
Beyond Model Scaling: Harness Engineering and the Systems Turn in Agentic AI

Shangding Gu*
UC Berkeley

Website: <https://cheetahclaws.github.io>

Abstract

This paper studies the next major bottleneck in agentic AI as *system scaling*, not only model scaling: the design of auditable, persistent, modular, and verifiable architectures around foundation models. Recent progress in large language models (LLMs) has enabled agents that use tools, retrieve information, maintain memory, and execute long-horizon workflows, yet evaluation of these agents remains largely model-centric, reducing them to final-task success or benchmark accuracy while treating memory, retrieval, tool use, orchestration, verification and governance, and deployment cost as secondary implementation details. This framing is increasingly inadequate: agent performance emerges from the interaction among the foundation model, memory substrate, context constructor, skill-routing layer that dispatches tools and subagents, orchestration loop, and verification-and-governance layer. We highlight the harness engineering and systems turn in agentic AI around three core bottlenecks, *context governance*, *trustworthy memory*, and *dynamic skill routing*, along with the orchestration loop and verification-and-governance layer that coordinate and constrain them. We further outline a research agenda for system-scaling benchmarks that measure not only one-shot task success but also trajectory quality, memory hygiene, context efficiency, communication fidelity, verification cost, and safe evolution over time. Alongside the framework, we develop and release CheetahClaws², a Python-native reference harness, and use it together with Claude Code and OpenClaw as concrete points of comparison that make harness-level systems choices explicit. Our main claim is that future progress in agentic AI will depend as much on system design as on stronger foundation models.

*This manuscript is under active development, and we welcome any constructive comments and suggestions at shangding.gu@berkeley.edu.

²<https://github.com/SafeRL-Lab/cheetahclaws>

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Related Work | 4 |
| 3 | System Scaling: A Framework for Agentic AI | 4 |
| 3.1 | Agent Harnesses as System Infrastructure | 6 |
| 3.2 | Prompt, Skill, and Memory as Temporal Layers | 7 |
| 4 | Three Bottlenecks in System Scaling | 7 |
| 4.1 | Context Governance | 7 |
| 4.2 | Trustworthy Memory | 8 |
| 4.3 | Dynamic Skill Routing and Verification | 8 |
| 5 | Toward System-Level Evaluation and Agent Evolution | 9 |
| 5.1 | From Outcome Metrics to Process Metrics | 9 |
| 5.2 | From Single Episodes to Longitudinal Evaluation | 9 |
| 5.3 | Standards for Safe Agent Evolution | 10 |
| 6 | Discussion: Alternative Views and Limitations | 10 |
| 7 | Conclusion | 11 |

1 Introduction

The dominant story of recent AI progress has been *model scaling*: larger models, more data, stronger post-training, and higher benchmark scores [27, 3, 11]. For agentic AI, this story is now incomplete. Once foundation models are embedded into tools, terminals, browsers, repositories, memory stores, and external services, their behavior is no longer determined by the model alone. It is determined by a *system*: how context is constructed, how memory is retrieved, how tools are invoked, how subagents are routed, how actions are verified, and how failures are audited.

Our key claim is therefore that **agentic AI should be studied and evaluated as a system-scaling problem, not merely as a model-scaling problem**. By *model scaling*, we refer to improvements in the standalone foundation model, including model size, training data, post-training, and raw reasoning capability. By *system scaling*, we refer to improvements in the surrounding architecture, including memory, context construction, skill routing across tools and subagents, orchestration, and verification-and-governance, and how these components adapt over time. Our claim is not that model scaling no longer matters; rather, once models reach a sufficient capability threshold, many additional gains in long-horizon agent performance increasingly depend on how the system around the model is designed.

Modern agentic systems already illustrate this transition. Production harnesses such as Claude Code [7] and OpenClaw [34] couple foundation models to tools, subagents, and persistent project memory (detailed in §3.1); research-side harnesses such as SWE-agent further show that careful tool-schema design alone can improve benchmark accuracy substantially even with a fixed backbone model [40]. These systems show that practical agent capability does not arise from next-token prediction alone, but from the interaction between the foundation model and the harness that surrounds it. The relevant object of study is therefore not simply a model plus prompt, but a structured execution system.

This perspective is highlighted by recent empirical findings. A field-level analysis of agent benchmarks finds that many results do not separate capability from costs, prompting strategy, and demonstrations, and become non-Pareto-optimal once these factors are controlled [18]. Consistent with this, redesigning the agent–computer interface alone, while holding the underlying model fixed, can substantially improve SWE-bench accuracy [40]. Thus, what is often reported as a model score is in fact a model-plus-harness score. Context length is another example: larger context windows do not guarantee effective information access, because attention dilutes over long inputs [12], and models often prefer evidence at the start or end of the context rather than in the middle [22]. Multi-agent systems show a similar pattern: they can outperform single agents on breadth-first tasks but introduce coordination failures that single-agent metrics miss [9, 5]; we return to this in §5.2. Realistic agent benchmarks such as GAIA [26], τ -bench [43], and Terminal-Bench [25] further show that frontier models struggle when evaluation moves from one-shot prompting to multi-step interaction with tools, environments, and users. In particular, τ -bench shows that agents that look strong under single-shot pass rates can collapse under pass^k , the probability of succeeding on k independent rollouts. This exposes a reliability gap that endpoint accuracy hides.

These findings suggest that we need to rethink several parts of the agent system. Prompt engineering [37] remains useful for local control, but long-horizon performance increasingly depends on reusable skills, persistent memory, disciplined context construction, and verification-aware execution. The key issue is not only context size, but *context governance*: what should be retrieved, compressed, ordered, refreshed, trusted, and kept active at each step. Memory is not merely a storage layer; the harder problem is memory *quality*, including what to store, what to discard, how to retrieve the right information at the right time, and how to avoid staleness, drift, contamination [1], and over-generalization. Multi-agent systems are not automatically collaborative; reliable collaboration requires explicit communication protocols and uncertainty sharing [13], which we expand on in §5.2. Finally, the field still lacks a mature framework for *agent evolution* over time, including how agents should update skills, refine memory, communicate across roles, and remain auditable as they adapt.

This paper makes three contributions. *First*, we develop a systems-centered framing of agentic AI in which progress depends on system scaling, not only model scaling. *Second*, we propose a framework that separates base-model reasoning from system factors including memory, context construction, skill routing, orchestration, and verification-and-governance. *Third*, we outline an evaluation agenda for agentic systems, proposing that future benchmarks should measure process-level and longitudinal

properties such as trajectory quality, memory hygiene, context efficiency, verification cost, safe evolution, and robustness under repeated use. Our main claim is simple: the next major bottleneck in agentic AI is not only how powerful the model is, but how well the system around the model manages memory, context, skill routing across tools and subagents, orchestration, verification and governance, and adaptation. To make the discussion concrete, we develop CheetahClaws, a Python-native reference harness, and compare it against Claude Code and OpenClaw, treating their harness-level design choices as instances of the system-scaling variables identified by our framework.

2 Related Work

Agentic coding systems and harness engineering. Modern coding agents follow a line of work on tool-using language models, beginning with interleaved reasoning-and-acting policies such as ReAct [44], self-taught tool invocation [31], and verbal self-correction loops [32]. Production systems such as Claude Code [7, 8] and Codex-style “harness engineering” [30] package these primitives into programmable agent runtimes with tools, subagents, hooks, and persistent project memory. A parallel research line targets software engineering specifically, including SWE-agent’s agent-computer interface, which shows that carefully designed tool schemas can by themselves move benchmark accuracy substantially even with a fixed backbone model [40]. Most of this work, however, reports results at the level of individual model variants; comparatively little attention has been paid to the *harness itself* as a controllable, reproducible object of study, which is the vantage we adopt throughout this paper.

Context, memory, and retrieval. Retrieval-augmented generation [20] showed that augmenting parametric language models with external non-parametric memory can substantially improve knowledge-intensive generation and question answering. And following work studies memory as a system component, including MemGPT’s hierarchical memory management [29] and Voyager’s growing skill library for open-ended exploration [35]. At the same time, recent analyses show that longer context windows come with their own failure modes such as privacy drift [12], and that agents still need calibrated uncertainty to decide when to retrieve at all [13]. These results motivate our treatment of context, memory, and retrieval as a *context-governance* problem rather than as independent capabilities.

Skills and multi-agent coordination. Reusable skills have emerged as a way to offload recurring behavior from prompts into durable, callable components [19, 10, 35], extending earlier work on chain-of-thought prompting [36] and prompt-pattern catalogs [37]. In parallel, multi-agent frameworks such as AutoGen [38], MetaGPT [14], and CAMEL [21] formalize agent-to-agent communication, while Anthropic reports substantial gains from orchestrator-plus-subagent configurations on breadth-first research tasks [9]. Complementary work studies how population diversity [42, 45] and negotiation-style frameworks [23] shape collective behavior, and how such agents compose into a broader “agentic web” [41]. Our framing treats skills and delegation jointly as the *skill* lever and emphasizes that skill routing under heterogeneous subagents, rather than the existence of skills or subagents, is the next open systems bottleneck.

Benchmarks, governance, and agent evolution. A growing line of work evaluates agents as systems through executable, multi-step benchmarks [17, 24, 46, 25], alongside broader surveys of LLM-based agents [39] and catalogues of agentic safety threats [28]; yet single-episode success still dominates the reported metrics, leaving memory quality, context efficiency, communication fidelity, and safe evolution under repeated use largely unmeasured (we return to these in §5). Compared to these lines of work, our contribution is to reframe prior developments through a systems-scaling perspective and to make its engineering content concrete through a comparative analysis of Claude Code, OpenClaw, and our Python-native reference harness CheetahClaws.

3 System Scaling: A Framework for Agentic AI

We use the term *harness* throughout this paper to refer to the deterministic substrate around a foundation model: the tool interface, control loop, context constructor, memory store, skill-routing mechanism, and verification-and-governance layer that together mediate between user intent, model

outputs, and the external environment. The harness is what model scaling does not include and what system scaling targets.

We use *system scaling* to denote improvements in this harness that determine how information, computation, authority, and verification are allocated over time. Under this view, an agent is not simply a model with a prompt, but a system composed of six interacting components: a reasoning substrate (\mathcal{R}), a memory store (\mathcal{M}), a context constructor (\mathcal{C}), a skill-routing layer (\mathcal{S} , which dispatches tools and subagents), an orchestration loop (\mathcal{O}), and a verification and governance layer (\mathcal{G}). Let performance over a horizon H be

$$\mathcal{P}_H = \Phi(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{S}, \mathcal{O}, \mathcal{G}), \quad (1)$$

where \mathcal{R} denotes base reasoning quality, \mathcal{M} memory quality, \mathcal{C} context-construction quality, \mathcal{S} skill selection and composition quality, \mathcal{O} orchestration quality, and \mathcal{G} governance quality. Model scaling primarily improves \mathcal{R} ; system scaling improves $\mathcal{M}, \mathcal{C}, \mathcal{S}, \mathcal{O}$, and \mathcal{G} . The main claim of this paper is that, once models reach a sufficient capability level, long-horizon agent performance may be limited not only by \mathcal{R} itself, but also by the surrounding system factors. A useful further factorization is

$$\mathcal{M} = (\text{precision, durability, retrievability, verifiability}), \quad (2)$$

$$\mathcal{C} = (\text{relevance, compactness, traceability, refresh policy}). \quad (3)$$

Each factor names a system-level lever, not a hidden engineering detail. Figure 1 sketches how these components interact: the orchestration loop \mathcal{O} wraps a flow in which \mathcal{C} draws from \mathcal{M} to assemble inputs for \mathcal{R} , \mathcal{S} dispatches tools and subagents, and \mathcal{G} gates both intermediate reasoning and external action before any verified result is written back to memory.

Status of the decomposition. Equation 1 is a conceptual organization rather than a quantitative model: Φ has no closed form, the factors are not strictly orthogonal, and we do not claim they jointly determine \mathcal{P}_H as a measurable equation. What we do claim is that each factor names a distinct point of *intervention*, a place where engineering or research effort changes long-horizon behavior, and that existing discussions frequently fail to distinguish between them. We choose these six axes because each one can be changed, turned off, or measured on its own, while keeping the same foundation model. For instance, run \mathcal{O} in a one-shot loop, or turn \mathcal{G} off, and the same $\mathcal{R}, \mathcal{C}, \mathcal{S}$ start to act like noticeably different agents. Among the six, \mathcal{R} and \mathcal{C} are the hardest to separate (a stronger reasoning substrate can compensate for noisier context, and vice versa), while \mathcal{M} and \mathcal{G} are the easiest to isolate, since they govern writes and audit trails that exist independently of any single inference step.

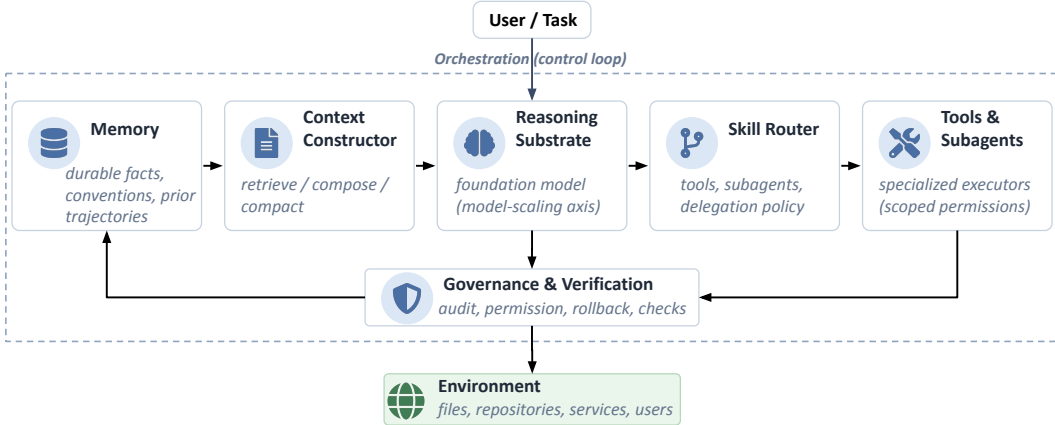


Figure 1: A six-component view of an agentic system: $\mathcal{P}_H = \Phi(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{S}, \mathcal{O}, \mathcal{G})$. The orchestration layer (\mathcal{O}) wraps a control loop in which the context constructor (\mathcal{C}) draws from durable memory (\mathcal{M}) and the current task to assemble inputs for the reasoning substrate (\mathcal{R} , i.e. the foundation model). The skill router (\mathcal{S}) dispatches tools or subagents; their effects on the environment, together with the model’s intermediate steps, are gated through verification and governance (\mathcal{G}) before they become permitted actions or verified memory write-backs. Model scaling improves \mathcal{R} ; system scaling improves $\mathcal{M}, \mathcal{C}, \mathcal{S}, \mathcal{O}$, and \mathcal{G} .

3.1 Agent Harnesses as System Infrastructure

Modern agent harnesses such as Claude Code [7] and OpenClaw [34] are better understood as *systems infrastructures* rather than simple model interfaces: their behavior depends not only on the underlying language model, but on the surrounding tool interface, execution loop, context constructor, memory substrate, and orchestration policy. Claude Code in particular benefits from substantial harness engineering [30, 6]: it bundles tools for codebase navigation, file editing, and command execution; dispatches specialized subagents with their own context windows, prompts, and permissions; and adopts a hybrid context strategy that loads persistent project guidance up front while retrieving information just in time through `glob/grep`-style tools.³ What distinguishes modern agentic coding systems from classic code assistants is therefore not stronger token-level generation alone, but the presence of an execution harness that supports tool use, iterative verification, and task decomposition.

These details matter because they reveal the real source of capability. The relevant stack is not simply a base model plus prompt, but the six interacting components $(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{S}, \mathcal{O}, \mathcal{G})$ introduced in Equation 1. These are not minor implementation details. They determine what information is available at decision time, how external actions are executed and verified, and how progress accumulates across turns. As a result, they increasingly govern task-level performance in long-horizon settings. Once these components are treated as first-class objects, the key research question shifts from “*How do we prompt the model better?*” to “*How do we allocate computation across memory, retrieval, tools, and subagents over time?*”

Table 1: Illustrative harness design patterns. The point is not to rank systems, but to show that comparable agent primitives can be governed differently under different deployment priorities.

| Systems problem | Claude Code | OpenClaw | CheetahClaws |
|--------------------|---|--|--|
| Implementation | TypeScript | TypeScript | Python |
| Primary setting | Vendor-scale coding agent | Personal assistant harness | Research reference harness |
| Context governance | Vendor-managed | Channel-scoped | Explicit and editable |
| Memory | CLAUDE.md + auto-extracted session notes + per-turn tools | Per-channel JSONL + LanceDB vector store | User/project Markdown with per-entry confidence and age fields |
| Skill routing | Curated extension boundary | Integration-driven | Modular |
| Governance | Centralized control | User-owned boundary | Transparent and reproducible |

A natural skeptical view holds that, once the foundation model is held fixed, most harnesses collapse to the same tool loop, with only superficial differences between them. We show instead that the same core systems problems, context governance, memory trust, skill routing, and auditability, admit substantially different solutions depending on deployment priorities. Table 1 sketches three illustrative design points built around comparable frontier-model capabilities: **Claude Code**, a production-grade vendor harness; **OpenClaw**, a community TypeScript harness for multi-channel personal assistance; and **CheetahClaws**, a Python-native reference harness used here as an open illustrative design point. Two observations follow. First, the three systems converge on a shared systems decomposition, all can address context governance, memory trust, skill routing, and auditability, even though they instantiate these levers differently, suggesting that these are intrinsic problems of agentic AI rather than product-specific artifacts. Second, their main differences are driven less by the foundation model than by deployment priorities: vendor-scale systems prioritize reliable use, personal-assistant systems prioritize a gateway for multi-channel management, and research-oriented harnesses prioritize transparency and reproducibility. The remainder of the paper makes these levers concrete: §4 expands the three bottlenecks of context, memory, and skill, and §5 discusses how to evaluate and govern their evolution.

Notably, only CheetahClaws stores per-entry confidence and freshness as explicit fields in its memory representation, operationalizing the trust axes of §4.2. Claude Code and OpenClaw treat memory as flat Markdown or vector-indexed log, with trust metadata present only implicitly in access patterns.

³See documentation at <https://code.claude.com/docs/en/overview> (overview), <https://code.claude.com/docs/en/sub-agents> (subagents), and <https://platform.claude.com/docs/en/agent-sdk/python> (SDK).

Table 2: Prompt, skill, and memory as three core axes of system scaling in long-horizon agents.

| Lever | Timescale | Primary role | Typical failure mode |
|--------|--------------|--|--|
| Prompt | Local | Specify current goal, constraints, and style | Brittle over long horizons; poor transfer |
| Skill | Task-level | Reusable procedure or workflow pattern | Wrong routing or poor composition |
| Memory | Longitudinal | Preserve durable facts and prior experience | Drift, over-generalization, pollution (durability / precision / verifiability) |

CheetahClaws is therefore not just a research illustration but the harness whose data model most closely tracks the framework this paper proposes.

3.2 Prompt, Skill, and Memory as Temporal Layers

We interpret prompt, skill, and memory as three primary *temporal* axes of system scaling in agentic AI. This view is complementary to Equation 1: skill corresponds to \mathcal{S} and memory to \mathcal{M} , while prompt sits inside each per-turn output of the context constructor \mathcal{C} ; the orchestration \mathcal{O} and verification and governance \mathcal{G} layers determine how the three are sequenced and verified over time. As shown in Table 2, they operate at different temporal scales and support different forms of adaptation.

Prompt. Prompt is the short-horizon control interface. It specifies the immediate role, constraints, and objective. Prompting is flexible and cheap, but also brittle: it does not by itself create persistence, transfer, or reliable long-horizon structure.

Skill. A skill is a reusable execution pattern. In practice, a skill may appear as a workflow template, a tool-use routine, a specialized subagent, or a versioned bundle of instructions and scripts. OpenAI’s recent discussions of skills for coding agents indicate this direction: durable workflows are separated from one-off prompts and attached to the environment as reusable components [19, 10]. Skills make behavior more reusable, but introduce a routing problem: the agent must decide which skill to invoke, when to switch skills, and how to compose multiple skills in one trajectory.

Memory. Memory is the longitudinal layer. It stores what should persist across turns or sessions: project conventions, user preferences, stable facts about the environment, prior failures, and distilled structure from earlier work. Memory is essential for repeated tasks, but it can fail along three trust axes elaborated in Section 4.2: drift (loss of durability), over-generalization (loss of precision), and pollution (loss of verifiability).

These three levers are complementary rather than interchangeable. Prompt controls *what to do now*; skill controls *how to do this class of things*; memory controls *what should survive over time*. A robust agent is therefore not merely well prompted. It is well prompted *and* appropriately skilled *and* selectively grounded in durable memory.

4 Three Bottlenecks in System Scaling

We now expand three system factors from Equation 1 where model scaling alone has been least sufficient: context construction \mathcal{C} , memory \mathcal{M} , and skill routing \mathcal{S} , tightly coupled to verification and governance \mathcal{G} . Each subsection names four subaxes of its component, the dominant failure mode, and the system move that addresses it; Table 3 summarizes the three.

4.1 Context Governance

The hard problem of context is not capacity, but *governance*. From the four axes of \mathcal{C} in Equation 2, an effective context assembly is jointly *relevant* to the current task, *compact* (no more than the minimum sufficient set), *traceable* to its sources, and refreshed against a moving environment. Larger windows raise the ceiling on capacity, but enforce none of these conditions.

Table 3: Three bottlenecks of system scaling. Each subsection names four subaxes of one component, a characteristic failure mode, and the system move that addresses it.

| Component | Subaxes | Failure mode | System move |
|---------------------------------|--|-------------------------|--|
| \mathcal{C} governance (§4.1) | relevance, compactness, traceability, refresh | exposure without access | assembly as a policy; persistent priors plus just-in-time refresh |
| \mathcal{M} trust (§4.2) | precision, durability, retrievability, verifiability | stale-but-confident | trust re-established at retrieval; periodic verification against the environment |
| \mathcal{S} routing (§4.3) | specificity, selectivity, composability, verifiability | confident-but-unchecked | adaptive routing coupled with explicit post-condition checks |

The threat we are guarding against is *exposure without access*: as context grows, the model sees more tokens but does not necessarily attend to the right ones. Relevant evidence competes with low-value padding (signal dilution [12]), task-relevant structure is buried in unorganized text, and token salience is driven by local statistics rather than decision importance. Long context is not good context; tokens added without governance often degrade performance rather than improve it.

The system move is to treat each turn’s context as the output of a selection policy, not a fixed buffer. The policy should weight semantic relevance, penalize bulk against a token budget, prefer recently validated content, and record provenance so failures can be attributed at audit time. The right systems question is therefore not how many tokens the model can hold, but how the system constructs the *minimum sufficient context* for the current subproblem.

4.2 Trustworthy Memory

The hard problem of agent memory is not storage, but *trust*. Matching three of the four axes of \mathcal{M} in Equation 2, a memory item earns trust when it is *precise* within a defined scope, remains *durable* (its target has not silently drifted), and is *verifiable* against the current environment. The fourth axis, retrievability, governs whether that trust can be used at acceptable cost, but is not itself part of trust.

The threat we are guarding against is *stale-but-confident*. A note that was correct at one point, say “the data loader is defined in `utils/loader.py`”, can become flatly wrong after a refactor without any change to its wording. Semantic search and reuse statistics still rank it highly, but its target has drifted, and acting on it is now destructive (calling a deleted symbol, or reintroducing a fixed regression). The failure mode is asymmetric: stale memory rarely prevents retrieval, but regularly leads the agent to act confidently on invalidated assumptions.

The system move is to make trust a runtime decision, not a property of the stored item. Retrieval should weight a staleness penalty (against the time of last verification) and a confidence-gated risk term alongside any relevance score, and should treat the retrieved content as a hypothesis until re-checked against the live environment. Claude Code realizes this coupling by pairing persistent project context (`CLAUDE.md`) with just-in-time file navigation through built-in tools, which refreshes each item against the live repository [2, 16, 4].⁴ Durable memory without periodic verification accumulates undetected drift; environment-only search without distilled priors discards every prior verification. Trustworthy memory keeps both: it retains accumulated verification while bounding drift.

4.3 Dynamic Skill Routing and Verification

The hard problem of skill is not having skills, but routing and checking them. Extending the factorization in Equation 2 to \mathcal{S} , effective skill use requires four conditions: each skill is *specific* about its capability scope, the routing policy is *selective* in invoking the right skill, the skill set is *composable* (one skill’s post-conditions feed the next), and every skill output is *verifiable* against an explicit check.

The threat we are guarding against is *confident-but-unchecked*: a specialized subagent can return plausible output that no downstream layer validates. As specialized skills multiply, the failure mode

⁴<https://claude.ai/public/artifacts/f498a4cc-4c45-481c-a6dd-8e1d196dadbd0>

shifts from a missing capability to a present-but-unverified one. This is the symmetric form of stale-but-confident memory in §4.2: both let the agent act on a claim whose truth condition was never re-established.

The system move is to treat routing as a learned policy, not a fixed manifest, coupled with verification at every step. Dynamic skill routing is the analogue of scheduling in operating systems: raw skill capacity exists, but useful work depends on allocating it to the right specialized pathway at the right time. The open research direction is to make this allocation adaptive (online estimates of subtask type, confidence-aware escalation, mixture-style composition, and policies optimized for verified rather than fluent intermediate outputs), and to make post-condition checking first-class next to the skill body. In the notation of Equation 1, \mathcal{S} and \mathcal{G} are therefore not independent: scaling skill quality without scaling verification produces faster but less reliable progress.

5 Toward System-Level Evaluation and Agent Evolution

5.1 From Outcome Metrics to Process Metrics

Benchmarks for agentic AI have improved rapidly, and the current generation already gets several important things right. SWE-bench [17] demonstrated that executable, repository-level tasks can be evaluated automatically through their own test suites, anchoring agent evaluation in real codebases rather than synthetic problems; AgentBench [24] pushed evaluation across diverse interactive environments rather than a single domain; WebArena [46] did the same for browser-based agents under realistic distributions; and Terminal-Bench [25] most recently introduces hard, environment-grounded terminal tasks with per-task verification. These benchmarks have collectively moved evaluation away from static next-token prediction and toward multi-step execution against real artifacts, and our claim is not that they are wrong.

What they are, however, is *insufficient* for system-scaled agents. As noted in §1, single-score reporting cannot separate gains from a stronger model from gains from a better harness. The gap widens in long-horizon and multi-agent settings, because small system choices, which files are inspected first, which facts are retained, when tests are run, how failed actions are corrected, each compound across a trajectory (see §5.2 for the multi-agent case). Endpoint metrics also understate cost and risk: two agents may both solve a task while differing sharply in tokens, tool calls, retries, failed edits, human interventions, and whether the trajectory is auditable, factors that determine latency, monetary cost, user trust, reproducibility, and safety in deployment.

A stronger protocol should therefore report *outcome metrics* (whether the task was solved) jointly with *process metrics* (how much context and computation were used, how the trajectory unfolded, what was retrieved and verified, and how risk was incurred), so that the system factors in Equation 1 can be measured rather than hidden. The aim is to extend the evaluation surface that SWE-bench, AgentBench, WebArena, and Terminal-Bench have opened up, not to replace it.

5.2 From Single Episodes to Longitudinal Evaluation

Multi-agent systems are increasingly central to agent design, but they must be analyzed carefully. Anthropic reports that in an internal research evaluation, a lead agent plus subagents outperformed a single-agent setup by 90.2% on breadth-first queries, and that token usage explained most of the performance variance in their BrowseComp-based analysis [9]. This is striking evidence that multi-agent architectures can buy additional useful compute through parallel context windows and decomposition. It does not, however, imply that collaboration emerges automatically: decomposition is easier than collaboration, and the dominant failures in deployed multi-agent systems map to specification mishandling, inter-agent misalignment, and verification gaps rather than to base-model reasoning weakness [5]. True collaboration requires shared state, uncertainty communication, contradiction detection, task de-duplication, and conflict resolution. The real open problem is therefore not whether multiple agents can be wired together, but whether the communication protocol between them can be made reliable enough for long-horizon work; handoffs, summaries, requests for clarification, and uncertainty reports should be treated as optimized objects rather than ad hoc prompt fragments.

This points to a more general gap: most current benchmarks reset the agent between tasks, but real agents accumulate state across sessions, conversations, and projects. They store conventions, preferences, prior failures, and reusable procedures, and the same accumulation that enables improvement

can also produce contamination, staleness, over-generalization, and privacy leakage. A one-shot evaluation cannot reveal whether an agent’s memory becomes more useful, more noisy, or more dangerous over repeated use.

As Table 4 suggests, the next generation of agent benchmarks should additionally measure repeated-use properties such as memory retrieval precision and memory hygiene, minimal-context efficiency, communication fidelity across subagents, drift across long trajectories or sessions, verification-aware recovery after stale memory or wrong routing, and safety under tool access and autonomous execution. Current evaluation often measures whether an agent can finish *a* task, but not whether it can finish similar tasks repeatedly while improving, staying grounded, and avoiding silent degradation; agent quality should therefore be evaluated as a *longitudinal systems property* rather than a one-shot completion score.

5.3 Standards for Safe Agent Evolution

A mature agent should not only act, but evolve. Yet the field lacks a standard for what persistent adaptation should mean. What should be allowed to change over time, memory only, or also routing policies, skills, and collaboration protocols? What should be fixed for auditability? What counts as safe improvement versus dangerous drift? The questions are not abstract: persistent behaviors can survive subsequent training in ways that are hard to detect from outputs alone [15], optimization against imperfect proxies leads to characterizable reward-hacking failure modes that get worse as capability grows [33], and the OWASP catalogue of agentic threats lists memory poisoning, identity spoofing, tool misuse, and goal manipulation as exploitable failure surfaces of evolving agents [28]. These are exactly the failure modes a maturity standard for agent evolution would have to control.

Table 4: Benchmark dimensions for system-scaling evaluation.

| Dimension | Current | Future |
|------------------------|---------|--------|
| One-shot completion | Yes | Yes |
| Repeated adaptation | Limited | Yes |
| Memory hygiene | Rare | Yes |
| Context efficiency | Rare | Yes |
| Communication fidelity | Rare | Yes |
| Session drift | Rare | Yes |
| Safe evolution | No | Yes |

We therefore propose that future agent systems need an explicit *agent evolution standard* built around four questions:

1. **What persists?** Memory, skills, preferences, and guardrails should be distinguished rather than merged into one undifferentiated state, so that updates to one component do not silently rewrite another.
2. **What updates?** Some components should adapt online; others should require review, replay, or stronger verification before changing, especially when changes interact with tool permissions or governance boundaries.
3. **What is measured?** Longitudinal improvement should be balanced against regression, drift, recurrence of earlier failures, and the reward-hacking patterns catalogued in [33], rather than being inferred from a single rolling success rate.
4. **What is auditable?** Memory writes, routing changes, tool permissions, and collaboration failures should leave inspectable traces, a precondition for evaluating persistent risks of the kind documented in [15, 28].

Without such standards, many so-called learning agents risk becoming opaque accumulations of prompts, notes, and heuristics rather than reliable adaptive systems.

6 Discussion: Alternative Views and Limitations

The system-scaling claim is incomplete without an honest engagement with the views it stands against. We discuss three objections.

Objection 1: Stronger models will eventually solve system problems. One objection is that system scaling is a temporary concern: as foundation models become stronger, they may learn to

manage context, memory, tools, and verification internally, without an explicit harness. We agree that model scaling will continue to improve agent behavior. However, many failures in deployed agents are not failures of next-token prediction alone. Stale memory, over-broad tool permissions, missing provenance, unverified retrieval, and unsafe action execution are system failures. A stronger model may reduce their frequency, but it does not remove the need for explicit mechanisms that govern what information is exposed, which actions are authorized, and how failures are traced. Whatever the model’s capability, an agent that can act on the world requires a system around it that decides what actions are permitted and how they are verified.

Objection 2: End-to-end training will replace modular systems. A second view is that future agents should be trained end-to-end, making explicit system components unnecessary. End-to-end training may improve coordination across components, but deployed agents still require modular boundaries. They operate over private files, credentials, tools, repositories, browsers, and external services. In these settings, auditability, permission control, rollback, and provenance are not optional. Modularity is therefore not only an engineering convenience; it is a requirement for safe and governable deployment, and an end-to-end policy still has to act through the same permission, verification, and audit surfaces we describe.

Objection 3: System-level evaluation is too expensive or environment-specific. System-level evaluation is indeed more expensive than static prompting benchmarks, and trace-level metrics are harder to standardize than endpoint accuracy. However, this is precisely why it is needed. Agents are deployed in environments where cost, latency, tool risk, memory drift, and verification overhead determine whether the system is usable. Evaluation protocols should expose these factors rather than abstract them away. The goal is not to replace simple benchmarks, but to complement them with measurements that reflect real agent operation.

7 Conclusion

Agentic AI is moving from isolated model inference to persistent system execution. As models are embedded into tools, memory stores, repositories, browsers, subagents, and external services, their behavior is increasingly shaped by the architecture around them. This paper has shown that future progress therefore requires *system scaling*: improving how agents construct context, maintain trustworthy memory, route skills, verify actions, govern tools, communicate across roles, and evolve over time. Claude Code, OpenClaw, and CheetahClaws illustrate that comparable models projected onto different harnesses produce qualitatively different agents, and that the harness, not the model alone, is now a primary source of practical capability.

This does not diminish model scaling. Stronger foundation models remain essential, but model capability alone is no longer a sufficient unit of analysis for long-horizon agents. A mature science of agentic AI must study the full execution system: what it remembers, what it retrieves, what it exposes to the model, what actions it permits, what it verifies, and what it leaves auditable. Future benchmarks should therefore treat memory, context, skill routing across tools and subagents, orchestration, and verification-and-governance as first-class objects of design and evaluation, rather than measuring only one-shot success.

Acknowledgements

We sincerely thank the open-source contributors to CheetahClaws for their valuable issue reports, pull requests, suggestions, and comments. We also thank Prof. Dawn Song and Prof. Costas Spanos for their great support.

References

- [1] Ahmad Al-Tawaha, Shangding Gu, Peizhi Niu, Ruoxi Jia, and Ming Jin. Remembering more, risking more: Longitudinal safety risks in memory-equipped llm agents, 2026.
- [2] Anthropic. Effective Context Engineering for AI Agents. <https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents>, September 2025. Anthropic Engineering Blog. Accessed: 2026-04-18.

- [3] Anthropic. Introducing Claude Opus 4.7. <https://www.anthropic.com/news/claude-opus-4-7>, April 2026. Accessed: 2026-04-01.
- [4] Anthropic. Manage Claude’s Memory (Claude Code Documentation). <https://docs.claude.com/en/docs/claude-code/memory>, 2026. Documents CLAUDE.md instruction files and auto memory. Accessed: 2026-04-18.
- [5] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*, 2025.
- [6] Claude. Claude Code. <https://claude.com/product/claude-code>, Sep 2025. Accessed: 2026-04-02.
- [7] Claude. Claude Code Understand how to integrate Claude Code into your development workflows with best practices and real-world examples. <https://claude.com/product/claude-code>, 2025. Accessed: 2026-04-02.
- [8] Claude. Enabling Claude Code to work more autonomously. <https://www.anthropic.com/news/enabling-claude-code-to-work-more-autonomously>, Sep 2025. Accessed: 2026-04-02.
- [9] Claude. How we built our multi-agent research system. <https://www.anthropic.com/engineering/multi-agent-research-system>, Jun 2025. Accessed: 2026-04-02.
- [10] Emre Okcular. Skills in OpenAI API. https://developers.openai.com/cookbook/examples/skills_in_api, Feb 2026. Accessed: 2026-04-02.
- [11] Google. Gemini 3.1 Pro: A smarter model for your most complex tasks. <https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-pro/>, february 2026. Accessed: 2026-04-01.
- [12] Shangding Gu. Long context, less focus: A scaling gap in llms revealed through privacy and personalization. *arXiv preprint arXiv:2602.15028*, 2026.
- [13] Junyu Guo, Shangding Gu, Ming Jin, Costas Spanos, and Javad Lavaei. Llms should express uncertainty explicitly. *arXiv preprint arXiv:2604.05306*, 2026.
- [14] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.
- [15] Evan Hubinger, Carson Denison, Jesse Mu, Mike Lambert, Meg Tong, Monte MacDiarmid, Tamera Lanham, Daniel M. Ziegler, Tim Maxwell, Newton Cheng, Sandipan Kundu, Tom Brown, and Ethan Perez. Sleeper agents: Training deceptive LLMs that persist through safety training. *arXiv preprint arXiv:2401.05566*, 2024.
- [16] Isabella He. Context engineering: memory, compaction, and tool clearing. <https://platform.claude.com/cookbook/tool-use-context-engineering-context-engineering-tools>, March 2026. Accessed: 2026-04-01.
- [17] 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? In *International Conference on Learning Representations (ICLR)*, 2024.
- [18] Sayash Kapoor, Benedikt Stroebel, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.
- [19] Kazuhiro Sera. Using skills to accelerate OSS maintenance. <https://developers.openai.com/blog/skills-agents-sdk>, Mar 2026. Accessed: 2026-04-02.

- [20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [21] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for “mind” exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [22] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.
- [23] Xianyang Liu, Shangding Gu, and Dawn Song. Agenticpay: A multi-agent llm negotiation system for buyer-seller transactions. *arXiv preprint arXiv:2602.06008*, 2026.
- [24] 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. In *International Conference on Learning Representations (ICLR)*, 2024.
- [25] Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.
- [26] Grégoire Mialon, Clémentine Fourier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.
- [27] OpenAI. Introducing GPT-5.4. <https://openai.com/index/introducing-gpt-5-4/>, March 2026. Accessed: 2026-04-01.
- [28] OWASP GenAI Security Project. Claude Code Understand how to integrate Claude Code into your development workflows with best practices and real-world examples. <https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/>, April 2025. Accessed: 2026-04-20.
- [29] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.
- [30] Ryan Lopopolo. Harness engineering: leveraging Codex in an agent-first world. <https://openai.com/index/harness-engineering/>, Feb 2026. Accessed: 2026-04-02.
- [31] 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [32] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [33] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward hacking. In *Advances in Neural Information Processing Systems*, 2022.
- [34] Openclaw Team. Openclaw — personal ai assistant. *github*, 2026.
- [35] 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.

- [36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [37] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.
- [38] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.
- [39] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.
- [40] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent–computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [41] Yingxuan Yang, Mulei Ma, Yuxuan Huang, Huacan Chai, Chenyu Gong, Haoran Geng, Yuanjian Zhou, Ying Wen, Meng Fang, Muhao Chen, et al. Agentic web: Weaving the next web with ai agents. *arXiv preprint arXiv:2507.21206*, 2025.
- [42] Yingxuan Yang, Chengrui Qu, Muning Wen, Laixi Shi, Ying Wen, Weinan Zhang, Adam Wierman, and Shangding Gu. Understanding agent scaling in llm-based multi-agent systems via diversity. *arXiv preprint arXiv:2602.03794*, 2026.
- [43] 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.
- [44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- [45] Rui Ye, Xiangrui Liu, Qimin Wu, Xianghe Pang, Zhenfei Yin, Lei Bai, and Siheng Chen. X-mas: Towards building multi-agent systems with heterogeneous llms. *arXiv preprint arXiv:2505.16997*, 2025.
- [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. In *International Conference on Learning Representations (ICLR)*, 2024.