The Model–Harness Configuration as the Unit of Agentic Capability

Bert Colemont*

June 9, 2026

Abstract

Progress in autonomous LLM agents is conventionally attributed to the model: "model M scores $X\%$ on benchmark Y." Yet the same model embedded in different runtimes (different context management, tool surfaces, orchestration, and verification) succeeds or fails on the same task, so model-level attribution obscures a large and controllable source of performance variance. This paper argues that the appropriate unit of analysis for agentic capability is not the bare model but the *model–harness configuration*: the model together with the runtime that governs what it perceives, what it may do, what state survives across turns, what it may not do, how its work is evaluated, and how it recovers from failure. We make three contributions. First, we define the harness and the model–harness configuration, distinguishing *capability-of-the-model* (per-step competence of the weights) from *capability-of-the-configuration* (end-to-end task success under a given runtime). Second, we give an operating-systems-grounded taxonomy of harness components (perception, action, state, permissions, orchestration, observability, and recovery), each with its OS analog and the design question it poses. Third, we synthesize empirical evidence, spanning reasoning orchestration, tool retrieval, context compression, and memory, that scaffolding shifts end-to-end task success substantially with the model held fixed. We draw out implications for benchmarking (configuration-level reporting), for open-weight competitiveness (a "build to delete" account in which weaker models need more scaffolding and stronger models need less), and for governance, where a runtime-enforced, logged control is auditable evidence while a prompt instruction is only behavior.

1 Introduction

Statements of the form "model M scores $X\%$ on benchmark Y" are the default currency of progress reports in autonomous agents. They are convenient, comparable across leaderboards, and almost always misleading for the systems people actually deploy. A language model does not, by itself, resolve a GitHub issue, navigate a website, or hold a multi-turn customer-service conversation under a policy. To do those things it must be embedded in a runtime that decides what it perceives, what it may invoke, what survives between turns, what it is forbidden to do, and how its output is checked. That runtime is the subject of this paper. We call it the *harness*, and we argue that the appropriate unit of analysis for agentic capability is not the model but the *model–harness configuration*.

The gap between how capability is reported and how it is produced is now well documented. On real-world software engineering, state-of-the-art models resolve only a small fraction of issues

*Euraika Labs. Correspondence: bert@euraika.net. A non-technical companion essay appears at https://www.euraika-labs.net/engineering/2026-06-09-the-model-is-the-cpu.

that require multi-file edits, tool interaction, and long-context reasoning, and the benchmarks built to measure this explicitly require an execution environment rather than a bare completion endpoint [5, 10]. On realistic interactive web tasks, agents built on a strong backbone reach a small fraction of human performance [46], and on general assistant tasks the gap between an equipped model and a human is similarly large [19]. None of these gaps is closed by the model alone; closing them is an engineering problem in the surrounding runtime.

Crucially, that runtime is not incidental glue. It is a large and controllable source of variance. Holding the backbone model fixed, the scaffolding around it (context management, note-taking, tool design) measurably changes the success rate: enabling hierarchical memory and advanced context management raises one coding agent's resolve rate from 42.0% to 48.6% on a SWE-Bench Pro subset, and the full scaffold outperforms a simpler baseline (45.5% versus 42.7%) on the same model [38]. Systematic sweeps over model–harness pairings find substantial variation in completion, process quality, efficiency, and failure behavior, with plausible reasoning becoming decoupled from tool feedback and verifiable output contracts [26, 43]. Meta-analyses of agent benchmarks go further: flaws in task setup and reward design (which are properties of the harness, not the model) can cause up to 100% relative over- or under-estimation of measured capability [47]. A number attributed to a model is, in practice, a number produced by a configuration that usually goes unreported.

We use the term *harness* for the runtime that governs an agent's interaction with the world along six axes: (1) what the model perceives (context and retrieval); (2) what it can do (tools and actions); (3) how state survives turns (files, commits, memory); (4) what it may not do (permissions, sandboxing, approvals); (5) how its work is evaluated (tests, traces, evaluators); and (6) how it recovers from failure. The *model–harness configuration* is the pairing of a specific model with a specific instantiation of these axes, and it is this pairing (not the model name) to which a capability claim properly attaches. This view is consistent with a growing line of work that treats harness components as analyzable scientific objects rather than ambient infrastructure [21, 33, 45].

This paper is a position-and-survey contribution scoped to long-horizon, autonomous agents: the regime in which a model must take many dependent steps against a stateful environment, where harness effects compound and where reporting at the bare-model level is most distorting. We introduce no new experiments; instead we consolidate published evidence and organize it. Our contributions are:

- **A definition.** We give a precise definition of an agent harness and of the model–harness configuration as the unit of analysis for reasoning about agentic capability, distinguishing capability-of-the-configuration from capability-of-the-model (Section 3.1).
- **A taxonomy.** We present an operating-systems-grounded taxonomy of harness components (perception/context, action/tools, state/memory, permissions/sandboxing, orchestration, observability/evaluation, and recovery), each with its OS analog, a one-sentence definition, and the design question it poses (Section 3.2). We treat the OS analogy as an organizing device, not a validated isomorphism.
- **A synthesis.** We synthesize empirical evidence that scaffolding materially shifts agent task success with the model held fixed, spanning reasoning orchestration, tool retrieval, context compression, memory, and configuration-level benchmarking (Section 4).
- **Implications.** We draw out consequences for practice: configuration-level benchmark reporting; a "build to delete" view in which each harness component encodes an assumption about a current model weakness, so weaker and open-weight models need more scaffolding while

stronger models need less; and the harness as the governance boundary, where a runtime-enforced, logged control is auditable evidence while a prompt instruction is only behavior: a distinction with direct relevance to regulated and EU contexts [6] (Section 5).

2 Related Work

We organize prior work into seven themes. Each names a body of evidence that, read together, treats the runtime around the model as an engineerable object whose configuration shifts measured capability. We then position our framing (an operating-systems-grounded taxonomy with the model–harness configuration as the unit of analysis) against existing architectural decompositions.

2.1 Reasoning and Orchestration as Harness, Not Model

A line of work shows that *how* inference is orchestrated around a fixed model produces large capability gains without any weight change. Chain-of-thought prompting elicits intermediate reasoning that improves complex tasks [37]; self-consistency samples and votes over reasoning paths for substantial arithmetic and commonsense gains [36]; Tree of Thoughts adds explicit search, backtracking, and pruning over intermediate states [41]; ReAct interleaves reasoning with grounded actions for interactive tasks [40]; and Self-Refine and Reflexion add test-time feedback and episodic-memory reflection loops [16, 31]. We read these collectively as evidence that the orchestration loop is a first-class harness component: the same processing core clocked by a different scheduler does materially different work.

2.2 Tools, Retrieval, and the Case for a Smaller Action Space

A substantial tool-use literature finds that invocation reliability depends on the runtime that selects, describes, and retrieves tools rather than on the model alone. Surveys frame reasoning and tool augmentation as distinct, composable capabilities [18] and organize tool use into prompting, supervised, and reward-driven paradigms [7]; Toolformer learns when to call APIs [29]; Gorilla shows a retriever-augmented open model surpassing a stronger model on API calls, so reliability is a property of the full retrieval-plus-documentation stack [23]; ToolLLM and ToolSandbox extend this to thousands of real and stateful APIs [15, 24]; and dynamic, history-conditioned retrieval improves function calling over large static surfaces [22]. This grounds our claim that more harness is not always better: the syscall interface should be pruned to the task, and tool selection is itself a governed runtime decision.

2.3 Context, Memory, and State Across Turns

Work on context compression, retrieval matching, skill libraries, and agent memory treats what the model perceives and what survives between turns as engineerable substrate. Failure-driven context compression reduces tokens and improves long-horizon success, including for smaller models [11], and production terminal agents converge on the same lever through adaptive compaction of older observations [3]; retrieval method must be matched to task characteristics rather than applied universally [44]; Voyager persists executable skills in an ever-growing library [34]; and surveys of agent memory systematize episodic and semantic stores [8]. The synthesis is that long-horizon capability is increasingly produced by reorganizing externalized state rather than by changing weights [45].

2.4 Coding-Agent Ablations: Scaffolding Moves the Score

Real-world software-engineering benchmarks provide the cleanest model-held-fixed ablations [5, 10]. With the backbone fixed, scaffolding measurably changes SWE-Bench Pro success across harness configurations [38]; harnesses can be versioned, optimized, and ablated as artifacts [21, 33]; and domain-specialized open-weight models punch above their size [27]. Together these make scaffolding a measurable, attributable variable rather than incidental glue.

2.5 Configuration-Level Evaluation and Benchmark Rigor

Agent benchmarks and meta-analyses converge on a finding that the evaluation harness shapes, and can badly distort, measured capability. Realistic stateful environments expose gaps that bare-model scores miss [14, 19, 42, 46]; harness and reward-design flaws cause large over- or under-estimates, motivating harness disclosure as part of reporting [47]; consistency metrics reveal what single runs hide [42]; and capability varies systematically across model–harness pairings and along a hierarchy of agentic skills [26, 43]. We use this to argue that capability must be reported at the configuration level.

2.6 Permissions, Observability, and Governance as Runtime Properties

A governance and security literature locates control in the runtime: mandatory and context-aware access control enforce least privilege that prompts cannot [9, 13]; runtime telemetry frameworks instrument agents with structured logs of reasoning, state changes, and tool interactions, making inspection a property of execution rather than reconstruction [1]; and integrated governance frameworks operationalize risk into design-, runtime-, and audit-stage controls, escalating high-impact actions to human oversight [12, 25]. Read alongside oversight studies [2, 30] and the EU AI Act's human-oversight requirements [6, 17], this supports our governance claim: a runtime-enforced, logged control is auditable evidence whereas a prompt instruction is only behavior.

2.7 Architectural Decompositions and How We Differ

Surveys and system-theoretic frameworks already decompose agents into components, patterns, and communication protocols [4, 28, 32, 35, 39], and formalize agentic RAG as a controlled decision process [20]. We build on this decomposition in two ways. First, we reframe it around an operating-systems analogy that names each component's OS analog and the design question it poses. Second, and more centrally, we elevate the resulting model–harness configuration to the unit of analysis for reporting capability, rather than treating it as background structure beneath a model name.

3 The Harness: Definition and Taxonomy

3.1 Definitions

We define a *harness* as the runtime that surrounds a language model and governs its operation as an agent: the software that decides what the model perceives at each step, what actions it may invoke, what state survives across turns, what it is forbidden to do, how its reasoning and acting are sequenced, how its work is observed and evaluated, and how failures are detected and recovered. The harness is everything between the bare next-token predictor and the task: it is not part of the weights, and it is not the task environment, but the engineered layer that mediates between them. Surveys of agent architectures already enumerate these layers under various names (policy

Harness component	OS analog	Design question
Perception / Context	RAM, working set	What to retrieve, compact, or evict?
Action / Tools	Syscalls, devices	Which tools to expose at each step?
State / Memory	Disk, filesystem	What must persist across turns?
Permissions / Sandboxing	Kernel rings	What least privilege, enforced how?
Orchestration Loop	Scheduler	How to sequence reason and act?
Observability / Evaluation	Logging, tracing	How to make work inspectable?
Recovery	Exceptions, rollback	How to detect failure and retry?
Governance Boundary	Trusted kernel	Which controls are runtime-enforced?

Table 1: The harness components, their operating-systems analogs, and the design question each poses. The mapping is an organizing analogy, not a validated isomorphism.

core, memory, planners, tool routers, critics) and treat them as recurring, designable components rather than incidental code [4, 35, 39].

We define the *model–harness configuration* as the pair of a fixed model and a fully specified harness, and we take this pair, rather than the model name alone, to be the proper unit of analysis when reporting or reasoning about agentic capability. This distinction matters because the two quantities it separates are empirically distinct. *Capability-of-the-model* is a property of the weights: the per-step competence a model exhibits on a single forward pass. *Capability-of-the-configuration* is the end-to-end task success of the model under a particular harness, and it can differ sharply for the same model under different harnesses (and for the same harness under different models) across completion rate, process quality, efficiency, and failure behavior [38, 43]. Treating the harness as a scientific representation object, rather than as glue, is what makes configuration-level reporting tractable: harness policies can be written, ablated, and compared as artifacts [21].

3.2 An operating-systems analogy

To organize the components of a harness we adopt an operating-systems analogy. A bare language model resembles a processor: capable of useful work per cycle, but unable on its own to manage memory, persist state, reach devices, enforce protection, schedule work, or recover from faults. An operating system supplies exactly these services, and a harness supplies their agentic counterparts. The mapping is summarized in Table 1.

We stress that this is an *analogy*, not a claimed isomorphism. The correspondences are heuristic and the boundaries between components are softer than in a real kernel: context management and memory shade into one another, orchestration and recovery overlap, and the governance boundary is a cross-cutting property rather than a discrete subsystem. We use the analogy because it names the design questions crisply and because each harness component answers, for the agent, a problem the operating system solves for the program. It is an organizing device, and the claims that depend on it are positions rather than measured results.

3.3 The components

We name eight components. The first seven are subsystems; the eighth is a cross-cutting property of the runtime.

Perception / Context (RAM, working set). The layer that decides what subset of available information enters the model's finite context window at each step, through retrieval, compression, and reminders. *Given a finite context budget, what should be retrieved, compacted, or evicted so*

the model perceives the right state without distraction? [11, 44]

Action / Tools (syscalls, devices). The set of tools the model may invoke, together with how they are described, retrieved, and selected at runtime. *Which tools should be exposed at each step, and is a smaller, dynamically retrieved action space better than a large static one?* [22, 23, 29]

State / Memory (disk, filesystem). The mechanisms by which work products and learned context survive across turns and episodes: files, commits, skill libraries, and episodic or semantic stores. *What must persist across turns and episodes, and in what substrate, so that progress accumulates rather than being relitigated each step?* [8, 34]

Permissions / Sandboxing (kernel rings). The runtime constraints that bound what the agent may not do: sandboxes, access-control policies, approval gates, and privilege separation. *What is the least privilege sufficient for the task, and how is it enforced by the runtime rather than requested in a prompt?* [9, 13]

Orchestration Loop (scheduler). The control structure sequencing reasoning and acting: iteration, search, reflection, self-consistency, and single- versus multi-agent coordination. *How should reasoning and acting be sequenced, searched, or parallelized to convert per-step capability into reliable end-to-end completion?* [36, 40, 41]

Observability / Evaluation (logging, tracing). The layer that records structured traces of reasoning, state, and tool interactions, and evaluates work against tests, checkers, or judges. *How is the agent's work made inspectable and verifiable at runtime so that success is measured against a real contract rather than asserted?* [1, 47]

Recovery (exceptions, rollback). The mechanisms that detect failure and respond through retries, rollbacks to prior snapshots, or revised plans. *When a step fails or a trajectory goes off-contract, how does the runtime detect it and roll back or retry rather than compounding the error?* [16, 31]

Governance Boundary (the trusted kernel). The cross-cutting property by which permissions, authorization, telemetry, and audit trails are enforced and recorded by the runtime, making controls verifiable. *Which controls are enforced and logged by the runtime (and therefore auditable evidence) versus merely instructed in a prompt, and therefore only behavior?* [12] This boundary is the locus of our governance argument: a control that the runtime enforces is evidence, whereas an instruction the model is asked to follow is at best behavior.

3.4 The configuration, sketched

Figure 1 sketches the resulting picture: a fixed model, ringed by the harness subsystems, with the governance boundary drawn as the perimeter through which every action and observation passes.

The figure makes the section's central claim visual: the model is the processor at the center, but the system is the enclosing configuration. The remainder of the paper reads the empirical literature through these components and argues that capability should be attributed to the configuration that produced it.

4 Evidence that Scaffolding Shifts Capability

This section marshals the empirical literature for the central claim of the paper: that the harness, the runtime surrounding a fixed model, is a large and controllable source of variance in agent task success. We organize the evidence by harness component, following the taxonomy of Section 3.2, and we label evidence strength honestly. Most published results vary more than one factor at a time and were obtained on heterogeneous benchmarks with different backbones; clean single-

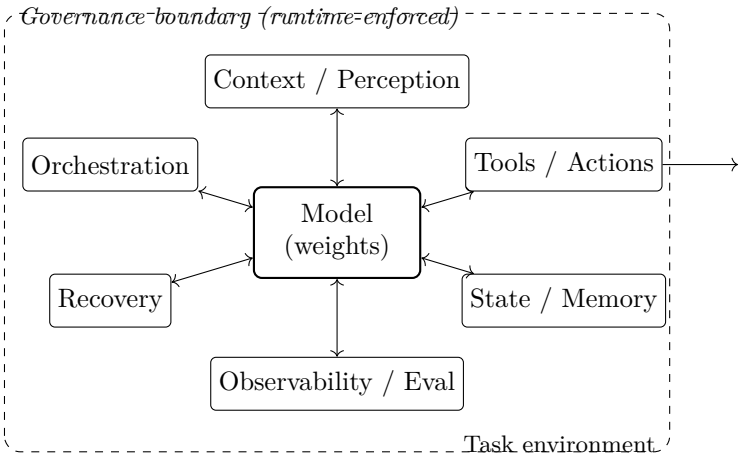

Figure 1: The model–harness configuration. A fixed model (center) is surrounded by harness subsystems that govern what it perceives, what it can do, what persists, how it is sequenced, observed, and recovered. The dashed perimeter is the governance boundary: every action and observation crosses a runtime that can enforce and log controls. The model is one component among several, not the whole system.

variable ablations with the model held strictly fixed are the exception rather than the rule. Where the evidence is direct we say so, and where it is suggestive or position-only we say that instead.

4.1 Orchestration: the loop, not the weights

The strongest model-held-fixed evidence comes from inference-time orchestration, where the weights are untouched and only the control structure around generation changes. Chain-of-thought prompting elicits intermediate reasoning that materially improves performance on arithmetic and symbolic tasks without any weight update [37]. Self-consistency, which samples multiple reasoning paths and marginalizes by voting, improves GSM8K by 17.9% over greedy chain-of-thought decoding on the same model [36]. Tree of Thoughts replaces linear decoding with explicit search, lookahead, and backtracking, and reports that GPT-4 reaches 74% on Game of 24 against 4% for chain-of-thought prompting [41]; this is among the cleanest demonstrations that the orchestration loop, not the model, can dominate measured capability. Interleaving reasoning with tool-grounded action via ReAct yields absolute success improvements of 34% on ALFWorld and 10% on WebShop over baselines [40]. Test-time feedback loops extend the pattern: Self-Refine reports roughly 20% absolute average improvement across seven tasks using a single model as generator, critic, and refiner [16], and Reflexion, which maintains verbal reflections in an episodic-memory buffer, reaches 91% pass@1 on HumanEval [31]. We read these results collectively as strong evidence that the orchestration loop is a first-class, capability-bearing harness component.

4.2 Coding-agent ablations with a fixed backbone

Software-engineering agents provide the closest approximation to controlled, model-held-fixed ablations of the surrounding scaffold. SWE-bench established that resolving real GitHub issues demands execution interaction and long-context handling beyond bare-model competence [10], and SWE-Bench Pro extends this to enterprise-scale, long-horizon tasks [5]. Against this backdrop, the Confucius Code Agent reports the most directly relevant ablation: holding the backbone fixed

at Claude 4 Sonnet, enabling hierarchical memory and advanced context management improves Resolve@1 from 42.0% to 48.6% on a SWE-Bench Pro subset, and the full scaffold (CCA) reaches 45.5% versus 42.7% for a SWE-Agent baseline on the same model [38]. This is strong, single-backbone evidence that scaffolding, not weights, accounts for the gap. Complementing it, work that treats the harness itself as a manipulable artifact strengthens the structural case: Natural-Language Agent Harnesses show that explicit harness modules can be systematically composed and ablated as scientific objects rather than incidental glue [21], and VeRO provides versioned snapshots and structured traces for optimizing one agent's harness via another [33]. That specialized open-weight models such as Code Llama can exceed larger general models on code [27] further indicates that deployed capability is a property of the configuration rather than of raw scale alone (moderate evidence, as the comparison varies the model rather than holding it fixed).

4.3 Tools: more harness is not always better

The tool-use literature supports both that the action surface is load-bearing and that pruning it can help. Toolformer shows a model can learn when and how to invoke APIs [29]; Gorilla shows that a retriever-augmented open model can surpass GPT-4 at writing API calls, locating reliability in the full retrieval-plus-documentation stack rather than the model [23]. Against large static tool availability, Dynamic Tool Dependency Retrieval conditions selection on the evolving tool-invocation history and reports function-calling success improvements between 23% and 104% over static retrievers [22]. This is strong support for the "smaller, dynamically retrieved action space" position, with the usual caveat that the comparison spans different retrieval methods rather than a single isolated variable.

4.4 Context and memory across turns

What the model perceives and what survives between turns is engineerable substrate. ACON optimizes failure-driven context compression in natural-language space, reducing peak token usage by 26–54% while improving task success, and reports that smaller models can function as long-horizon agents under it (up to 46% improvement by reducing context distraction) [11]. Voyager demonstrates that an ever-growing library of executable skills produces compounding capability that transfers to new worlds in an embodied setting [34], and a recent survey systematizes the memory substrates and cognitive mechanisms on which long-horizon agents depend [8]. These are strong for the compression result and structural for memory, given the heterogeneity of memory designs surveyed.

4.5 Configuration-level variation across pairings

Finally, work that varies model and harness jointly shows that capability is a property of the pairing. Harness-Bench evaluates representative harness configurations across multiple backends over 5,194 trajectories and finds substantial variation in completion, process quality, efficiency, and failure behavior, including execution-alignment failures where plausible reasoning decouples from tool feedback [43]. A complementary study derives a hierarchy of agentic capabilities in a realistic RL environment, where weaker models fail at tool use and planning while stronger ones fail at contextual inference [26]. Together these support configuration-level rather than model-level reporting (strong).

4.6 Honest caveats

The case is strong in aggregate but uneven in rigor. Benchmarks differ in task distribution and setup, and few studies isolate a single harness variable on a single frozen model; many co-vary scaffold and backbone. Reported numbers are themselves harness-sensitive: an audit of agentic benchmarks finds that task-setup and reward-design flaws can drive up to 100% relative over- or under-estimation of capability, and argues that the evaluation harness must be disclosed and open-sourced as part of any result [47]. The figures cited above should therefore be read as evidence of direction and magnitude under the reported configuration, not as model constants.

5 Implications: Benchmarking, Open-Weights, and Governance

If the model–harness configuration is the unit of agentic capability, then three consequences follow directly: for how capability is measured, for how open-weight models are judged competitive, and for where controls must be enforced. We take each in turn.

5.1 Benchmarking: report at the configuration level

The most immediate implication is that a benchmark number attached to a bare model name is under-specified. The evidence assembled above shows that the harness is a controllable source of variance: holding the backbone fixed, scaffolding choices move SWE-Bench Pro success rates by several points with the backbone held fixed [38], and across model–harness pairings the variation in completion, process quality, efficiency, and failure behavior is substantial enough that completion rate alone is a lossy summary [43]. A result of the form "model M scores x on benchmark B" is therefore an abbreviation for "model M under harness H scores x on B," with H silently fixed and undisclosed.

This is not merely an attribution nicety. Systematic audits of agentic benchmarks show that flaws in task setup and reward design can produce up to 100% relative over- or under-estimation of capability, and the proposed remedy explicitly includes transparency about the evaluation harness and open-sourcing of the benchmark and its scaffolding [47]. We read these findings together as a single prescription: capability should be reported at the configuration level, with the harness disclosed and, where feasible, released so that a number can be reproduced and a comparison can hold the harness constant. Configuration-level reporting also clarifies what is being compared (a harness improvement, a model improvement, or their interaction) which model-only leaderboards conflate.

5.2 Open-weights and "build to delete"

The second implication concerns the open-weight competitiveness picture. Because each harness component encodes an assumption about a current model weakness (retrieval compensates for limited parametric recall, context compression for finite working memory, structured tool retrieval for unreliable selection over large action spaces) the optimal quantity of scaffolding is not fixed but decreases as the underlying model improves. We summarize this as *build to delete*: scaffolding is provisional, written against a weakness that a stronger model may not have.

The empirical reading supports this directional claim. Open-weight models equipped with instruction tuning and search-based tool orchestration can approach proprietary performance on tool use [24], and domain-specialized open models can match or exceed larger general models within their domain [27], indicating that deployed capability is a property of the configuration rather than

of raw scale. Harness-level context management can let smaller models function as long-horizon agents that would otherwise fail through context distraction [11], and the broader trend is that recent capability gains increasingly come from reorganizing runtime infrastructure rather than from changing weights [45]. The practical consequence is asymmetric: weaker and open-weight models benefit more from added scaffolding and so should be evaluated with their intended harness, while stronger models need less and may even be impaired by scaffolding written for their predecessors. A fair open-versus-proprietary comparison is therefore a comparison of configurations, and a harness that is load-bearing today is a candidate for deletion tomorrow.

5.3 Governance: where controls must live

The third implication is that the harness is the governance boundary, and this relocates where a control counts as evidence. A constraint expressed only as a prompt instruction is behavior: it may be followed, but it is neither enforced nor logged by anything outside the model. A constraint enforced by the runtime is evidence: it is auditable independently of the model's output. This distinction is operationalized by work that places access control in the runtime (mandatory and attribute-based frameworks that monitor agent-tool interactions over information-flow graphs and block privilege escalation [9], and context-aware information-flow governance that supersedes binary permission models [13]) and by runtime telemetry that instruments agents with structured logging of reasoning, state changes, and tool interactions, so that oversight is grounded in execution traces rather than post-hoc reconstruction [1].

This framing matters most in regulated and EU contexts. The EU AI Act requires that high-risk systems be designed so that natural persons can effectively oversee them: monitoring operation to detect and address anomalies, deciding to disregard, override, or reverse the system's output, and interrupting the system through a stop function [6]. These are architectural requirements on the deployed configuration, not properties of model weights: the capacity to intervene, to stop, and to inspect must be realized by the runtime. Studies of public-sector oversight reinforce that continuous, integrated supervision rather than episodic approval is what such deployments require [30], and emerging work on computational compliance argues that conformance checks must run across the system lifecycle rather than being asserted at a single review point [17]. The throughline is consistent with the thesis: oversight, like capability, is a property of the model–harness configuration, and only controls the runtime enforces and records can serve as auditable evidence.

6 Discussion and Limitations

The argument of this paper is deliberately structural rather than experimental. We contribute a definition, an organizing taxonomy, and a synthesis of existing evidence; we do not run new ablations, and the framework should be judged on whether it makes the published record easier to read coherently, not on novel measurements of our own. Several limitations follow directly from this stance, and we state them plainly so that the framing is not read as a stronger empirical claim than it is.

6.1 The evidence is assembled, not controlled

The studies we synthesize were conducted independently, on different models, benchmarks, harnesses, and time periods, so their reported numbers cannot be placed on a common axis. When we observe that tree search reaches a far higher success rate than chain-of-thought prompting on

a puzzle task [37, 41], that self-consistency improves arithmetic reasoning [36], that reason-act interleaving improves interactive task success [40], or that scaffolding shifts software-engineering success with a backbone held fixed [38], each result is internally meaningful but the magnitudes are not comparable across papers. They were obtained on different tasks with different reward definitions and different frontier models, and absolute scores drift as base models improve. Our claim is therefore qualitative: across heterogeneous settings, the runtime that surrounds a fixed model is a large and controllable source of variance. We do not claim a universal effect size, and readers should resist arithmetic across our citations. Genuinely model-held-fixed ablations remain comparatively rare; the cleanest we have are scaffold comparisons on a single coding benchmark [38], configuration-sweeping diagnostics [43], and module ablations of harness components [21]. The conclusion that the harness materially moves capability is well supported in direction but under-supported in precise quantity, and the field would benefit from far more controlled, single-variable harness ablations.

6.2 The operating-systems analogy is heuristic

The CPU/RAM/disk/syscall/kernel/scheduler/observability/recovery mapping is an organizing device, not a proven isomorphism. It earns its place by making the taxonomy memorable and by suggesting where design questions live, but it can mislead if pressed too far. Agent context is not literally paged memory, tools are not literally system calls with stable calling conventions, and an orchestration loop schedules an unreliable stochastic worker rather than deterministic instructions. The analogy is a teaching scaffold for the decomposition, and like the harness components it describes, it is itself something to build and then discard once the underlying distinction (what the runtime supplies versus what the weights supply) is internalized. Where the analogy and the evidence diverge, the evidence governs.

6.3 "Build to delete" is a hypothesis

The claim that each harness component encodes an assumption about a current model weakness, so that the optimal amount of scaffolding decreases as models improve, is presented as a hypothesis with illustrative rather than conclusive support. It is consistent with capability migrating from weights into runtime infrastructure [45] and with context compression letting smaller models function as long-horizon agents [11], and with the observation that weaker and stronger models fail along different parts of a capability hierarchy [26]. But none of these establishes the monotone trajectory the slogan implies. We are not aware of a controlled longitudinal study holding a harness fixed across a model-capability axis and measuring where each component stops paying for itself. The direction is plausible and useful as a design heuristic; it is not demonstrated, and counterexamples (components that remain load-bearing regardless of model strength, such as permissions and audit) are not merely possible but expected, since governance controls exist for reasons unrelated to capability.

6.4 Configuration-level reporting has costs

Our recommendation that capability be reported at the model–harness configuration level rather than the bare model [43, 47] is not free. It enlarges the reporting surface and the combinatorial space of things to evaluate: a model crossed with context policies, tool surfaces, memory substrates, orchestration loops, and recovery strategies is a large grid, and exhaustively benchmarking it is expensive and quickly stale. There is a real risk of trading one distortion (attributing configuration effects to the model) for another (an unmanageable, irreproducible thicket of bespoke

configurations that no two groups share). We do not resolve this tension. The honest position is that disclosure of the harness is necessary for valid comparison, but that the field still needs conventions (reference harnesses, reported configuration deltas, and consistency metrics across runs [42]) to keep configuration-level evaluation tractable rather than merely correct. Finally, our governance argument that a runtime-enforced, logged control is auditable evidence whereas a prompt instruction is only behavior [6, 9] is a position grounded in the security and oversight literature, not a measured comparison of enforcement mechanisms; it states where controls should live, not how well any particular implementation performs.

7 Conclusion

This paper has argued for a single shift in the unit of analysis: for autonomous language-model agents engaged in serious long-horizon work, capability is not a property of the model but of the model–harness configuration. The language model supplies per-step competence; the harness governs what that competence is allowed to perceive, do, remember, avoid, and recover from, and how its work is evaluated. We made this precise by defining the harness and distinguishing capability-of-the-configuration from capability-of-the-model, by organizing harness components through an operating-systems analogy (context as memory, tools as system calls, files and commits as disk, permissions as kernel rings, orchestration as the scheduler, tracing and evaluation as observability, retries and rollbacks as recovery), and by synthesizing empirical evidence that scaffolding materially moves agent task success with the weights held fixed.

The evidence is consistent across independent lines of work. Inference orchestration alone shifts outcomes dramatically on the same backbone, from sampling-and-voting and tree search to reason-act interleaving and self-feedback loops [16, 31, 36, 40, 41]. Tool reliability is a property of the runtime that selects, describes, and retrieves tools, and a smaller, dynamically conditioned action surface can beat a large static one [22, 23, 29]. Context compression, persistent skills, and memory determine whether progress accumulates across turns [8, 11, 34]. Most directly, software-engineering agents with identical backbones but different scaffolds record measurably different success rates [5, 38], and configuration-level benchmarking finds substantial variation in completion, process quality, efficiency, and failure behavior across model–harness pairings [26, 43]. Harness components can be ablated as scientific objects rather than dismissed as incidental glue [21, 33].

The payoff of adopting the configuration as the unit of analysis is threefold. First, honest measurement. Reported agent numbers can be badly mis-estimated when harness, task setup, and reward design go undisclosed [47], and realistic stateful environments expose gaps that bare-model scores hide [19, 42, 46]. Capability should therefore be reported with the harness specified, ablated, and ideally open-sourced, so that a result attaches to a configuration rather than to a model name. Second, open-weight competitiveness. If harness components encode assumptions about current model weaknesses, then scaffolding is a substitute for raw scale that one builds in order to delete as models improve [11, 45]: weaker and open-weight models can be made to approach proprietary performance with sufficient retrieval, tool orchestration, and context management [24, 27], and the right amount of scaffolding is task-dependent rather than universal. Third, auditable governance. The harness is the same boundary at which controls become evidence: a permission enforced by a sandbox or access-control policy, a telemetry stream recorded at runtime, and an audit trail written by the runtime are verifiable facts, whereas an instruction in a prompt is only behavior [1, 9, 13]. This distinction is not academic for regulated deployment: effective human oversight, intervention, and override are architectural requirements on the model-plus-oversight system, not on the model in isolation [6, 17].

The harness is thus both the principal lever on performance and the locus of governance, and these two roles are inseparable: the runtime that decides what the agent perceives and may do is also the runtime that can log, constrain, and roll back those decisions. Treating it as a first-class, reportable, ablatable variable is the precondition for measuring agents honestly, for judging open-weight systems on equal footing, and for building agents whose controls can be audited rather than merely asserted. We hope the definition and taxonomy offered here give the field a shared vocabulary for that work, and that future benchmarks, model cards, and compliance artifacts come to name the configuration, not the model, as the thing whose capability they certify.

References

[1] Adam AlSayyad, Kelvin Yuxiang Huang, and Richik Pal. AgentTrace: A structured logging framework for agent system observability. *arXiv preprint arXiv:2602.10133*, 2026. URL https://arxiv.org/abs/2602.10133.

[2] Seán Boddy and Joshua Joseph. Regulating the agency of llm-based agents. *arXiv preprint arXiv:2509.22735*, 2025. URL https://arxiv.org/abs/2509.22735.

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

[4] Minh-Dung Dao, Quy Minh Le, Hoang Thanh Lam, Duc-Trong Le, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Agentic design patterns: A system-theoretic framework. *arXiv preprint arXiv:2601.19752*, 2026. URL https://arxiv.org/abs/2601.19752.

[5] Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Vijay Bharadwaj, Jeff Holm, Raja Aluri, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025. URL https://arxiv.org/abs/2509.16941.

[6] European Parliament and Council of the European Union. Regulation (eu) 2024/1689 of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act). Official Journal of the European Union, 2024. URL https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng.

[7] Jinchao Hu, Meizhi Zhong, Kehai Chen, Xuefeng Bai, and Min Zhang. Agentic tool use in large language models. *arXiv preprint arXiv:2604.00835*, 2026. URL https://arxiv.org/abs/2604.00835.

[8] Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, Ruiyao Xu, Liangwei Yang, Rui Yang, Wooseong Yang, Chin-Yuan Yeh, Hanrong Zhang, Haozhen Zhang, Siqi Zhu, Henry Peng Zou, Wanjia Zhao, Song Wang, Wujiang Xu, Zixuan Ke, Zheng Hui, Dawei Li, Yaozu Wu, Langzhou He, Chen Wang, Xiongxiao Xu, Baixiang Huang, Juntao Tan, Shelby Heinecke, Huan Wang, Caiming Xiong, Ahmed A. Metwally, Jun Yan, Chen-Yu Lee, Hanqing Zeng, Yinglong Xia, Xiaokai Wei, Ali Payani, Yu Wang, Haitong Ma, Wenya Wang, Chenguang Wang, Yu Zhang, Xin Wang, Yongfeng Zhang, Jiaxuan You, Hanghang Tong, Xiao Luo, Xue

Liu, Yizhou Sun, Wei Wang, Julian McAuley, James Zou, Jiawei Han, Philip S. Yu, and Kai Shu. Rethinking memory mechanisms of foundation agents in the second half: A survey. *arXiv preprint arXiv:2602.06052*, 2026. URL https://arxiv.org/abs/2602.06052.

[9] Zimo Ji, Daoyuan Wu, Wenyuan Jiang, Pingchuan Ma, Zongjie Li, Yudong Gao, Shuai Wang, and Yingjiu Li. Taming various privilege escalation in llm-based agent systems: A mandatory access control framework. *arXiv preprint arXiv:2601.11893*, 2026. URL https://arxiv.org/abs/2601.11893.

[10] 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? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

[11] Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon llm agents. *arXiv preprint arXiv:2510.00615*, 2025. URL https://arxiv.org/abs/2510.00615.

[12] Rafflesia Khan, Declan Joyce, and Mansura Habiba. AGENTSAFE: A unified framework for ethical assurance and governance in agentic ai. *arXiv preprint arXiv:2512.03180*, 2025. URL https://arxiv.org/abs/2512.03180.

[13] Xinfeng Li, Dong Huang, Jie Li, Hongyi Cai, Zhenhong Zhou, Wei Dong, XiaoFeng Wang, and Yang Liu. A vision for access control in llm-based agent systems. *arXiv preprint arXiv:2510.11108*, 2025. URL https://arxiv.org/abs/2510.11108.

[14] 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. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

[15] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024. URL https://arxiv.org/abs/2408.04682.

[16] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023. URL https://arxiv.org/abs/2303.17651.

[17] Bill Marino and Nicholas D. Lane. Computational compliance for ai regulation: Blueprint for a new research domain. *arXiv preprint arXiv:2601.04474*, 2026. URL https://arxiv.org/abs/2601.04474.

[18] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842*, 2023. URL https://arxiv.org/abs/2302.07842.

[19] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023. URL https://arxiv.org/abs/2311.12983.

[20] Saroj Mishra, Suman Niroula, Umesh Yadav, Dilip Thakur, Srijan Gyawali, and Shiva Gaire. SoK: Agentic retrieval-augmented generation (rag): Taxonomy, architectures, evaluation, and research directions. *arXiv preprint arXiv:2603.07379*, 2026. URL https://arxiv.org/abs/2603.07379.

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

[22] Bhrij Patel, Davide Belli, Amir Jalalirad, Maximilian Arnold, Aleksandr Ermolov, and Bence Major. Dynamic tool dependency retrieval for lightweight function calling. *arXiv preprint arXiv:2512.17052*, 2025. URL https://arxiv.org/abs/2512.17052.

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

[24] 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. *arXiv preprint arXiv:2307.16789*, 2023. URL https://arxiv.org/abs/2307.16789.

[25] Shaina Raza, Ranjan Sapkota, Manoj Karkee, and Christos Emmanouilidis. TRiSM for agentic ai: A review of trust, risk, and security management in llm-based agentic multi-agent systems. *arXiv preprint arXiv:2506.04133*, 2025. URL https://arxiv.org/abs/2506.04133.

[26] Logan Ritchie, Sushant Mehta, Nick Heiner, Mason Yu, and Edwin Chen. The hierarchy of agentic capabilities: Evaluating frontier models on realistic rl environments. *arXiv preprint arXiv:2601.09032*, 2026. URL https://arxiv.org/abs/2601.09032.

[27] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023. URL https://arxiv.org/abs/2308.12950.

[28] Anjana Sarkar and Soumyendu Sarkar. Survey of llm agent communication with mcp: A software design pattern centric review. *arXiv preprint arXiv:2506.05364*, 2025. URL https://arxiv.org/abs/2506.05364.

[29] 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. *arXiv preprint arXiv:2302.04761*, 2023. URL https://arxiv.org/abs/2302.04761.

[30] Chris Schmitz, Jonathan Rystrøm, and Jan Batzner. Oversight structures for agentic ai in public-sector organizations. *arXiv preprint arXiv:2506.04836*, 2025. URL https://arxiv.org/abs/2506.04836.

[31] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

[32] Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*, 2025. URL https://arxiv.org/abs/2501.06322.

[33] Varun Ursekar, Apaar Shanker, Veronica Chatrath, Yuan Xue, and Samuel Marc Denton. VeRO: A harness for agents to optimize agents. *arXiv preprint arXiv:2602.22480*, 2026. URL https://arxiv.org/abs/2602.22480.

[34] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

[35] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Ji-Rong Wen. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023. URL https://arxiv.org/abs/2308.11432.

[36] 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. *arXiv preprint arXiv:2203.11171*, 2023. URL https://arxiv.org/abs/2203.11171.

[37] 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. *arXiv preprint arXiv:2201.11903*, 2022. URL https://arxiv.org/abs/2201.11903.

[38] Sherman Wong, Zhenting Qi, Zhaodong Wang, Nathan Hu, Samuel Lin, Jun Ge, Erwin Gao, Wenlin Chen, Yilun Du, Minlan Yu, and Ying Zhang. Confucius code agent: Scalable agent scaffolding for real-world codebases. *arXiv preprint arXiv:2512.10398*, 2025. URL https://arxiv.org/abs/2512.10398.

[39] Bin Xu. Ai agent systems: Architectures, applications, and evaluation. *arXiv preprint arXiv:2601.01743*, 2026. URL https://arxiv.org/abs/2601.01743.

[40] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

[41] 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. *arXiv preprint arXiv:2305.10601*, 2023. URL https://arxiv.org/abs/2305.10601.

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

[43] Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, Xiangzheng Zhang, and Tong Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026. URL https://arxiv.org/abs/2605.27922.

[44] Siyun Zhao, Yuqing Yang, Zilong Wang, Zhiyuan He, Luna K. Qiu, and Lili Qiu. Retrieval augmented generation (rag) and beyond: A comprehensive survey on how to make your llms use external data more wisely. *arXiv preprint arXiv:2409.14924*, 2024. URL https://arxiv.org/abs/2409.14924.

[45] Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026. URL https://arxiv.org/abs/2604.08224.

[46] 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. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

[47] Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, Antony Kellermann, Sarah Schwettmann, Matei Zaharia, Ion Stoica, Percy Liang, and Daniel Kang. Establishing best practices for building rigorous agentic benchmarks. *arXiv preprint arXiv:2507.02825*, 2025. URL https://arxiv.org/abs/2507.02825.