agentic LLM systems / LLM agents survey (interfaces, planning, tool use, multi-agent, evaluation, safety)

January 22, 2026

Contents

Abstract

Large language model (LLM) agents embed an LLM in a closed-loop system that observes, decides, and acts through tools or environments. In this setting, interface contracts and evaluation protocols are part of the object of study: changing tool access, budgets, or logging policies can change what a reported success rate actually means. We survey recent agentic systems with an evidence-first lens, organizing the design space from interfaces and action spaces, through planning and memory, to adaptation and multi-agent coordination, and finally evaluation and risk. Across these lenses, we highlight protocol-sensitive comparisons (what transfers across benchmarks, and what does not) and summarize recurring failure modes that emerge at the system boundary. The result is a practical map for designing and evaluating agents, and a research agenda for making agent results more interpretable and reproducible [63, 45, 51, 34, 39, 69].

1 Introduction

LLM agents are increasingly deployed as interactive systems that must decide what to do next, call tools, recover from failures, and complete multi-step tasks under latency and cost constraints. The resulting behaviors are not determined by the base model alone: the surrounding loop, the tool/environment boundary, and the evaluation protocol jointly shape what "capable" means in practice [43, 56, 22].

We adopt a systems definition that makes this dependence explicit. An LLM agent is an LLM embedded in a control loop that (i) maintains an internal state, (ii) selects actions from a defined action space (tool calls, code edits, web interactions, environment actions), and (iii) receives observations that condition subsequent decisions. This definition separates agents from adjacent settings such as one-off tool invocation or retrieval-only augmentation, while aligning with the dominant "reason + act" lineage in which reasoning is coupled to grounded actions [63, 45, 51].

A recurring difficulty is that agent results are protocol-sensitive. Cost-aware studies show that planning depth, retries, and verification policies can move an agent along a sharp success–cost frontier, so headline success rates are hard to compare unless budgets are normalized [34, 40]. Similarly, evaluations that approximate realistic tool ecosystems highlight that API complexity, tool noise, and permission constraints can dominate outcomes, which makes "progress" inseparable from benchmark design and environment versioning [22, 61, 36].

This survey therefore uses two lenses as first-class citizens: interface contracts and evaluation/threat-model assumptions. We organize the design space from foundations and interfaces (agent-loop semantics, action spaces, and tool orchestration) to core components (planning loops and memory), then to adaptation and coordination (self-improvement and multi-agent interaction), and finally to evaluation and risk. The goal is not only to summarize mechanisms, but to make comparisons interpretable by pairing claims with the protocols and failure modes that delimit them [39, 22, 69].

Survey methodology (one-paragraph evidence policy): we queried arXiv from 2022 onward using agent-related keywords, collecting 809 candidate records and deduplicating to 800. From this pool, we curated a 220-paper core set to construct the taxonomy and the evidence base used throughout. Unless explicitly stated otherwise, our synthesis relies primarily on titles, abstracts, and arXiv metadata; when evaluation details (budgets, tool access, logging policies, threat models) are under-specified, we keep claims conservative and treat protocol descriptions as key evidence for comparison [39, 22].

Our contributions are threefold. First, we provide a taxonomy that treats the agent loop and its interfaces as the unit of analysis, making explicit how action spaces and orchestration policies constrain reliability [63, 45]. Second, we distill protocol-aware comparison axes across planning, memory, and adaptation, emphasizing how cost models and environment assumptions affect conclusions [34, 61]. Third, we integrate risk surfaces into the same evaluative frame, including tool-induced security failures and monitoring challenges that arise only in closed-loop deployments [69, 26, 25].

The remainder of the paper follows this lens order: interfaces and action spaces (Section 3), core components for long-horizon behavior (Section 4), adaptation and coordination (Section 5), and evaluation and risk (Section 6), followed by a cross-cutting discussion and conclusion.

2 Related Work

Recent surveys and overviews of agentic LLM systems have mapped out architectures, applications, and emerging design patterns, often emphasizing the agent loop as a systems object rather than a prompting trick [43, 56, 54, 38, 44]. We build on this line of work but shift the organizing principle: rather than grouping papers primarily by application domains, we position the literature through interface contracts and protocol-aware evaluation, because these assumptions frequently determine whether results are comparable across settings.

On the mechanism side, a large body of work extends reasoning beyond single-pass prompting by coupling deliberation to action. ReAct makes the loop explicit by interleaving reasoning traces with tool-grounded actions [63], while search-style deliberation explores multiple candidates or trajectories before committing to an action sequence [64]. Reflection and critique loops further demonstrate that apparent "capability" can be amplified or destabilized by how a system stores state and revises decisions over time [51]. These approaches motivate the need for protocol-aware comparisons: changing the deliberation and verification policy can change both success and cost, even when the base model is held fixed.

A complementary line of work focuses on tool use and interface learning. Training-time methods that expose explicit tool calls broaden the action space available to an agent, but they also make interface design and schema discipline central to correctness [45]. As evaluations move toward real-world tool ecosystems, papers increasingly report failures that are interface failures (routing mistakes, schema mismatches, tool-side instability) rather than purely reasoning errors, reinforcing the view that the interface contract is part of the research contribution [22, 36, 61].

Evaluation work makes the same point from the measurement side. Cost-sensitive benchmarks show that success rates must be interpreted together with budgets and search policies, because higher success can be achieved by spending more steps or tool calls [34, 40]. Benchmarks that stress realistic API complexity further highlight that tool noise, permissions, and logging assumptions can dominate outcomes, making "agent progress" inseparable from benchmark artifacts and environment versioning [22, 61, 65]. Methodology papers therefore increasingly recommend protocol templates and reporting checklists that make comparisons debuggable and reproducible [39].

Finally, security and governance work argues that agentic tool use introduces qualitatively new attack surfaces, including prompt/tool injection, tool misuse, and data exfiltration. Security benchmarks and monitoring-oriented evaluations operationalize these risks as end-to-end system properties rather than model-only attributes [69, 26, 25, 29, 7]. This reinforces our decision to integrate threat models into the same evaluation lens used for performance, rather than treating safety as an afterthought.

In summary, prior work provides strong foundations on agent architectures and applications, but cross-paper synthesis remains difficult because protocols, cost models, and threat models vary widely and are often implicit. Our survey addresses this gap by centering interface contracts and protocol-aware evaluation as the connective tissue that makes comparisons interpretable across planning, memory, adaptation, and risk.

3 Foundations & Interfaces

Interfaces define the action space an agent can execute and the observations it can reliably incorporate; this contract often determines robustness more directly than a change in the base model. Early agent designs make the loop explicit by interleaving reasoning with grounded actions, while tool-use training expands the action set beyond natural language, pushing schema design and orchestration policies to the foreground [63, 45].

Seen through this lens, comparisons hinge on what is held fixed. Tool catalogs, permissions, and error semantics can reshape outcomes in evaluations that model realistic API complexity, so interface choices are only interpretable when paired with protocol assumptions [22, 39]. The next two subsections separate loop semantics (what counts as state, action, and recovery) from interface and orchestration (how tools are selected, invoked, and audited), so later chapters can treat planning and memory as decisions made within a well-defined contract rather than as free-form prompting tricks [69, 39].

3.1 Agent loop and action spaces

The first design decision in an agent system is what counts as an action and what counts as an observation. That choice fixes the shape of the control loop—what the model is allowed to do, what feedback it can condition on, and which failures it can even detect. In practice, results are easiest to interpret when the action space is explicit and the evaluation protocol makes loop assumptions visible, a perspective reflected in early interactive agents such as ReAct and in open-world testing environments that treat tool use as part of the task rather than an implementation detail [63, 61].

A useful way to view "agent loop design" is as a bundle of interface commitments: state representation, action representation, and recovery policy. Some systems keep state largely implicit (the chat history) and let the model emit free-form action descriptions that are parsed into tool calls, while others constrain the action space with typed commands, schemas, or environment APIs. These constraints trade expressivity for debuggability: a narrower action language can reduce parsing and tool-selection ambiguity, whereas a more open action space can cover diverse tasks at the cost of higher variance and harder-to-localize failures [49, 24, 14].

This tension shows up in how agents couple reasoning to action. ReAct makes the coupling explicit and reports large gains on interactive decision-making benchmarks such as ALFWorld and WebShop (absolute success rate improvements of 34% and 10% over imitation and reinforcement learning baselines) under a fixed prompting setup [63]. In contrast, environment-first work emphasizes that the same loop abstraction can behave very differently once tool access, noise, and logging are varied, motivating "agent testing" as a first-class object rather than an afterthought [61, 52]. When loop assumptions shift—for example, whether tools fail loudly or silently, or whether observations are lossy summaries—comparisons can drift even when the model and prompts are unchanged.

A second thread treats loop design as something that can be searched or discovered rather than handcrafted. AgentSquare reports that automatically constructed agents can outperform human designs by an average of 17.2% across a set of interactive environments, suggesting that seemingly small loop choices (when to verify, how to retry, what to store as state) can dominate outcomes once held to a fixed benchmark [47]. Similarly, AgentSwift evaluates discovered agents across seven benchmarks spanning embodied, math, web, tool, and game settings and reports an 8.34% average performance gain over both automated baselines and manual agents, reinforcing the point that "agent loop" is an optimization surface, not a single canonical architecture [31, 53].

Efficiency-oriented variants highlight that loop design is inseparable from cost models. EvoRoute reports that, on agentic benchmarks such as GAIA and BrowseComp+, routing choices integrated into off-the-shelf systems can sustain or improve task success while reducing execution cost by up to 80% and latency by over 70% [70]. These numbers are less a claim about any one model than about the loop as a budgeted system: changing when the agent re-plans, re-queries tools, or short-circuits verification directly changes both measured performance and the resources consumed.

Two limitations recur across this literature. First, action spaces are often under-described: papers may not fully specify what the agent is permitted to do (and what is disallowed), which makes failure analysis and reproducibility limited even when benchmark names are shared [61, 12]. Second, evaluation protocols can hide "failure assumptions" (e.g., whether tool errors are adversarial, stochastic, or deterministic), so synthesis should treat protocol descriptions as evidence, not as boilerplate; where those assumptions are unclear, the safest takeaway is about the direction of trade-offs rather than any absolute ranking [52, 24].

Domain differences also matter for how action spaces are engineered. In geometry problem solving, for example, recent work notes that performance can still be dominated by expert systems that rely on large-scale data synthesis and search, highlighting a regime where a specialized action language and solver loop can outweigh generic tool-calling heuristics [72]. At the other extreme, open-world tool environments stress breadth: they reward agents that can adapt to changing APIs and noisy observations, even when individual actions are simple [61, 52]. This gap suggests that "agent loop" comparisons are often limited by mismatched action languages—when papers evaluate under different state and action abstractions, synthesis is clearest when conclusions are phrased as conditional trade-offs (what breaks under which interface assumptions) rather than as universal claims about which loop is best [24, 72].

Across these threads, the dominant comparison is between widening the action space and tightening the contract. Whereas open-ended action languages maximize coverage and let agents improvise in novel environments, constrained action APIs make failures easier to localize and make protocols easier to reproduce. A useful reporting norm is therefore to treat the action space itself as an evaluation artifact: specify the allowed actions, the observation channels, and the recovery semantics, so readers can tell whether an improvement comes from better decision-making or simply from a more forgiving interface.

Once the agent loop defines what actions are possible, tool interfaces determine how those actions are grounded in executable APIs and how orchestration policies manage ambiguity and failure.

3.2 Tool interfaces and orchestration

Tool use widens what an agent can accomplish, but it also creates a new bottleneck: correctness becomes a property of the interface contract as much as of the model. A minor mismatch in schemas, permissions, or error semantics can turn a capable planner into a brittle system, and reported gains can be hard to compare when tool access and orchestration policies differ across evaluations. Recent evaluation work therefore treats interface contracts and protocol definitions as part of the measurable object, not as incidental implementation detail. The key point is that interface contracts constrain which comparisons are meaningful, so orchestration results must be read through the protocol that fixes tool access and error behavior [39, 36].

At a high level, tool interfaces answer three questions: how actions are represented (free-form text versus structured calls), how tools are selected (routing and ranking), and how tool outputs are validated and incorporated into state. Some systems emphasize expressive tool descriptions and rely on the model to map language to actions, whereas others tighten the contract with typed parameters, constrained formats, or confidence/verification policies. The trade-off is familiar: looser contracts increase coverage but raise ambiguity and silent failure risk; tighter contracts reduce ambiguity but can introduce brittleness when tools evolve or when the environment deviates from assumptions [10, 62, 27].

Benchmark design has started to reflect this reality by elevating orchestration as a multi-hop systems problem. ETOM introduces a five-level benchmark that evaluates end-to-end tool orchestration within a hierarchical tool ecosystem, explicitly stressing multi-step coordination rather than single tool calls [9]. MCPAgentBench similarly anchors evaluation in real-world tool definitions, aiming to make tool-use results more reproducible by standardizing the interface artifacts that agents must operate over [36]. These efforts illustrate an important evaluation anchor: unless the tool catalog and its semantics are fixed, it is unclear whether an improvement comes from better decision-making or from a cleaner interface.

Security further raises the stakes because tool interfaces double as an input channel. An analysis of 66 real-world tools from LangChain and LlamaIndex repositories reports that 75% are vulnerable to XTHP-style attacks, suggesting that many tool ecosystems are adversarial by default unless contracts and sanitization are explicitly designed [32]. In this regime, the relevant comparison axis is not "does the agent call the tool", but "under what threat model is the call safe", including whether prompts can be injected through tool descriptions, tool outputs, or chained tool behaviors [15, 32].

Tool interfaces also interact with memory and long-horizon execution. MemTool, for example, evaluates multiple "tool modes" across 13+ LLMs on a ScaleMCP benchmark over 100 consecutive user interactions, tracking both task completion and tool removal ratios as a proxy for short-term memory efficiency [37]. This kind of protocol makes explicit that orchestration is not a single decision but a policy over time: when to call tools, what to retain as state, and when to discard or summarize intermediate artifacts all change both cost and failure modes.

Two limitations shape how to read this literature. First, interface assumptions are often implicit, which makes cross-paper synthesis limited when tool catalogs, permissions, or logging policies are not fully specified [39, 36]. Second, many benchmarks still under-exercise the adversarial and distribution-shift regimes that real deployments face, so strong performance claims should be interpreted as protocol-scoped unless they are backed by threat-model-aware evaluation and hardening analyses [32, 9, 15].

A practical implication is that "tool use" should be evaluated as a trajectory problem, not as a one-shot classification of which tool to call. Multi-hop orchestration benchmarks make this explicit by grading chains of calls and intermediate state updates under a fixed protocol, which helps separate routing quality from tool curation [9, 36]. Complementary benchmarks target more fine-grained interface skills (e.g., parameter reasoning and schema compliance), reflecting that orchestration failures often come from small contract violations rather than from missing high-level plans [27, 10]. Even with these advances, comparisons remain limited when papers change tool catalogs or hide tool behaviors behind proprietary endpoints, so the most reusable results are those that ship the interface artifacts and specify evaluation constraints in a way that later work can replay [39, 36].

A simple but high-leverage reporting practice is to publish the interface artifacts alongside results—schemas, tool descriptions, and a brief "tool card" describing expected failures and permissions. Without these artifacts, it is difficult to tell whether an orchestration gain reflects a better policy or merely a cleaner tool contract, and reproduction becomes limited to approximate reimplementations [36, 32].

Interface choices also determine how errors are surfaced and corrected. A permissive contract that silently coerces inputs can inflate success rates but hide brittleness, whereas a strict contract that fails loudly can make agents look weaker while producing more diagnosable trajectories. For synthesis, the important question is not just whether a tool was called, but whether the call was valid under a declared schema and whether the system had a principled recovery path when the interface rejected or mis-executed the request.

4 Core Components (Planning + Memory)

Once an interface fixes what actions are executable, the next bottleneck is how agents choose actions over time under uncertainty and budget constraints. Deliberate search and verification can improve correctness on long-horizon tasks, but they also increase cost and can introduce new failure modes, making success rates hard to interpret without explicit budget models [64, 34].

This chapter contrasts planning loops that search, decompose, or verify action sequences with memory mechanisms that determine what evidence the policy can condition on and how that evidence persists across steps. Keeping the evaluation protocol in view is essential: planners and memories that look strong under one cost model, tool catalog, or retrieval noise profile may not transfer under another, and methodology work increasingly treats these assumptions as part of the reported result rather than background detail [40, 39].

4.1 Planning and reasoning loops

Planning is where agents trade reliability for resources. A central tension is that deeper deliberation, explicit search, and verification can improve success on multi-step tasks, but they also increase latency and tool cost and can introduce their own failure modes (overthinking, inconsistent intermediate state, cascading retries). As a result, planning results are easiest to interpret when evaluation protocols make budgets and loop policies explicit rather than treating them as background settings [63, 46].

Moreover, one family of approaches builds planning loops by prompting and structuring the model's reasoning and action choices. ReAct interleaves reasoning traces with grounded actions, providing a simple but effective template for long-horizon interaction [63]. Hierarchical or tree-structured controllers extend this idea by imposing control flow, decomposing tasks, or exploring multiple candidates before committing to an action sequence, aiming to reduce early-commitment errors that are common in interactive environments [5, 21]. These designs often improve robustness by changing *how* decisions are made rather than by changing the base model.

A contrasting family treats planning competence as something that can be trained or optimized directly. On a complex task-planning benchmark, Hu et al. report that a 1.5B parameter model trained with single-turn GRPO can achieve a 70% success rate for long-horizon planning tasks, outperforming larger baselines up to 14B parameters under that evaluation setup [16]. Unlike prompt-structured planners, training-based approaches shift the main comparison axis toward data signals, reward definitions, and generalization: gains may reflect better policy learning, but they also risk being benchmark- or protocol-specific if the training distribution matches the evaluation too closely.

Safety-aware planning highlights that "good plans" are not always "safe plans". SafeAgentBench is designed to evaluate embodied agents in interactive simulations under both explicit and implicit hazards, forcing planners to account for risk while pursuing task goals [65]. This changes what evaluation should measure: success under hazards is not only a function of planning depth, but also of how agents model constraints, detect unsafe states, and decide when to abort or ask for clarification.

Because planning loops can fail in rare but consequential ways, evaluation increasingly borrows from testing methodology. Zhou et al. report that seed test case generation can yield a 2–2.5x boost to the coverage of risk outcomes and tool-calling trajectories across diverse agent settings, suggesting that benchmark scores alone may miss important tail failures [75, 18]. In this view, the evaluation protocol is part of the planner: what scenarios are sampled, what constitutes a failure, and how trajectories are logged can change which planning strategies appear reliable.

Two limitations temper strong conclusions in this space. First, planning comparisons are often under-specified with respect to budgets, retries, and verification policies, so results can be difficult to transfer across benchmarks even when task names match [46, 63]. Second, as planners become more structured, it becomes unclear whether improvements come from better decision-making or from better tooling around the model (control flow, gating, testing harnesses); where ablations are missing, the safest interpretation is about the *direction* of trade-offs rather than about any universal ranking [5, 75].

Unified evaluations that hold the loop constant can also change what we learn about planning. Seo et al. evaluate 16 agents under a shared ReAct-style framework and report distinct capability and limitation profiles across models, suggesting that "planning quality" is not a single scalar but a bundle of behaviors (decomposition, recovery, verification) that surface differently under the same protocol [46, 63]. This kind of setup helps interpret training-based gains as well: a 70% success rate on one long-horizon benchmark is informative only insofar as the budget, retries, and tool assumptions align with the target deployment regime [16, 65]. When those assumptions are mismatched, the right conclusion is often about which planning *strategy* is robust under which constraints, not about whether one model "can plan" in general.

Finally, planning benchmarks differ in what they reward: some score only end-task success, while others expose intermediate safety violations, dead-ends, or hazard encounters. This distinction can flip conclusions about which planners are "better", because conservative policies may look weak under pure success metrics but strong under hazard-aware protocols that penalize unsafe trajectories [65, 46]. In that sense, planning is not only about generating a plan but about defining what counts as an acceptable trajectory under the evaluation rules, and current evidence remains limited when those rules are not reported precisely [18, 75].

A key contrast, then, is between planners that externalize structure in prompts and control flow and planners that internalize it through training. Prompt-structured planners can be adapted quickly to new tasks and constraints, whereas training-centric planners may achieve stronger in-distribution reliability under a fixed benchmark but risk locking in protocol-specific behaviors. Making this contrast visible requires reporting not only final success but also intermediate decision traces, budget usage, and the failure modes the planner is designed to avoid.

Planning determines which actions to take under uncertainty, while memory determines what evidence those decisions can reliably condition on across long horizons.

4.2 Memory and retrieval (RAG)

Memory turns an agent from a single-trajectory solver into a stateful system, but it also introduces a new failure surface: retrieval can be wrong, stale, or adversarial, and the agent may not know which. The core tension is that richer memory can improve long-horizon coherence and tool use, whereas it can also amplify compounding errors and make evaluation less interpretable unless retrieval assumptions are made explicit [17, 3].

A practical distinction is between ephemeral working memory (scratchpads, short-term summaries) and persistent memory (indexes, long-term stores, artifact repositories). Ephemeral memory is cheap and easy to reset between episodes, but it can force repeated tool calls and re-derivation; persistent memory can reduce redundancy, yet it raises questions about what is written, when it is retrieved, and how leakage or contamination is prevented across tasks. These design choices are part of the protocol: changing write or retrieval policy can change both success and cost even when the planner is unchanged [63, 66].

In code-centric settings, summarization and representation choices can dominate whether retrieval is useful. Meta-RAG proposes using summaries to condense codebases by an average of 79.8% into a compact, structured natural-language representation, aiming to make retrieval more stable and context-efficient for downstream reasoning [55]. This kind of approach illustrates an evaluation anchor for memory systems: improvements should be tied to task families (e.g., code comprehension or modification) and to measurable properties of the memory (compression ratio, retrieval accuracy), not only to end-task success.

Tooling scaffolds also function as memory. Verma et al. evaluate a coding-agent setup on N=5 context-intensive SWE-bench Lite instances using an optimized scaffold with persistent bash state and a string-replacement editor, effectively changing what information is carried forward across tool calls and edits [57]. Even when models are fixed, such scaffolding can alter the agent's effective memory and error recovery behavior, which complicates head-to-head comparisons unless the toolchain and state persistence policy are reported.

Memory increases risk surfaces because it is an input channel. MSB catalogs 12 agent-specific attack patterns, including retrieval injection and prompt injections embedded in tool descriptions, emphasizing that retrieval and tool outputs can be manipulated to steer the loop [69]. Defensive work like Progent reports reducing attack success rates to 0% on agent benchmarks (e.g., AgentDojo, ASB, AgentPoison) while preserving utility and speed, highlighting a contrast: the same memory mechanisms that increase capability can widen the adversarial surface unless guarded end-to-end [50, 69].

Two limitations recur. First, many memory mechanisms are evaluated under narrow task distributions, so it remains unclear how robust they are to retrieval noise, distribution shift, or adversarial inputs outside the benchmark protocol [17, 69]. Second, memory improvements are easy to confound with better prompting or tooling; without ablations that hold the scaffold fixed, gains should be treated as conditional on the full system design rather than as properties of a single memory module [57, 3].

Therefore, for evaluation, this suggests separating at least three measurable questions: (i) retrieval quality (does the memory return the right evidence), (ii) grounding behavior (does the agent actually use retrieved evidence rather than hallucinating), and (iii) robustness (how retrieval behaves under noise and adversarial inputs). Work that reports compression ratios or structured representations (e.g., 79.8% codebase condensation) provides one handle on (i), but it does not by itself guarantee grounded decisions in an interactive loop [55, 3]. Meanwhile, security taxonomies that include retrieval injection make clear that robustness requires explicit threat models and end-to-end defenses, not just better embeddings [69, 50]. Without protocols that measure these pieces separately, memory improvements can remain ambiguous and limited to the specific benchmark harness.

In addition, memory systems should be stress-tested under adversarial and worst-case conditions, not only under average-case workloads. Security benchmarks that explicitly include retrieval injection and tool-description injection illustrate that a memory store can become an attack surface that steers the loop, even when the base model is unchanged [69]. Defensive systems that claim strong robustness benefits are most convincing when paired with attack-aware evaluation protocols and when they report which parts of the memory and toolchain are trusted versus sanitized; otherwise, robustness can remain limited to the specific harness and threat model used in the paper [50, 17].

Finally, memory systems should be compared as policies, not as components. A retrieval-heavy design can improve factuality and reduce repeated tool calls, whereas a memory-light design can be easier to reset and less vulnerable to contamination across episodes. The right choice depends on the protocol: whether memory persists across tasks, whether it is user-specific, and whether the evaluation penalizes privacy leakage or unsafe reuse of prior context.

5 Learning, Adaptation & Coordination

Agent behavior is not fixed after deployment: many systems incorporate feedback, critique, or self-revision loops that change how decisions are made across episodes. Such adaptation can improve task completion, but it also raises questions about stability and what constitutes a fair comparison when policies continue to change during evaluation [51, 39].

Coordination compounds these issues by distributing state and decision-making across multiple interacting agents. Communication protocols, role assignments, and aggregation rules can dominate outcomes, and controlled studies suggest that multi-agent advantages are highly sensitive to interaction structure and task choice [60, 6, 42]. This chapter connects adaptation mechanisms and multi-agent coordination under a shared lens: how feedback and interaction policies reshape the agent loop under a fixed protocol.

5.1 Self-improvement and adaptation

Adaptation asks a different question than static agent design: instead of choosing a fixed loop once, how can the system improve its behavior as it collects feedback, observes failures, or revises its own outputs? This shift makes evaluation protocol central, because "better" can mean higher task success, lower cost, greater robustness, or safer behavior, and these objectives can conflict under realistic constraints. The key point is that adaptation mechanisms shape the frontier between success, cost, and robustness, so improvement claims are only comparable when they are tied to explicit evaluation settings and reporting conventions [76, 10].

A common approach is self-generated training or self-challenging loops, where the agent creates its own difficult cases and learns from them. On multi-turn tool-use benchmarks such as M3ToolEval and TauBench, the Self-Challenging framework reports more than a two-fold improvement for Llama-3.1-8B-Instruct using only self-generated training data [76]. This result highlights a concrete trade-off: self-generated feedback can be scalable and cheap, whereas it can also drift toward idiosyncratic errors unless the evaluation protocol checks generalization beyond the agent's own generated distribution.

A contrasting approach adapts by changing data and supervision rather than the loop policy online. Zhou et al. report that optimized datasets can yield an average improvement of 12% with larger gains on specific metrics (e.g., a 40% improvement in Fermi) as measured by benchmarks such as MT-bench, Vicuna Bench, and the WizardLM testset [74]. Compared with online self-revision, dataset-centric adaptation can stabilize training signals and make improvements more reproducible, but it can also make it harder to attribute gains to "agentic" behavior unless the evaluation includes interactive tasks and tool-use protocols.

Efficiency-oriented adaptation targets the success–cost frontier directly. EvoRoute reports that, when integrated into existing agentic systems, routing choices can sustain or enhance performance on benchmarks such as GAIA and BrowseComp+ while reducing execution cost by up to 80% and latency by over 70% [70]. This illustrates a practical evaluation anchor: for adaptive systems, improvements should be reported together with budgets and latency, because a change in adaptation policy can shift where an agent sits on the frontier even if raw success looks similar.

However, adaptation is especially prone to protocol artifacts. Anytool revisits a prior evaluation protocol and identifies a limitation that leads to an artificially high pass rate, showing how self-improvement claims can be inflated when metrics are too forgiving or when the harness fails to stress the right failure modes [10]. Related analyses of "fault lines" in agent evaluation emphasize that feedback loops can overfit to loopholes in reward definitions or test harnesses, so robustness claims should be read as conditional on a clearly stated protocol [41, 73].

Two limitations are worth keeping explicit. First, self-improvement loops can trade off stability for performance: more aggressive self-revision can amplify errors or reward hacking when feedback signals are misaligned, which makes careful reporting and ablations essential [41, 76]. Second, adaptation results often rely on domain-specific benchmarks (from interactive web tasks to scientific simulation), so transfer to new environments can be limited unless evaluation anchors are diversified and compared against strong non-adaptive baselines such as ReAct-style controllers [30, 63].

A final perspective comes from domain-specific "autonomous scientist" style agents, where adaptation is framed as iterative experimentation and correction rather than as benchmark gaming. Li et al. report that an LLM agent can autonomously perform tensor network simulations of quantum many-body systems with approximately 90% success across representative tasks, illustrating a regime where self-correction and tool-driven iteration are central to progress [30]. At the same time, such results are often tightly coupled to the toolchain and task distribution, so transfer to other interactive settings can be limited unless evaluation protocols stress failures and distribution shift in addition to average success [76, 41]. This reinforces the broader lesson: adaptation is powerful, but it is only interpretable when tied to explicit evaluation settings and when improvements are decomposed into cost, robustness, and failure-mode changes rather than reported as a single headline gain [70, 10].

For survey-level comparisons, it is also useful to separate "within-episode" adaptation (revision during a single task) from "across-episode" adaptation (learning from prior tasks). These regimes can have different failure modes: within-episode revision can amplify a bad assumption, while across-episode learning can overfit to benchmark quirks if feedback is not diverse. Evaluations that report learning curves or improvement trajectories over time make these dynamics visible, whereas one-shot scores can be misleading when adaptation continues during testing [76, 63].

For interpretation, the most important contrast is between adaptation that changes the policy online and adaptation that changes the data or supervision offline. Online revision can respond to task-specific feedback but may compound a wrong assumption, whereas offline optimization can be more stable but may blur whether gains come from better agents or better training distributions. For synthesis, it is therefore useful to treat "self-improvement" as a protocol-bound claim: improvements must be tied to when learning happens, what feedback is available, and how evaluation prevents leakage from training into test.

Adaptation changes how an agent updates behavior from feedback, and coordination extends this dynamic to interacting agents whose communication protocols reshape the loop.

5.2 Multi-agent coordination

Coordination turns an agent into a team. Once multiple agents interact, the "protocol" is no longer just an evaluation detail—it *is* the policy. As a result, turn-taking rules, message visibility, role specialization, and aggregation logic determine what information flows and what failures are possible. This makes multi-agent results particularly sensitive to interaction design, and it motivates protocol-aware synthesis rather than headline comparisons across disparate setups [60, 8].

A baseline contrast is between loosely coupled coordination (agents independently propose and a router selects) and tightly coupled coordination (agents iteratively negotiate, critique, or verify). Controlled analyses of debate-style settings suggest that the benefits of multi-agent interaction can vary widely with task structure and evaluation choices, cautioning against treating "more agents" as a universally reliable lever [60, 53]. In practice, coordination must be evaluated with the same care as planning: what constitutes agreement, when is communication allowed, and how are failures attributed to individual roles.

Training frameworks have begun to treat multi-turn interaction as a first-class object. SkyRL-Agent is presented as a framework for efficient, multi-turn, long-horizon agent training and evaluation, and results derived from it include a software engineering agent (SA-SWE-32B) trained from Qwen3-32B with a reported 24.4% Pass@1 baseline before reinforcement learning [2]. This line of work underscores an evaluation anchor for coordination: interaction policies change learning signals, so reported improvements should be paired with clear descriptions of turn structure, tool access, and reward definitions.

Applied settings provide another perspective on coordination: the goal is often not maximal accuracy on a single task, but throughput and reliability over many jobs. Continuum reports improved average job completion times on real-world agentic workloads such as SWE-Bench and BFCL with Llama-3.1 8B/70B, with improvements that scale as the number of turns increases [28]. In cybersecurity-style workloads, CRAKEN reports solving 25–30% more MITRE ATT&CK techniques than prior work via knowledge-based execution, illustrating how coordination and tool use can change practical coverage even when tasks are heterogeneous [48].

Coordination is also mediated by shared state, which links it back to memory design. MemTool evaluates different interaction "modes" across 13+ LLMs over 100 consecutive user interactions, tracking both task completion and tool removal ratios as a proxy for short-term memory efficiency under a fixed benchmark protocol [37]. This suggests a concrete comparison axis:

whether coordination policies rely on shared persistent memory, per-agent private memories, or explicit message passing can change both cost and failure modes, and these choices should be reported alongside performance.

Two limitations shape how to read multi-agent evidence. First, attribution is hard: when a team succeeds, it can be unclear whether gains come from better reasoning, better division of labor, or simply more total compute and retries, so ablations that hold budget fixed are essential [60, 28]. Second, many systems are evaluated under narrow interaction protocols; transfer to new coordination settings can be limited unless protocols and incentives are explicitly modeled and stress-tested, especially when tools and environments are adversarial or noisy [33, 8].

From an evaluation standpoint, the key confounder is budget. Coordination policies often increase the total amount of computation and the number of tool calls, so improvements should be compared under matched constraints whenever possible. This is visible in practice-oriented settings: job completion time gains that scale with turn count can reflect better coordination, but they can also reflect simply doing more work per task if budgets are not normalized [28, 2]. Similarly, coverage improvements on heterogeneous cybersecurity workloads (e.g., solving 25–30% more ATT&CK techniques) are meaningful only when the interaction protocol and tool permissions are explicitly stated, because coordination can change the effective action space [48, 8]. Where these details are missing, multi-agent claims are best read as evidence that interaction policies matter, not as definitive rankings of which coordination scheme is superior [60, 53].

A further complication is that coordination protocols embed implicit assumptions about trust and information flow. Whether agents can see each other's tool outputs, whether messages are summarized or raw, and whether roles are fixed or adaptive can all change the effective state space of the system. Studies that explicitly control interaction structure make it easier to attribute gains to coordination rather than to incidental information sharing; when these controls are absent, coordination evidence is limited and comparisons across papers become brittle [60, 33].

Coordination design also admits multiple "styles" that are easy to conflate. A decentralized team that gathers parallel proposals can be efficient, whereas a deliberative team that iteratively critiques and verifies can be slower but more robust on long-horizon tasks. Because these styles imply different communication costs and different error-correction mechanisms, evaluation protocols should state message budgets, visibility, and termination criteria to make multi-agent comparisons interpretable.

6 Evaluation & Risks

Evaluation is the binding constraint that makes the rest of the taxonomy meaningful: without matched protocols, it is unclear whether two agents differ in decision quality, tool access, budget, or interface cleanliness. Recent benchmarks emphasize this by varying API realism and budget assumptions, showing that protocol choices can dominate reported performance and that reporting conventions are part of reproducible progress [39, 22, 34].

Those same protocols also define risk surfaces. Tool use introduces new attack channels and monitoring requirements, and security-focused benchmarks operationalize failures such as tool-chain jailbreaks, software-task exploitation, and privacy leakage as end-to-end system properties [69, 29, 26, 25, 7]. The chapter therefore first clarifies what should be measured and reported for interpretable agent comparisons, then synthesizes how threat models and governance constraints change what it means to deploy agents safely.

6.1 Benchmarks and evaluation protocols

A central tension is that the tasks we want agents to solve are open-ended and interactive, whereas the comparisons we want to make require controlled, repeatable protocols. When benchmarks vary tool access, budgets, or logging assumptions, they can change not only measured performance but also which failure modes are even observable. As a result, the most useful evaluations are those that specify protocols as carefully as they specify tasks [39, 22].

Recent survey work on agent evaluation has begun to formalize what "protocol-aware" evaluation should mean. Mohammadi et al. propose a two-dimensional taxonomy that organizes evaluation by objectives (e.g., behavior, capability, reliability, safety) and by evaluation setups, providing a vocabulary for comparing papers that would otherwise look incomparable due to different harnesses and assumptions [39]. This taxonomy is valuable because it reframes benchmarks as design choices: a suite that measures step-by-step correctness under strict budgets answers a different question than one that measures end-task completion under permissive retries.

Realism-oriented benchmarks further highlight that API complexity and tool ecosystems are part of the measurement. Beyond Perfect APIs evaluates agents under real-world API complexity and demonstrates that assumptions about tool noise, schema strictness, and permissions can dominate outcomes [22]. Complementary work proposes more explicit "agent specifications" and reporting conventions that treat environment artifacts and contracts as part of the evaluated object, aiming to reduce ambiguity when results are reproduced or extended [58, 46].

A parallel thread focuses on evaluation design guidelines within specific domains. Chen et al. propose an evidence-based guideline for LLM-based robotic process automation evaluation by systematically reviewing 1,676 papers, arguing for more consistent evaluation design choices and reporting across studies [4]. Domain surveys such as LLM agents in law similarly emphasize that evaluation methodology and task selection must match the deployed setting, because "agent success" depends on the interaction protocol as much as on model capability [35].

Security and privacy evaluations underscore that protocol definition is inseparable from threat modeling. Fu et al. introduce RAS-Eval, which evaluates agent vulnerabilities across diverse scenarios and highlights how attack tasks and tool formats influence measured robustness [11]. Related work on contextual privacy auditing similarly shows that what is considered "safe" depends on adversarial assumptions and logging policies, making threat models a first-class part of benchmark design [7, 11].

Two limitations recur across current benchmarks. First, many suites still leave important protocol degrees of freedom unspecified (budget caps, retry policies, tool catalog versions), making cross-paper comparisons limited even when benchmark names match [39, 58]. Second, benchmark coverage can be narrow relative to the space of real failures, motivating meta-evaluation efforts (e.g., sentinel-style monitoring benchmarks and protocol taxonomies) that treat coverage and reporting as part of the evaluation target [68, 19].

One concrete path forward is to treat benchmarks as bundles of artifacts: tool catalogs, permission models, logging policies, budget definitions, and scoring scripts. When these are shipped and versioned, later work can replay protocol choices and make controlled changes, which turns "evaluation drift" into an explicit experimental variable rather than an accidental confound [58, 22]. This perspective aligns with broader evaluation-design guidance in applied domains, where consistency and traceability are prerequisites for cumulative progress [4, 39]. Without artifact-level specificity, benchmark results can remain limited to one-off demonstrations and resist the kind of synthesis that surveys aim to provide.

From this standpoint, a benchmark score is only the final line of a protocol definition. The underlying artifacts—tool catalogs, permission policies, budget caps, retry rules, and logging schemas—determine what behavior is even measurable and which failures are visible to the evaluator. Protocol templates and checklists help because they force these degrees of freedom into the paper, making later replication and ablation possible [39, 58]. Realism-oriented suites that vary API complexity make the same point empirically: once tool noise and schema constraints are introduced, small protocol changes can dominate outcomes [22]. In applied domains, evaluation-design guidelines similarly stress that "what to measure" must be paired with "how to measure it" in a way that can be re-run, otherwise evidence remains limited to narrative claims [4, 35].

In other words, evaluation choices create a spectrum from realism to control. Benchmarks that tightly fix tools, budgets, and logging make causal attribution easier, whereas more open-world setups better approximate deployment but make it harder to isolate what changed. Good survey synthesis depends on recognizing where each benchmark sits on this spectrum and avoiding head-to-head comparisons across incompatible protocols.

Protocol design enables interpretable evaluation, but threat models and governance assumptions are required to make safety and security claims meaningful in deployed agents.

6.2 Safety, security, and governance

Safety and security failures in agentic systems are rarely model-only. Because agents act through tools and environments, the relevant threat model includes the entire loop: tool descriptions, tool outputs, memory stores, and the policies that decide when to execute actions. This shifts governance from "prompt hygiene" to system design questions such as permissioning, observability, and end-to-end monitoring under explicit adversarial assumptions. The key point is that governance must be evaluated at the loop level, because protocol and interface choices determine both the attack surface and the meaning of robustness claims [69, 11].

A concrete step toward loop-level threat modeling is to name attack classes that are specific to tool use. MSB (an MCP security benchmark) contributes a taxonomy of 12 attacks, including name-collision, preference manipulation, prompt injections embedded in tool descriptions, out-of-scope parameter requests, user-impersonating responses, false-error escalation, tool-transfer, and retrieval injection [69]. The same work evaluates nine popular agents across 10 domains and 400+ tools, producing 2,000 attack instances, illustrating that "tool access" is not a benign assumption but a primary driver of security outcomes [69].

Benchmarking work also quantifies how attacks change end-task behavior. RAS-Eval comprises 80 test cases and 3,802 attack tasks mapped to 11 CWE categories, implemented across multiple tool formats (JSON, LangGraph, MCP), and reports that attacks reduce agent task completion rates by 36.78% on average with an 85.65% attack success rate in academic settings [11]. These numbers are evaluation anchors: robustness claims should be interpreted together with the attack surface and tool format, because defenses that work under one interface may fail under another.

Defense-oriented systems emphasize that robustness is an end-to-end property. Progent reports reducing attack success rates to 0% across agent use cases while preserving agent utility and speed, suggesting that effective mitigations must intervene across planning, invocation, and response-handling stages rather than treating the model as the only control point [50, 69]. Related lines of work explore enforcement and checking mechanisms around tool calls and execution, aiming to make security constraints explicit and auditable even when the model's internal reasoning is not [20, 1].

Safety is broader than adversarial security: even non-adversarial environments can contain hazards, specification gaps, or privacy risks. Benchmarks such as Agent-SafetyBench emphasize diverse unsafe interaction patterns and failure modes, encouraging evaluation beyond task success to include harm and policy compliance under interactive protocols [71, 67]. Adversarial reinforcement learning approaches similarly frame "agent safety" as a training problem under explicit threat models, reinforcing the idea that deployment governance must combine evaluation, monitoring, and learning signals [59, 23].

Two limitations remain. First, many papers still leave threat models implicit, making it unclear which inputs are adversarial and which actions are permitted; without explicit assumptions about permissions and logging, claims about safety can be limited to narrow settings [69, 11]. Second, robustness results often depend on tool and environment artifacts that are evolving in real deployments, so governance work needs to treat tool catalogs and their contracts as versioned objects and to bridge evaluation with operational monitoring, not as a one-off benchmark score [13, 20].

Governance mechanisms can therefore be viewed as "operational protocols" that complement benchmarks. Permissioning and audit logging constrain what actions are executable and make post-hoc analysis possible, while automated checking and enforcement aim to prevent unsafe tool chains before they execute [20, 1]. Bridging evaluation with deployment monitoring is especially important for agents, because the tool ecosystem and its failure modes evolve over time; a benchmark score without ongoing monitoring can give a false sense of security once the environment shifts [13, 67]. In this sense, governance is not an optional appendix but a continuation of the evaluation protocol under real constraints, and current evidence remains limited whenever papers do not specify permissions, logging, and response-handling policies alongside model and benchmark details [23, 69].

There is also an inherent security–utility trade-off: stronger defenses can restrict tool access or add verification steps that increase latency and cost. The most actionable papers make this trade-off measurable by pairing attack success metrics with task completion and runtime, rather than claiming "secure" behavior in the abstract [50, 11]. When defenses are learned (e.g., via adversarial training), the evaluation must also clarify the adversary model and whether robustness generalizes beyond the attack distribution used during training; otherwise, safety evidence remains limited to the tested regime [59, 69].

A final contrast is between defenses that harden the interface and defenses that harden the policy. Interface hardening focuses on sanitization, permissions, and execution guards, whereas policy hardening aims to train or steer the agent to avoid unsafe actions in the first place. In practice, the strongest evidence comes from systems that combine both and report how safety interventions change not only attack success but also task completion, cost, and monitoring load.

7 Discussion

A first cross-cutting lesson is that "agent capability" is routinely confounded with interface design. When benchmarks increase API realism—through noisy tools, partial schemas, or complex permissioning—failures shift from reasoning errors toward interface failures such as routing mistakes, schema mismatch, and brittle recovery logic [22, 36, 61]. This suggests a practical reporting norm: treat the tool catalog and its contracts as versioned artifacts, because otherwise improvements can reflect a cleaner interface rather than a better policy.

A second lesson is that cost and budget assumptions are not a nuisance variable but a primary axis of comparison. Cost-aware studies show that the same base model can traverse a steep success–cost frontier depending on how planners search, retry, and verify [34, 40]. For survey-level synthesis, this implies that performance claims should be interpreted as protocol-scoped trade-offs, and that benchmarks should report cost models and budgets with the same care as task definitions.

A third lesson concerns coordination. Multi-agent settings amplify protocol sensitivity because communication, role specialization, and aggregation choices create new degrees of freedom that change both outcomes and failure modes [60, 6, 42]. As a result, "multi-agent gains" are particularly hard to transfer across papers unless interaction protocols (turn structure, visibility, incentives) and evaluation constraints are fixed.

Finally, risk surfaces in agentic systems are inseparable from tool use. Security benchmarks and monitoring-oriented evaluations show that prompt/tool injection, tool chaining, and privacy leakage arise at the system boundary, and defenses must be evaluated end-to-end rather than as model-only guardrails [69, 26, 25, 29, 7]. A survey lens that jointly tracks protocols and threat models can therefore serve as an “evaluation checklist”: if a paper omits explicit tool access, logging, and adversarial assumptions, its claims should be treated as narrower than their headline numbers suggest.

8 Conclusion

LLM agents should be understood as closed-loop systems: their behavior is jointly shaped by the model, the action space exposed through interfaces, and the protocol under which they are evaluated. This view helps reconcile why results often fail to transfer across benchmarks and deployments: changes in tool access, budgets, logging, or threat models can change what success means, even when the base model is unchanged [63, 22, 39].

Across interfaces, planning and memory, adaptation, and coordination, the most reusable comparisons are those that make protocol assumptions explicit and tie claims to evaluation anchors. Looking forward, the field will benefit from (i) versioned tool catalogs and realistic test environments, (ii) budget-aware reporting that exposes success–cost trade-offs, and (iii) end-to-end security and monitoring benchmarks that operationalize agent-specific risks [61, 34, 69, 25].

References

[1] Vamshi Krishna Bonagiri, Ponnurangam Kumaraguru, Khanh Nguyen, and Benjamin Plaut. Check yourself before you wreck yourself: Selectively quitting improves llm agent safety. *arXiv preprint arXiv:2510.16492v2*, 2025. URL http://arxiv.org/abs/2510.16492v2.

[2] Shiyi Cao, Dacheng Li, Fangzhou Zhao, Shuo Yuan, Sumanth R. Hegde, Connor Chen, Charlie Ruan, Tyler Griggs, Shu Liu, Eric Tang, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Skyrl-agent: Efficient rl training for multi-turn llm agent. *arXiv preprint arXiv:2511.16108v1*, 2025. URL http://arxiv.org/abs/2511.16108v1.

[3] Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Grounded test-time adaptation for llm agents. *arXiv preprint arXiv:2511.04847v3*, 2025. URL http://arxiv.org/abs/2511.04847v3.

[4] Chaoran Chen, Bingsheng Yao, Ruishi Zou, Wenyue Hua, Weimin Lyu, Yanfang Ye, Toby Jia-Jun Li, and Dakuo Wang. Towards a design guideline for rpa evaluation: A survey of large language model-based role-playing agents. *arXiv preprint arXiv:2502.13012v3*, 2025. URL http://arxiv.org/abs/2502.13012v3.

[5] Jae-Woo Choi, Hyungmin Kim, Hyobin Ong, Minsu Jang, Dohyung Kim, Jaehong Kim, and Youngwoo Yoon. Reactree: Hierarchical llm agent trees with control flow for long-horizon task planning. *arXiv preprint arXiv:2511.02424v1*, 2025. URL http://arxiv.org/abs/2511.02424v1.

[6] Yun-Shiuan Chuang, Ruixuan Tu, Chengtao Dai, Smit Vasani, Binwei Yao, Michael Henry Tessler, Sijia Yang, Dhavan Shah, Robert Hawkins, Junjie Hu, and Timothy T. Rogers. Debate: A large-scale benchmark for role-playing llm agents in multi-agent, long-form debates. *arXiv preprint arXiv:2510.25110v1*, 2025. URL http://arxiv.org/abs/2510.25110v1.

[7] Saswat Das, Jameson Sandler, and Ferdinando Fioretto. Beyond jailbreaking: Auditing contextual privacy in llm agents. *arXiv preprint arXiv:2506.10171v3*, 2025. URL http://arxiv.org/abs/2506.10171v3.

[8] João Vitor de Carvalho Silva and Douglas G. Macharet. Can llm agents solve collaborative tasks? a study on urgency-aware planning and coordination. *arXiv preprint arXiv:2508.14635v1*, 2025. URL http://arxiv.org/abs/2508.14635v1.

[9] Jia-Kai Dong, I-Wei Huang, Chun-Tin Wu, and Yi-Tien Tsai. Etom: A five-level benchmark for evaluating tool orchestration within the mcp ecosystem. *arXiv preprint arXiv:2510.19423v2*, 2025. URL http://arxiv.org/abs/2510.19423v2.

[10] Yu Du, Fangyun Wei, and Hongyang Zhang. Anytool: Self-reflective, hierarchical agents for large-scale api calls. *arXiv preprint arXiv:2402.04253v1*, 2024. URL http://arxiv.org/abs/2402.04253v1.

[11] Yuchuan Fu, Xiaohan Yuan, and Dongxia Wang. Ras-eval: A comprehensive benchmark for security evaluation of llm agents in real-world environments. *arXiv preprint arXiv:2506.15253v1*, 2025. URL http://arxiv.org/abs/2506.15253v1.

[12] Stefano Fumero, Kai Huang, Matteo Boffa, Danilo Giordano, Marco Mellia, Zied Ben Houidi, and Dario Rossi. Cybersleuth: Autonomous blue-team llm agent for web attack forensics. *arXiv preprint arXiv:2508.20643v1*, 2025. URL http://arxiv.org/abs/2508.20643v1.

[13] Tarek Gasmi, Ramzi Guesmi, Ines Belhadj, and Jihene Bennaceur. Bridging ai and software security: A comparative vulnerability assessment of llm agent deployment paradigms. *arXiv preprint arXiv:2507.06323v1*, 2025. URL http://arxiv.org/abs/2507.06323v1.

[14] Amur Ghose, Andrew B. Kahng, Sayak Kundu, and Zhiang Wang. Orfs-agent: Tool-using agents for chip design optimization. *arXiv preprint arXiv:2506.08332v2*, 2025. URL http://arxiv.org/abs/2506.08332v2.

[15] Bingguang Hao, Zengzhuang Xu, Yuntao Wen, Xinyi Xu, Yang Liu, Tong Zhao, Maolin Wang, Long Chen, Dong Wang, Yicheng Chen, Cunyin Peng, Xiangyu Zhao, Chenyi Zhuang, and Ji Zhang. From failure to mastery: Generating hard samples for tool-use agents. *arXiv preprint arXiv:2601.01498v1*, 2026. URL http://arxiv.org/abs/2601.01498v1.

[16] Hanjiang Hu, Changliu Liu, Na Li, and Yebin Wang. Training task reasoning llm agents for multi-turn task planning via single-turn reinforcement learning. *arXiv preprint arXiv:2509.20616v2*, 2025. URL http://arxiv.org/abs/2509.20616v2.

[17] Jingyi Huang, Yuyi Yang, Mengmeng Ji, Charles Alba, Sheng Zhang, and Ruopeng An. Use of retrieval-augmented large language model agent for long-form covid-19 fact-checking. *arXiv preprint arXiv:2512.00007v1*, 2025. URL http://arxiv.org/abs/2512.00007v1.

[18] Zhenlan Ji, Daoyuan Wu, Pingchuan Ma, Zongjie Li, and Shuai Wang. Testing and understanding erroneous planning in llm agents through synthesized user inputs. *arXiv preprint arXiv:2404.17833v1*, 2024. URL http://arxiv.org/abs/2404.17833v1.

[19] Zimo Ji, Xunguang Wang, Zongjie Li, Pingchuan Ma, Yudong Gao, Daoyuan Wu, Xincheng Yan, Tian Tian, and Shuai Wang. Taxonomy, evaluation and exploitation of ipi-centric llm agent defense frameworks. *arXiv preprint arXiv:2511.15203v1*, 2025. URL http://arxiv.org/abs/2511.15203v1.

[20] Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing temporal constraints for llm agents. *arXiv preprint arXiv:2512.23738v1*, 2025. URL http://arxiv.org/abs/2512.23738v1.

[21] Arsham Gholamzadeh Khoee, Shuai Wang, Yinan Yu, Robert Feldt, and Dhasarathy Parthasarathy. Gatelens: A reasoning-enhanced llm agent for automotive software release analytics. *arXiv preprint arXiv:2503.21735v2*, 2025. URL http://arxiv.org/abs/2503.21735v2.

[22] Doyoung Kim, Zhiwei Ren, Jie Hao, Zhongkai Sun, Lichao Wang, Xiyao Ma, Zack Ye, Xu Han, Jun Yin, Heng Ji, Wei Shen, Xing Fan, Benjamin Yao, and Chenlei Guo. Beyond perfect apis: A comprehensive evaluation of llm agents under real-world api complexity. *arXiv preprint arXiv:2601.00268v1*, 2026. URL http://arxiv.org/abs/2601.00268v1.

[23] Hanna Kim, Minkyoo Song, Seung Ho Na, Seungwon Shin, and Kimin Lee. When llms go online: The emerging threat of web-enabled llms. *arXiv preprint arXiv:2410.14569v3*, 2024. URL http://arxiv.org/abs/2410.14569v3.

[24] Myung Ho Kim. Bridging symbolic control and neural reasoning in llm agents: The structured cognitive loop. *arXiv preprint arXiv:2511.17673v3*, 2025. URL http://arxiv.org/abs/2511.17673v3.

[25] Jonathan Kutasov, Yuqi Sun, Paul Colognese, Teun van der Weij, Linda Petrini, Chen Bo Calvin Zhang, John Hughes, Xiang Deng, Henry Sleight, Tyler Tracy, Buck Shlegeris, and Joe Benton. Shade-arena: Evaluating sabotage and monitoring in llm agents. *arXiv preprint arXiv:2506.15740v2*, 2025. URL http://arxiv.org/abs/2506.15740v2.

[26] Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. Sec-bench: Automated benchmarking of llm agents on real-world software security tasks. *arXiv preprint arXiv:2506.11791v2*, 2025. URL http://arxiv.org/abs/2506.11791v2.

[27] Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. Toolprmbench: Evaluating and advancing process reward models for tool-using agents. *arXiv preprint arXiv:2601.12294v1*, 2026. URL http://arxiv.org/abs/2601.12294v1.

[28] Hanchen Li, Qiuyang Mang, Runyuan He, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn llm agent scheduling with kv cache time-to-live. *arXiv preprint arXiv:2511.02230v2*, 2025. URL http://arxiv.org/abs/2511.02230v2.

[29] Jing-Jing Li, Jianfeng He, Chao Shang, Devang Kulshreshtha, Xun Xian, Yi Zhang, Hang Su, Sandesh Swamy, and Yanjun Qi. Stac: When innocent tools form dangerous chains to jailbreak llm agents. *arXiv preprint arXiv:2509.25624v1*, 2025. URL http://arxiv.org/abs/2509.25624v1.

[30] Weitang Li, Jiajun Ren, Lixue Cheng, and Cunxi Gong. Autonomous quantum simulation through large language model agents. *arXiv preprint arXiv:2601.10194v1*, 2026. URL http://arxiv.org/abs/2601.10194v1.

[31] Yu Li, Lehui Li, Zhihao Wu, Qingmin Liao, Jianye Hao, Kun Shao, Fengli Xu, and Yong Li. Agentswift: Efficient llm agent design via value-guided hierarchical search. *arXiv preprint arXiv:2506.06017v2*, 2025. URL http://arxiv.org/abs/2506.06017v2.

[32] Zichuan Li, Jian Cui, Xiaojing Liao, and Luyi Xing. Les dissonances: Cross-tool harvesting and polluting in pool-of-tools empowered llm agents. *arXiv preprint arXiv:2504.03111v3*, 2025. URL http://arxiv.org/abs/2504.03111v3.

[33] Ilija Lichkovski, Alexander Müller, Mariam Ibrahim, and Tiwai Mhundwa. Eu-agent-bench: Measuring illegal behavior of llm agents under eu law. *arXiv preprint arXiv:2510.21524v1*, 2025. URL http://arxiv.org/abs/2510.21524v1.

[34] Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R. Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for llm tool-use agents. *arXiv preprint arXiv:2511.02734v1*, 2025. URL http://arxiv.org/abs/2511.02734v1.

[35] Shuang Liu, Ruijia Zhang, Ruoyun Ma, Yujia Deng, Lanyi Zhu, Jiayu Li, Zelong Li, Zhibin Shen, and Mengnan Du. Llm agents in law: Taxonomy, applications, and challenges. *arXiv preprint arXiv:2601.06216v1*, 2026. URL http://arxiv.org/abs/2601.06216v1.

[36] Wenrui Liu, Zixiang Liu, Elsie Dai, Wenhan Yu, Lei Yu, and Tong Yang. Mcpagent-bench: A real-world task benchmark for evaluating llm agent mcp tool use. *arXiv preprint arXiv:2512.24565v2*, 2025. URL http://arxiv.org/abs/2512.24565v2.

[37] Elias Lumer, Anmol Gulati, Vamse Kumar Subbiah, Pradeep Honaganahalli Basavaraju, and James A. Burke. Memtool: Optimizing short-term memory management for dynamic tool calling in llm agent multi-turn conversations. *arXiv preprint arXiv:2507.21428v1*, 2025. URL http://arxiv.org/abs/2507.21428v1.

[38] Junyu Luo, Weizhi Zhang, Ye Yuan, Yusheng Zhao, Junwei Yang, Yiyang Gu, Bohan Wu, Binqi Chen, Ziyue Qiao, Qingqing Long, Rongcheng Tu, Xiao Luo, Wei Ju, Zhiping Xiao, Yifan Wang, Meng Xiao, Chenwu Liu, Jingyang Yuan, Shichang Zhang, Yiqiao Jin, Fan Zhang, Xian Wu, Hanqing Zhao, Dacheng Tao, Philip S. Yu, and Ming Zhang. Large language model agent: A survey on methodology, applications and challenges. *arXiv preprint arXiv:2503.21460v1*, 2025. URL http://arxiv.org/abs/2503.21460v1.

[39] Mahmoud Mohammadi, Yipeng Li, Jane Lo, and Wendy Yip. Evaluation and benchmarking of llm agents: A survey. *arXiv preprint arXiv:2507.21504v1*, 2025. URL http://arxiv.org/abs/2507.21504v1.

[40] Nayantara Mudur, Hao Cui, Subhashini Venugopalan, Paul Raccuglia, Michael P. Brenner, and Peter Norgaard. Feabench: Evaluating language models on multiphysics reasoning ability. *arXiv preprint arXiv:2504.06260v1*, 2025. URL http://arxiv.org/abs/2504.06260v1.

[41] Vikram Nitin, Baishakhi Ray, and Roshanak Zilouchian Moghaddam. Faultline: Automated proof-of-vulnerability generation using llm agents. *arXiv preprint arXiv:2507.15241v1*, 2025. URL http://arxiv.org/abs/2507.15241v1.

[42] Charidimos Papadakis, Angeliki Dimitriou, Giorgos Filandrianos, Maria Lymperaiou, Konstantinos Thomas, and Giorgos Stamou. Atlas: Adaptive trading with llm agents through dynamic prompt optimization and multi-agent coordination. *arXiv preprint arXiv:2510.15949v2*, 2025. URL http://arxiv.org/abs/2510.15949v2.

[43] Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *arXiv preprint arXiv:2503.23037v3*, 2025. URL http://arxiv.org/abs/2503.23037v3.

[44] Anjana Sarkar and Soumyendu Sarkar. Survey of llm agent communication with mcp: A software design pattern centric review. *arXiv preprint arXiv:2506.05364v1*, 2025. URL http://arxiv.org/abs/2506.05364v1.

[45] 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:2302.04761v1*, 2023. URL http://arxiv.org/abs/2302.04761v1.

[46] Gyuhyeon Seo, Jungwoo Yang, Junseong Pyo, Nalim Kim, Jonggeun Lee, and Yohan Jo. Simuhome: A temporal- and environment-aware benchmark for smart home llm agents. *arXiv preprint arXiv:2509.24282v2*, 2025. URL http://arxiv.org/abs/2509.24282v2.

[47] Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. *arXiv preprint arXiv:2410.06153v3*, 2024. URL http://arxiv.org/abs/2410.06153v3.

[48] Minghao Shao, Haoran Xi, Nanda Rani, Meet Udeshi, Venkata Sai Charan Putrevu, Kimberly Milner, Brendan Dolan-Gavitt, Sandeep Kumar Shukla, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. Craken: Cybersecurity llm agent with knowledge-based execution. *arXiv preprint arXiv:2505.17107v1*, 2025. URL http://arxiv.org/abs/2505.17107v1.

[49] Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. Small llms are weak tool learners: A multi-llm agent. *arXiv preprint arXiv:2401.07324v3*, 2024. URL http://arxiv.org/abs/2401.07324v3.

[50] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents. *arXiv preprint arXiv:2504.11703v2*, 2025. URL http://arxiv.org/abs/2504.11703v2.

[51] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366v4*, 2023. URL http://arxiv.org/abs/2303.11366v4.

[52] Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Zhicheng Dou, and Ji-Rong Wen. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. *arXiv preprint arXiv:2601.05808v1*, 2026. URL http://arxiv.org/abs/2601.05808v1.

[53] Lu Sun, Shihan Fu, Bingsheng Yao, Yuxuan Lu, Wenbo Li, Hansu Gu, Jiri Gesi, Jing Huang, Chen Luo, and Dakuo Wang. Llm agent meets agentic ai: Can llm agents simulate customers to evaluate agentic-ai-based shopping assistants? *arXiv preprint arXiv:2509.21501v1*, 2025. URL http://arxiv.org/abs/2509.21501v1.

[54] Zirui Tang, Weizheng Wang, Zihang Zhou, Yang Jiao, Bangrui Xu, Boyu Niu, Dayou Zhou, Xuanhe Zhou, Guoliang Li, Yeye He, Wei Zhou, Yitong Song, Cheng Tan, Xue Yang, Chunwei Liu, Bin Wang, Conghui He, Xiaoyang Wang, and Fan Wu. Llm/agent-as-data-analyst: A survey. *arXiv preprint arXiv:2509.23988v3*, 2025. URL http://arxiv.org/abs/2509.23988v3.

[55] Vali Tawosi, Salwa Alamir, Xiaomo Liu, and Manuela Veloso. Meta-rag on large codebases using code summarization. *arXiv preprint arXiv:2508.02611v1*, 2025. URL http://arxiv.org/abs/2508.02611v1.

[56] Minh-Hao Van, Prateek Verma, Chen Zhao, and Xintao Wu. A survey of ai for materials science: Foundation models, llm agents, datasets, and tools. *arXiv preprint arXiv:2506.20743v1*, 2025. URL http://arxiv.org/abs/2506.20743v1.

[57] Nikhil Verma. Active context compression: Autonomous memory management in llm agents. *arXiv preprint arXiv:2601.07190v1*, 2026. URL http://arxiv.org/abs/2601.07190v1.

[58] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666v3*, 2025. URL http://arxiv.org/abs/2503.18666v3.

[59] Zizhao Wang, Dingcheng Li, Vaishakh Keshava, Phillip Wallis, Ananth Balashankar, Peter Stone, and Lukas Rutishauser. Adversarial reinforcement learning for large language model agent safety. *arXiv preprint arXiv:2510.05442v1*, 2025. URL http://arxiv.org/abs/2510.05442v1.

[60] Haolun Wu, Zhenkun Li, and Lingyao Li. Can llm agents really debate? a controlled study of multi-agent debate in logical reasoning. *arXiv preprint arXiv:2511.07784v1*, 2025. URL http://arxiv.org/abs/2511.07784v1.

[61] Ziqiao Xi, Shuang Liang, Qi Liu, Jiaqing Zhang, Letian Peng, Fang Nan, Meshal Nayim, Tianhui Zhang, Rishika Mundada, Lianhui Qin, Biwei Huang, and Kun Zhou. Toolgym: an open-world tool-using environment for scalable agent testing and data curation. *arXiv preprint arXiv:2601.06328v1*, 2026. URL http://arxiv.org/abs/2601.06328v1.

[62] Weihao Xuan, Qingcheng Zeng, Heli Qi, Yunze Xiao, Junjue Wang, and Naoto Yokoya. The confidence dichotomy: Analyzing and mitigating miscalibration in tool-use agents. *arXiv preprint arXiv:2601.07264v1*, 2026. URL http://arxiv.org/abs/2601.07264v1.

[63] 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:2210.03629v3*, 2022. URL http://arxiv.org/abs/2210.03629v3.

[64] 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:2305.10601v2*, 2023. URL http://arxiv.org/abs/2305.10601v2.

[65] Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. Safeagentbench: A benchmark for safe task planning of embodied llm agents. *arXiv preprint arXiv:2412.13178v5*, 2024. URL http://arxiv.org/abs/2412.13178v5.

[66] Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. *arXiv preprint arXiv:2601.01885v1*, 2026. URL http://arxiv.org/abs/2601.01885v1.

[67] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-judge: Benchmarking safety risk awareness for llm agents. *arXiv preprint arXiv:2401.10019v3*, 2024. URL http://arxiv.org/abs/2401.10019v3.

[68] Simon Sinong Zhan, Yao Liu, Philip Wang, Zinan Wang, Qineng Wang, Zhian Ruan, Xiangyu Shi, Xinyu Cao, Frank Yang, Kangrui Wang, Huajie Shao, Manling Li, and Qi Zhu. Sentinel: A multi-level formal framework for safety evaluation of llm-based embodied agents. *arXiv preprint arXiv:2510.12985v1*, 2025. URL http://arxiv.org/abs/2510.12985v1.

[69] Dongsen Zhang, Zekun Li, Xu Luo, Xuannan Liu, Peipei Li, and Wenjun Xu. Mcp security bench (msb): Benchmarking attacks against model context protocol in llm agents. *arXiv preprint arXiv:2510.15994v1*, 2025. URL http://arxiv.org/abs/2510.15994v1.

[70] Guibin Zhang, Haiyang Yu, Kaiming Yang, Bingli Wu, Fei Huang, Yongbin Li, and Shuicheng Yan. Evoroute: Experience-driven self-routing llm agent systems. *arXiv preprint arXiv:2601.02695v1*, 2026. URL http://arxiv.org/abs/2601.02695v1.

[71] Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470v2*, 2024. URL http://arxiv.org/abs/2412.14470v2.

[72] Haiteng Zhao, Junhao Shen, Yiming Zhang, Songyang Gao, Kuikun Liu, Tianyou Ma, Fan Zheng, Dahua Lin, Wenwei Zhang, and Kai Chen. Achieving olympia-level geometry large language model agent via complexity boosting reinforcement learning. *arXiv preprint arXiv:2512.10534v2*, 2025. URL http://arxiv.org/abs/2512.10534v2.

[73] Yusheng Zheng, Yanpeng Hu, Wei Zhang, and Andi Quinn. Towards agentic os: An llm agent framework for linux schedulers. *arXiv preprint arXiv:2509.01245v4*, 2025. URL http://arxiv.org/abs/2509.01245v4.

[74] Hang Zhou, Yehui Tang, Haochen Qin, Yujie Yang, Renren Jin, Deyi Xiong, Kai Han, and Yunhe Wang. Star-agents: Automatic data optimization with llm agents for instruction tuning. *arXiv preprint arXiv:2411.14497v1*, 2024. URL http://arxiv.org/abs/2411.14497v1.

[75] Kaiwen Zhou, Ahmed Elgohary, A S M Iftekhar, and Amin Saied. Siraj: Diverse and efficient red-teaming for llm agents via distilled structured reasoning. *arXiv preprint arXiv:2510.26037v1*, 2025. URL http://arxiv.org/abs/2510.26037v1.

[76] Yifei Zhou, Sergey Levine, Jason Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. *arXiv preprint arXiv:2506.01716v1*, 2025. URL http://arxiv.org/abs/2506.01716v1.