abs/2601.18510.

Yu Li, Haoyu Luo, Yuejin Xie, Yuqian Fu, Zhonghao Yang, Shuai Shao, Qihan Ren, Wanying Qu, Yanwei Fu, Yujiu Yang, Jing Shao, Xia Hu, and Dongrui Liu. Atbench: A diverse and realistic agent trajectory benchmark for safety evaluation and diagnosis, 2026j. URL https://arxiv.org/abs/2604.02022.

Zhuofeng Li, Haoxiang Zhang, Seungju Han, Sheng Liu, Jianwen Xie, Yu Zhang, Yejin Choi, James Zou, and Pan Lu. In-the-flow agentic system optimization for effective planning and tool use, 2025. URL https://arxiv.org/abs/2510.05592.

Huawei Lin, Peng Li, Jie Song, Fuxin Jiang, and Tieying Zhang. Muse-autoskill: Self-evolving agents via skill creation, memory, management, and evaluation, 2026a. URL https://arxiv.org/abs/2605.27366.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses, 2026b. URL https://arxiv.org/abs/2604.25850.

Minhua Lin, Juncheng Wu, Zijun Wang, Zhan Shi, Yisi Sang, Bing He, Zewen Liu, Tianxin Wei, Zongyu Wu, Zhiwei Zhang, Dakuo Wang, Xiang Zhang, Benoit Dumoulin, Cihang Xie, Yuyin Zhou, Suhang Wang, and Hanqing Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving llm agents, 2026c. URL https://arxiv.org/abs/2605.30621.

Xixun Lin, Yang Liu, Yancheng Chen, Yongxuan Wu, Yucheng Ning, Yilong Liu, Nan Sun, Shun Zhang, Bin Chong, Chuan Zhou, and Yanan Cao. Safeharness: Lifecycle-integrated security architecture for llm-based agent deployment, 2026d. URL https://arxiv.org/abs/2604.13630.

Yuchen Ling, Shengcheng Yu, Zhenyu Chen, and Chunrong Fang. Toward secure llm agents: Threat surfaces, attacks, defenses, and evaluation, 2026. URL https://arxiv.org/abs/2606.10749.

Enqi Liu, Liyuan Pan, Zhi Gao, Yan Yang, Chenrui Shi, Yang Liu, Jingrong Wu, and Qing Li. Benchmarking and improving gui agents in high-dynamic environments, 2026a. URL https://arxiv.org/abs/2604.25380.

Guangya Liu and Sujay Solomon. Ai agent observability - evolving standards and best practices, 2025. URL https://opentelemetry.io/blog/2025/ai-agent-observability/. OpenTelemetry blog.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, Zhengxi Lu, Gao Wu, Hao Wang, Liang Liu, and Yong Liu. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments, 2026b. URL https://arxiv.org/abs/2602.06075.

Guohong Liu, Jialei Ye, Jiacheng Liu, Yuanchun Li, Wei Liu, Pengzhi Gao, Jian Luan, and Yunxin Liu. Mobile gui agents under real-world threats: Are we there yet?, 2026c. URL https://arxiv.org/abs/2507.04227.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. Dive into claude code: The design space of today's and future ai agent systems, 2026d. URL https://arxiv.org/abs/2604.14228.

Lewis Liu and Nishant Yadav. Introducing state-bench: A benchmark for ai agent memory, 2026. URL https://opensource.microsoft.com/blog/2026/05/19/introducing-state-bench-a-benchmark-for-ai-agent-memory/. Microsoft Open Source blog.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), January 2023. ISSN 0360-0300. doi: 10.1145/3560815. URL https://doi.org/10.1145/3560815.

Shi Liu, Xuehai Tang, Xikang Yang, Liang Lin, Biyu Zhou, Wenjie Xiao, and Wantao Liu. When the manual lies: A realistic benchmark to evaluate mcp poisoning attacks for llm agents, 2026e. URL https://arxiv.org/abs/2605.24069.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

Xiang Long, Li Du, Yilong Xu, RongJian Xu, Qiyanhui Lu, Ying Gao, Qinhua Xie, Fangcheng Liu, Ning Ding, Haoqing Wang, Ziheng Li, Changjiang Zhou, Jianyuan Guo, and Yehui Tang. Liveclawbench: Benchmarking llm agents on complex, real-world assistant tasks, 2026. URL https://arxiv.org/abs/2604.13072.

Ryan Lopopolo. Harness engineering: leveraging codex in an agent-first world, 2026. URL https://openai.com/index/harness-engineering/. OpenAI blog.

Elias Lumer, Faheem Nizar, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. Tool-to-agent retrieval: Bridging tools and agents for scalable llm multi-agent systems, 2025. URL https://arxiv.org/abs/2511.01854.

Elias Lumer, Anmol Gulati, Faheem Nizar, Dzmitry Hedroits, Atharva Mehta, Henry Hwangbo, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A Burke. Tool and agent selection for large language model agents in production: A survey. In *2026 IEEE Conference on Artificial Intelligence (CAI)*, pp. 701–708. IEEE, 2026.

Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, Wenjie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, Zixuan Hu, Hongze Mi, Yibo Wang, Naiqiang Tan, Hong Chen, Yi R. Fung, Chun Yuan, and Li Shen. Ultrahorizon: Benchmarking agent capabilities in ultra long-horizon scenarios, 2025. URL https://arxiv.org/abs/2509.21766.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of llm agent memory mechanisms, 2026. URL https://arxiv.org/abs/2605.06716.

Jianan Ma, Xiaohu Du, Ruixiao Lin, Yaoxiang Bian, Jialuo Chen, Jingyi Wang, Xiaofang Yang, Shiwen Cui, Changhua Meng, Xinhao Deng, and Zhen Wang. Benchmarking autonomous agents against temporal, spatial, and semantic evasions, 2026a. URL https://arxiv.org/abs/2605.22321.

Ruiying Ma, Shreya Shankar, Ruiqi Chen, Yiming Lin, Sepanta Zeighami, Rajoshi Ghosh, Abhinav Gupta, Anushrut Gupta, Tanmai Gopal, and Aditya G. Parameswaran. Can ai agents answer your data questions? a benchmark for data agents, 2026b. URL https://arxiv.org/abs/2603.20576.

Zihan Ma, Zhikai Zhao, Chuanbo Hua, Federico Berto, and Jinkyoo Park. Judgeflow: Agentic workflow optimization via block judge, 2026c. URL https://arxiv.org/abs/2601.07477.

Manus. Context engineering for ai agents: Lessons from building manus, 2025. URL https://manus.im/blog/Context-Engineering-for-AI-Agents-Lessons-from-Building-Manus. Official Manus blog.

Rahul Marchand, Art O Cathain, Jerome Wynne, Philippos Maximos Giavridis, Sam Deverett, John Wilkinson, Jason Gwartz, and Harry Coppock. Quantifying frontier llm capabilities for container sandbox escape, 2026. URL https://arxiv.org/abs/2603.02277.

Alexandre De Masi. Terminal is all you need: Design properties for human-ai agent collaboration, 2026. URL https://arxiv.org/abs/2603.10664.

Lingrui Mei, Jiayu Yao, Yuyao Ge, Yiwei Wang, Baolong Bi, Yujun Cai, Jiazhi Liu, Mingyu Li, Zhong-Zhi Li, Duzhen Zhang, Chenlin Zhou, Jiayi Mao, Tianze Xia, Jiafeng Guo, and Shenghua Liu. A survey of context engineering for large language models, 2025. URL https://arxiv.org/abs/2507.13334.

Lennart Meincke, Ethan Mollick, Lilach Mollick, and Dan Shapiro. Prompting science report 2: The decreasing value of chain of thought in prompting, 2025. URL https://arxiv.org/abs/2506.07142.

Qianyu Meng, Yanan Wang, Liyi Chen, Yihang Li, Wei Wu, Wenyuan Jiang, Qimeng Wang, Chengqiang Lu, Yan Gao, Yi Wu, and Yao Hu. Agent harness for large language model agents: A survey. *Preprints*, April 2026. doi: 10.20944/preprints202604.0428.v3. URL https://doi.org/10.20944/preprints202604.0428.v3.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand,

Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Microsoft. Conductor, 2026. URL https://github.com/microsoft/conductor. Official Microsoft GitHub repository.

Microsoft Security. What is agentic AI security?, 2026. URL https://www.microsoft.com/en-us/security/business/security-101/what-is-agentic-ai-security. Microsoft Security documentation.

Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of llm agents: A survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 6129–6139, 2025.

Yasmin Moslem and John D. Kelleher. Dynamic model routing and cascading for efficient llm inference: A survey, 2026. URL https://arxiv.org/abs/2603.04445.

Franck Ndzomga. Efficient benchmarking of ai agents, 2026. URL https://arxiv.org/abs/2603.23749.

Liangbo Ning, Ziran Liang, Zhuohang Jiang, Haohao Qu, Yujuan Ding, Wenqi Fan, Xiao yong Wei, Shanru Lin, Hui Liu, Philip S. Yu, and Qing Li. A survey of webagents: Towards next-generation ai agents for web automation with large foundation models, 2025. URL https://arxiv.org/abs/2503.23350.

Xuying Ning, Katherine Tieu, Dongqi Fu, Tianxin Wei, Zihao Li, Yuanchen Bei, Jiaru Zou, Mengting Ai, Zhining Liu, Ting-Wei Li, Lingjie Chen, Yanjun Zhao, Ke Yang, Bingxuan Li, Cheng Qian, Gaotang Li, Xiao Lin, Zhichen Zeng, Ruizhong Qiu, Sirui Chen, Yifan Sun, Xiyuan Yang, Ruida Wang, Rui Pan, Chenyuan Yang, Dylan Zhang, Liri Fang, Zikun Cui, Yang Cao, Pan Chen, Dorothy Sun, Ren Chen, Mahesh Srinivasan, Nipun Mathur, Yinglong Xia, Hong Li, Hong Yan, Pan Lu, Lingming Zhang, Tong Zhang, Hanghang Tong, and Jingrui He. Code as agent harness, 2026. URL https://arxiv.org/abs/2605.18747.

Nous Research. Hermes agent, 2026. URL https://github.com/NousResearch/hermes-agent. Official Nous Research GitHub repository.

Mika Okamoto, Ansel Kaplan Erol, and Mark Riedl. Explainable model routing for agentic workflows, 2026. URL https://arxiv.org/abs/2604.03527.

OpenAI. Function calling and other api updates, 2023. URL https://openai.com/index/function-calling-and-other-api-updates/. OpenAI blog, June 13, 2023.

OpenAI. Introducing codex, 2025. URL https://openai.com/index/introducing-codex/. OpenAI blog.

OpenAI. Human-in-the-loop, 2026. URL https://openai.github.io/openai-agents-python/human_in_the_loop/. OpenAI Agents SDK documentation.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, Andre Saraiva, Andrea Vallone, Andrew Duberstein, Andrew Kondrich, Andrey Mishchenko, Andy Applebaum, Angela Jiang, Ashvin Nair, Barret Zoph, Behrooz Ghorbani, Bohan Zhang, Ben Rossen, Benjamin Sokolowsky, Boaz Barak, Bob McGrew, Borys Minaiev, Botao Hao, Bowen Baker, Brandon Houghton, Brandon McKinzie, Brydon Eastman, Camillo Lugaresi, Cary Bassin, Cary Hudson, Chak Ming Li, Charles

de Bourcy, Chelsea Voss, Chen Shen, Chong Zhang, Chris Koch, Chris Orsinger, Christopher Hesse, Claudia Fischer, Clive Chan, Dan Roberts, Daniel Kappler, Daniel Levy, Daniel Selsam, David Dohan, David Farhi, David Mely, David Robinson, Dimitris Tsipras, Doug Li, Dragos Oprica, Eben Freeman, Eddie Zhang, Edmund Wong, Elizabeth Proehl, Enoch Cheung, Eric Mitchell, Eric Wallace, Erik Ritter, Evan Mays, Fan Wang, Felipe Petroski Such, Filippo Raso, Florencia Leoni, Foivos Tsimpourlas, Francis Song, Fred von Lohmann, Freddie Sulit, Geoff Salmon, Giambattista Parascandolo, Gildas Chabot, Grace Zhao, Greg Brockman, Guillaume Leclerc, Hadi Salman, Haiming Bao, Hao Sheng, Hart Andrin, Hessam Bagherinezhad, Hongyu Ren, Hunter Lightman, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian Osband, Ignasi Clavera Gilaberte, Ilge Akkaya, Ilya Kostrikov, Ilya Sutskever, Irina Kofman, Jakub Pachocki, James Lennon, Jason Wei, Jean Harb, Jerry Twore, Jiacheng Feng, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joaquin Quiñonero Candela, Joe Palermo, Joel Parish, Johannes Heidecke, John Hallman, John Rizzo, Jonathan Gordon, Jonathan Uesato, Jonathan Ward, Joost Huizinga, Julie Wang, Kai Chen, Kai Xiao, Karan Singhal, Karina Nguyen, Karl Cobbe, Katy Shi, Kayla Wood, Kendra Rimbach, Keren Gu-Lemberg, Kevin Liu, Kevin Lu, Kevin Stone, Kevin Yu, Lama Ahmad, Lauren Yang, Leo Liu, Leon Maksin, Leyton Ho, Liam Fedus, Lilian Weng, Linden Li, Lindsay McCallum, Lindsey Held, Lorenz Kuhn, Lukas Kondraciuk, Lukasz Kaiser, Luke Metz, Madelaine Boyd, Maja Trebacz, Manas Joglekar, Mark Chen, Marko Tintor, Mason Meyer, Matt Jones, Matt Kaufer, Max Schwarzer, Meghan Shah, Mehmet Yatbaz, Melody Y. Guan, Mengyuan Xu, Mengyuan Yan, Mia Glaese, Mianna Chen, Michael Lampe, Michael Malek, Michele Wang, Michelle Fradin, Mike McClay, Mikhail Pavlov, Miles Wang, Mingxuan Wang, Mira Murati, Mo Bavarian, Mostafa Rohaninejad, Nat McAleese, Neil Chowdhury, Neil Chowdhury, Nick Ryder, Nikolas Tezak, Noam Brown, Ofir Nachum, Oleg Boiko, Oleg Murk, Olivia Watkins, Patrick Chao, Paul Ashbourne, Pavel Izmailov, Peter Zhokhov, Rachel Dias, Rahul Arora, Randall Lin, Rapha Gontijo Lopes, Raz Gaon, Reah Miyara, Reimar Leike, Renny Hwang, Rhythm Garg, Robin Brown, Roshan James, Rui Shu, Ryan Cheu, Ryan Greene, Saachi Jain, Sam Altman, Sam Toizer, Sam Toyer, Samuel Miserendino, Sandhini Agarwal, Santiago Hernandez, Sasha Baker, Scott McKinney, Scottie Yan, Shengjia Zhao, Shengli Hu, Shibani Santurkar, Shraman Ray Chaudhuri, Shuyuan Zhang, Siyuan Fu, Spencer Papay, Steph Lin, Suchir Balaji, Suvansh Sanjeev, Szymon Sidor, Tal Broda, Aidan Clark, Tao Wang, Taylor Gordon, Ted Sanders, Tejal Patwardhan, Thibault Sottiaux, Thomas Degry, Thomas Dimson, Tianhao Zheng, Timur Garipov, Tom Stasi, Trapit Bansal, Trevor Creech, Troy Peterson, Tyna Eloundou, Valerie Qi, Vineet Kosaraju, Vinnie Monaco, Vitchyr Pong, Vlad Fomenko, Weiyi Zheng, Wenda Zhou, Wenting Zhan, Wes McCabe, Wojciech Zaremba, Yann Dubois, Yinghai Lu, Yining Chen, Young Cha, Yu Bai, Yuchen He, Yuchen Zhang, Yunyun Wang, Zheng Shao, and Zhuohan Li. Openai o1 system card, 2026. URL https://arxiv.org/abs/2412.16720.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

OWASP Gen AI Security Project. Llm06:2025 excessive agency, 2025. URL https://genai.owasp.org/llmrisk/llm062025-excessive-agency/. OWASP Gen AI Security Project.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL https://arxiv.org/abs/2310.08560.

Linyue Pan, Lexiao Zou, Shuo Guo, Jingchen Ni, and Hai-Tao Zheng. Natural-language agent harnesses, 2026a. URL https://arxiv.org/abs/2603.25723.

Wenbo Pan, Shujie Liu, Chin-Yew Lin, Jingying Zeng, Xianfeng Tang, Xiangyang Zhou, Yan Lu, and Xiaohua Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference, 2026b. URL https://arxiv.org/abs/2606.05922.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL https://arxiv.org/abs/2305.15334.

Hubert M. Pysklo, Artem Zhuravel, and Patrick D. Watson. Agent-diff: Benchmarking llm agents on enterprise api tasks via code execution with state-diff-based evaluation, 2026. URL https://arxiv.org/abs/2602.11224.

Xue Qin, Simin Luan, John See, Zeyd Boukhers, Cong Yang, and Zhijun Li. Harnessing embodied agents: Runtime governance for policy-constrained execution, 2026. URL https://arxiv.org/abs/2604.07833.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of ai agent reliability, 2026. URL https://arxiv.org/abs/2602.16666.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

Mofasshara Rafique and Laurent Bindschaedler. Clawvm: Harness-managed virtual memory for stateful tool-using llm agents, 2026. URL https://arxiv.org/abs/2604.10352.

Zhenzhen Ren, Xinpeng Zhang, Zhenxing Qian, Yan Gao, Yu Shi, Shuxin Zheng, and Jiyan He. Gtm: Simulating the world of tools for ai agents, 2025. URL https://arxiv.org/abs/2512.04535.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncearenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. The prompt report: A systematic survey of prompt engineering techniques, 2025. URL https://arxiv.org/abs/2406.06608.

Biswa Sengupta and Jinhua Wang. Harbor: Automated harness optimization, 2026. URL https://arxiv.org/abs/2604.20938.

Haebin Seong, Li Yin, Haoran Zhang, and Zhan Shi. The last harness you'll ever build, 2026. URL https://arxiv.org/abs/2604.21003.

Md Shamsujjoha, Qinghua Lu, Dehai Zhao, and Liming Zhu. Swiss cheese model for ai safety: A taxonomy and reference architecture for multi-layered guardrails of foundation model based agents. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*, pp. 37–48. IEEE, 2025.

Shuai Shao, Qihan Ren, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, Dongrui Liu, and Jing Shao. Your agent may misevolve: Emergent risks in self-evolving llm agents, 2026. URL https://arxiv.org/abs/2509.26354.

Reshabh K Sharma. Contextcov: Deriving and enforcing executable constraints from agent instruction files, 2026. URL https://arxiv.org/abs/2603.00822.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback, 2022. URL https://arxiv.org/abs/2009.01325.

Hang Su, Jun Luo, Chang Liu, Xiao Yang, Yichi Zhang, Yinpeng Dong, and Jun Zhu. A survey on autonomy-induced security risks in large model-based agents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents, 2024. URL https://arxiv.org/abs/2309.02427.

Hui-Ze Tan, Xiao-Wen Yang, Hao Chen, Jie-Jing Shao, Yi Wen, Yuteng Shen, Weihong Luo, Xiku Du, Lan-Zhe Guo, and Yu-Feng Li. Hindsight credit assignment for long-horizon llm agents, 2026. URL https://arxiv.org/abs/2603.08754.

Zhenheng Tang, Xin He, Tiancheng Zhao, Fanjunduo Wei, Xiang Liu, Peijie Dong, Qian Wang, Qi Li, Huacan Wang, Ronghao Chen, et al. Llm agent memory: A survey from a unified representation–management perspective, 2026. Preprints.

Mohammad Tavakoli, Alireza Salemi, Carrie Ye, Mohamed Abdalla, Hamed Zamani, and J Ross Mitchell. Beyond a million tokens: Benchmarking and enhancing long-term memory in llms, 2026. URL https://arxiv.org/abs/2510.27246.

Xiaoyu Tian, Haotian Wang, Shuaiting Chen, Hao Zhou, Kaichi Yu, Yudian Zhang, Jade Ouyang, Junxi Yin, Jiong Chen, Baoyan Guo, Lei Zhang, Junjie Tao, Yuansheng Song, Ming Cui, and Chengwei Liu. Astra: Automated synthesis of agentic trajectories and reinforcement arenas, 2026. URL https://arxiv.org/abs/2601.21558.

Vivek Trivedy. Improving deep agents with harness engineering, 2026. URL https://www.langchain.com/blog/improving-deep-agents-with-harness-engineering. LangChain blog.

Janneke Van de Pol, Monique Volman, and Jos Beishuizen. Scaffolding in teacher–student interaction: A decade of research. *Educational psychology review*, 22(3):271–296, 2010.

Vera V. Vishnyakova. Context engineering: From prompts to corporate multi-agent architecture, 2026. URL https://arxiv.org/abs/2603.09619.

Lev S Vygotsky. *Mind in society: The development of higher psychological processes*, volume 86. Harvard university press, 1978.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents, 2025a. URL https://arxiv.org/abs/2503.18666.

Jize Wang, Xuanxuan Liu, Yining Li, Songyang Zhang, Yijun Wang, Zifei Shan, Xinyi Le, Cailian Chen, Xinping Guan, and Dacheng Tao. Gta-2: Benchmarking general tool agents from atomic tool-use to open-ended workflows, 2026a. URL https://arxiv.org/abs/2604.15715.

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

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. Gui agents with foundation models: A comprehensive survey, 2025b. URL https://arxiv.org/abs/2411.04890.

Wenhao Wang, Peizhi Niu, Gongyi Zou, Xiyuan Yang, Jingxing Wang, Haoting Shi, Yaxin Du, Jingyi Chai, Xianghe Pang, Shuo Tang, Yanfeng Wang, and Siheng Chen. Mcp-persona: Benchmarking llm agents on real-world personal applications via environment simulation, 2026b. URL https://arxiv.org/abs/2606.02470.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break, 2026c. URL https://arxiv.org/abs/2604.11978.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL https://arxiv.org/abs/2203.11171.

Yiqi Wang, Jiaqi Zhang, Taotao Cai, Zirui Liu, Qingqiang Sun, Zequn Sun, Zhangkai Wu, Manqing Dong, Mingkai Zheng, Xuefei Yin, and Yanming Zhu. From agent traces to trust: A survey of evidence tracing and execution provenance in llm agents, 2026d. URL https://arxiv.org/abs/2606.04990.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026e. URL https://arxiv.org/abs/2602.10090.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers, 2025c. URL https://arxiv.org/abs/2508.20453.

Zhexuan Wang, Xuebo Liu, Li Wang, Zifei Shan, Yutong Wang, Zhenxi Song, and Min Zhang. Maspo: Joint prompt optimization for llm-based multi-agent systems, 2026f. URL https://arxiv.org/abs/2605.06623.

Zihao Wang, Yiming Li, Yutong Wu, Zheyu Liu, Kangjie Chen, Fok Kar Wai, Pin-Yu Chen, Vrizlynn L. L. Thing, Bo Li, Dacheng Tao, and Tianwei Zhang. Who pays the price? stakeholder-centric prompt injection benchmarking for real-world web agents, 2026g. URL https://arxiv.org/abs/2606.13385.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. Plangenllms: A modern survey of llm planning capabilities. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19497–19521, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, Zac Kenton, Sasha Brown, Will Hawkins, Tom Stepleton, Courtney Biles, Abeba Birhane, Julia Haas, Laura Rimell, Lisa Anne Hendricks, William Isaac, Sean Legassick, Geoffrey Irving, and Iason Gabriel. Ethical and social risks of harm from language models, 2021. URL https://arxiv.org/abs/2112.04359.

David Wood, Jerome S Bruner, and Gail Ross. The role of tutoring in problem solving. *Journal of child psychology and psychiatry*, 17(2):89–100, 1976.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation, 2023. URL https://arxiv.org/abs/2308.08155.

Qinyuan Wu, Soumi Das, Mahsa Amani, Arijit Nag, Seungeon Lee, Krishna P. Gummadi, Abhilasha Ravichander, and Muhammad Bilal Zafar. To call or not to call: A framework to assess and optimize llm tool calling, 2026a. URL https://arxiv.org/abs/2605.00737.

Tongzhou Wu, Yuhao Wang, Xinyu Ma, Xiuqiang He, Shuaiqiang Wang, Dawei Yin, and Xiangyu Zhao. Deepresearch-9k: A challenging benchmark dataset of deep-research agent, 2026b. URL https://arxiv.org/abs/2603.01152.

Xixi Wu, Qianguo Sun, Ruiyang Zhang, Chao Song, Junlong Wu, Yiyan Qi, and Hong Cheng. Demystifying reinforcement learning for long-horizon tool-using agents: A comprehensive recipe, 2026c. URL https://arxiv.org/abs/2603.21972.

Yong Wu, YanZhao Zheng, TianZe Xu, ZhenTao Zhang, YuanQiang Yu, JiHuai Zhu, Chao Ma, BinBin Lin, BaoHua Dong, HangCheng Zhu, RuoHui Huang, and Gang Yu. Contextbudget: Budget-aware context management for long-horizon search agents, 2026d. URL https://arxiv.org/abs/2604.01664.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023. URL https://arxiv.org/abs/2309.07864.

Zhiheng Xi, Jixuan Huang, Chenyang Liao, Baodai Huang, Honglin Guo, Jiaqi Liu, Rui Zheng, Junjie Ye, Jiazheng Zhang, Wenxiang Chen, Wei He, Yiwen Ding, Guanyu Li, Zehui Chen, Zhengyin Du, Xuesong Yao, Yufei Xu, Jiecao Chen, Tao Gui, Zuxuan Wu, Qi Zhang, Xuanjing Huang, and Yu-Gang Jiang. Agentgym-rl: Training llm agents for long-horizon decision making through multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2509.08755.

Yizhe Xie, Congcong Zhu, Xinyue Zhang, Tianqing Zhu, Dayong Ye, Minfeng Qi, Huajie Chen, and Wanlei Zhou. From spark to fire: Modeling and mitigating error cascades in llm-based multi-agent collaboration, 2026. URL https://arxiv.org/abs/2603.04474.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2025a. URL https://arxiv.org/abs/2412.14161.

Kewei Xu, Xiaoben Lu, Shuofei Qiao, Zihan Ding, Haoming Xu, Lei Liang, and Ningyu Zhang. Longds-bench: On the failure of long-horizon agentic data analysis, 2026a. URL https://arxiv.org/abs/2605.30434.

Renjun Xu and Jingwen Peng. A comprehensive survey of deep research: Systems, methodologies, and applications, 2025. URL https://arxiv.org/abs/2506.12594.

Tianshi Xu, Huifeng Wen, and Meng Li. Adapting the interface, not the model: Runtime harness adaptation for deterministic llm agents, 2026b. URL https://arxiv.org/abs/2605.22166.

Wanghan Xu, Shuo Li, Tianlin Ye, Qinglong Cao, Yixin Chen, Hengjian Gao, Yiheng Wang, Qi Li, Kun Li, Sheng Xu, Shengdu Chai, Fangchen Yu, Xiangyu Zhao, Zhangrui Zhao, Weijie Ma, Zijie Guo, Koutian Wu, Haoyu Zhou, Haoxiang Yin, Lixue Cheng, Chaofan Hu, Haoxuan Li, Lu Mi, Xuxuan Xie, Yifan Zhou, Ruizhe Chen, Zhiwang Zhou, Xingjian Guo, Yuhao Zhou, Xuming He, Shengyuan Xu, Xinyu Gu, Jiamin Wu, Mianxin Liu, Chunfeng Song, Fenghua Ling, Dongzhan Zhou, Shixiang Tang, Yuqiang Li, Mao Su, Peng Ye, Siqi Sun, Bin Wang, Xue Yang, Zhenfei Yin, Tianfan Fu, Guangtao Zhai, Wanli Ouyang, Bo Zhang, Lei Bai, and Wenlong Zhang. Researchclawbench: A benchmark for end-to-end autonomous scientific research, 2026c. URL https://arxiv.org/abs/2606.07591.

Xinbo Xu, Ruihan Yang, Haiyang Shen, Wendong Xu, Bofei Gao, Ruoyu Wu, Kean Shi, Weichu Xie, Xuanzhong Chen, Ming Wu, Jason Zeng, Michael Heinrich, Elvis Zhang, Liang Chen, Kuan Li, and Baobao Chang. Roadmapbench: Evaluating long-horizon agentic software development across version upgrades, 2026d. URL https://arxiv.org/abs/2605.15846.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. Hallucination is inevitable: An innate limitation of large language models, 2025b. URL https://arxiv.org/abs/2401.11817.

Faouzi El Yagoubi, Godwin Badu-Marfo, and Ranwa Al Mallah. Agentleak: A benchmark for internal-channel privacy leakage in multi-agent llm systems, 2026. URL https://arxiv.org/abs/2602.11510.

Chang Yang, Chuang Zhou, Yilin Xiao, Su Dong, Luyao Zhuang, Yujing Zhang, Zhu Wang, Zijin Hong, Zheng Yuan, Zhishang Xiang, Shengyuan Chen, Huachi Zhou, Qinggang Zhang, Ninghao Liu, Jinsong Su, Xinrun Wang, Yi Chang, and Xiao Huang. Graph-based agent memory: Taxonomy, techniques, and applications, 2026. URL https://arxiv.org/abs/2602.05665.

Ya-Ting Yang and Quanyan Zhu. Toward reliable design of llm-enabled agentic workflows: Optimizing latency-reliability-cost tradeoffs, 2026. URL https://arxiv.org/abs/2605.23929.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023b. URL https://arxiv.org/abs/2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Peijie Yu, Wei Liu, Yifan Yang, Jinjian Li, Zelong Zhang, Xiao Feng, and Feng Zhang. Benchmarking llm tool-use in the wild, 2026. URL https://arxiv.org/abs/2604.06185.

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for llm agents, 2026. URL https://arxiv.org/abs/2603.22386.

Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. The shift from models to compound ai systems, 2024. URL https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/. BAIR blog.

Weihao Zeng, Yuzhen Huang, and Junxian He. Loca-bench: Benchmarking language agents under controllable and extreme context growth, 2026. URL https://arxiv.org/abs/2602.07962.

Chenchen Zhang. From reasoning to agentic: Credit assignment in reinforcement learning for large language models, 2026. URL https://arxiv.org/abs/2604.09459.

Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. Mcp security bench (msb): Benchmarking attacks against model context protocol in llm agents, 2026a. URL https://arxiv.org/abs/2510.15994.

Huanyao Zhang, Jiepeng Zhou, Bo Li, Bowen Zhou, Yanzhe Shan, Haishan Lu, Zhiyong Cao, Jiaoyang Chen, Yuqian Han, Zinan Sheng, Zhengwei Tao, Hao Liang, Jialong Wu, Yang Shi, Yuanpeng He, Jiaye Lin, Qintong Zhang, Guochen Yan, Runhao Zhao, Zhengpin Li, Xiaohan Yu, Lang Mei, Chong Chen, Wentao Zhang, and Bin Cui. Browsecomp-v^3: A visual, vertical, and verifiable benchmark for multimodal browsing agents, 2026b. URL https://arxiv.org/abs/2602.12876.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation, 2025a. URL https://arxiv.org/abs/2410.10762.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026c. URL https://arxiv.org/abs/2510.04618.

Yabo Zhang, Yihan Zeng, Qingyun Li, Zhen Hu, Kavin Han, and Wangmeng Zuo. Tool-r1: Sample-efficient reinforcement learning for agentic tool use, 2025b. URL https://arxiv.org/abs/2509.12867.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, Chenxu Lv, and Junyang Lin. Deepplanning: Benchmarking long-horizon agentic planning with verifiable constraints, 2026d. URL https://arxiv.org/abs/2601.18137.

Yudi Zhang, Meng Fang, Zhenfang Chen, and Mykola Pechenizkiy. Self-evolving llm agents with in-distribution optimization, 2026e. URL https://arxiv.org/abs/2606.07367.

Zheyuan Zhang, Lin Ge, Hongjiang Li, Weicheng Zhu, Chuxu Zhang, and Yanfang Ye. Mapro: Recasting multi-agent prompt optimization as maximum a posteriori inference, 2025c. URL https://arxiv.org/abs/2510.07475.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. Ama-bench: Evaluating long-horizon memory for agentic applications, 2026. URL https://arxiv.org/abs/2602.22769.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vuli, Anna Korhonen, and Sercan Ö. Ark. Multi-agent design: Optimizing agents with better prompts and topologies, 2026a. URL https://arxiv.org/abs/2502.02533.

Qixing Zhou, Jiacheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. Featurebench: Benchmarking agentic coding for complex feature development, 2026b. URL https://arxiv.org/abs/2602.10975.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

Jie Zhu, Yimin Tian, Boyang Li, Kehao Wu, Zhongzhi Liang, Junhui Li, Xianyin Zhang, Lifan Guo, Feng Chen, Yong Liu, and Chi Zhang. Finmcp-bench: Benchmarking llm agents for real-world financial tool use under the model context protocol, 2026a. URL https://arxiv.org/abs/2603.24943.

Shiping Zhu, Yibo Yang, Zhengyang Wang, Tiancheng Shen, Dandan Guo, and Ming-Hsuan Yang. H2hmem: A multimodal memory benchmark for agents in human-human interactions, 2026b. URL https://arxiv.org/abs/2606.09461.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences, 2020. URL https://arxiv.org/abs/1909.08593.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, and Chao Yang. Mcp-safetybench: A benchmark for safety evaluation of large language models with real-world mcp servers, 2026. URL https://arxiv.org/abs/2512.15163.