LLM agents survey (tool-use, planning, memory, multi-agent)

January 18, 2026

Contents

Abstract

Tool-using LLM agents are best understood as closed-loop systems: they maintain an internal state, decide on actions, interact with tools or environments, and incorporate observations back into subsequent decisions. In practice, seemingly small choices about the loop boundary and action representation can dominate reliability, cost, and safety outcomes, which is why agent papers increasingly read like systems papers rather than pure model papers [85, 61].

This survey organizes the recent agent literature around four concrete levers: (i) interfaces and action spaces, (ii) planning and memory as reusable components, (iii) adaptation and multi-agent coordination, and (iv) evaluation and risk. Throughout, we emphasize interface contracts and evaluation protocols as the "execution layer" that makes claims comparable across works, and we highlight where benchmarks and threat models remain misaligned with deployment realities [54, 93].

1 Introduction

Agentic use of large language models has moved from a curiosity to a deployment pattern: systems increasingly combine an LLM with tools, memory, and execution scaffolding to solve tasks that are too long-horizon or too environment-dependent for one-shot generation. In this setting, the central question is no longer "can a model produce the right text," but "can a looped system make and verify decisions under constraints" [85, 61].

We use "agent" in the systems sense: a policy that observes state, chooses actions, and receives feedback through an environment or tool interface. This boundary is deliberately narrower than embodied robotics or classical RL; our focus is on tool-using LLM agents that operate via APIs, external resources, and structured tasks. As a result, concepts like action space, interface contract, and cost model become the right primitives for comparison, often more informative than model size alone [16, 48].

Method note (evidence policy): this run is abstract-first, so we treat quantitative details as provisional unless a paper note records the full protocol. We still prefer protocol-aware comparisons—benchmarks, metrics, cost budgets, tool availability—because those are the conditions under which claims become reproducible and transferable [54, 18].

This survey contributes a paper-like map of the agent design space. We structure the literature around (i) loop boundaries and action spaces, (ii) tool interfaces and orchestration, (iii) planning and memory components, (iv) adaptation and multi-agent coordination, and (v) evaluation and risks. The goal is not to list papers, but to surface the tensions that determine engineering outcomes: reliability vs flexibility, deliberation vs cost, memory depth vs drift, and autonomy vs governance [93, 32].

2 Related Work

The closest neighbors to tool-using LLM agents include work on tool use for language models, retrieval-augmented generation, planning/reasoning prompting, and evaluation of interactive systems. Tool use papers show that models can learn to call external functions and APIs, but they often abstract away the end-to-end loop details that dominate reliability in deployed agents [61, 16].

Planning and reasoning methods provide the cognitive substrate for agents, yet conclusions depend heavily on protocol choices: budgets, tool access, reset policies, and what counts as success. Recent benchmark-driven studies illustrate that planning quality can look very different under cost-aware versus accuracy-only metrics, making evaluation design inseparable from algorithm design [47, 54].

Memory and retrieval systems overlap with agents because many "agent failures" are state failures: missing context, stale summaries, or ungrounded assumptions. However, agent settings

add additional constraints—tool calls, multi-step execution, and dynamic environments—that are not always captured by standard RAG evaluations [74, 69].

A growing set of surveys and meta-analyses aim to systematize agent research. We build on these efforts but adopt an explicitly decision-oriented lens: rather than organizing by buzzwords, we organize by interface contracts, component roles (planning/memory/adaptation), and protocol-level evaluation anchors that make trade-offs comparable [71, 59].

Finally, safety and security work is increasingly central in agent deployments. Threat models such as prompt injection and tool abuse are not merely "misuse cases"; they reflect systematic vulnerabilities created by open-ended tool interfaces and insufficient provenance tracking. Accordingly, we treat risk and governance as part of the core design space, not as an afterthought [93, 67].

3 Foundations & Interfaces

Foundations and interfaces set the execution contract for everything that follows: what counts as state, what counts as an action, and what feedback closes the loop. A practical tension is that more expressive action spaces make it easier to solve open-ended tasks, but they also make it harder to verify behavior and to enforce safety boundaries under realistic budgets [85, 16].

We therefore separate the "agent loop" choice (how decisions and observations are represented) from the "tool interface" choice (how actions are grounded in APIs and orchestrators). The two are intertwined: orchestration policies can effectively redefine the action space, and benchmarks often implicitly assume one loop structure, which can bias conclusions about what works [50, 48].

3.1 Agent loop and action spaces

A tool-using agent is defined as much by its loop boundary and action space as by its underlying model. A central tension is that natural-language actions make the loop flexible and easy to prototype, whereas more structured actions make behavior easier to audit and evaluate under a protocol with explicit budgets and stop conditions [85, 34]. This suggests that "agent capability" should be compared through interface-level decisions, not just through anecdotal task success.

At the system level, evaluation is realized in a range of implementations (e.g., Li et al. [39]; Ghose et al. [21]; Song et al. [68]; Wu et al. [75]; You et al. [89]; and Xu et al. [82]).

To ground this loop view, many systems adopt variants of the state→decide→act→observe abstraction, but differ in what they treat as state (raw observations vs summarized memory) and what they treat as an action (free-form text vs constrained tool calls). In 2022, ReAct made this distinction visible by interleaving reasoning traces with explicit actions, which helped clarify where planning ends and execution begins [85]. Moreover, later systems often reinterpret "action space" as an interface contract: what actions are allowed, how errors are handled, and how retries are counted in evaluation [50].

Action spaces also shape what benchmarks can measure. When actions are continuous or open-ended, evaluation tends to rely on end-to-end task completion metrics that blur failure causes; in contrast, discretized action spaces enable finer-grained metrics (e.g., subtask success, tool-call validity, and cost) but may under-represent real deployment variability. Tool-focused environments such as ToolGym (2026) make this trade-off explicit by constraining what actions are executable while still exposing long-horizon dependencies that stress planning [78].

Recent benchmark work highlights that even "agent loop" papers often embed evaluation assumptions. For example, AgentSwift reports evaluation across seven benchmarks spanning multiple domains, which helps separate loop design choices from narrow task artifacts [41]. Similarly, LocoBench (2025) and MCPAgentBench (2025) emphasize protocol design—reset policies, task suite composition, and scoring—so that differences in action representations can

be compared under a more stable metric definition [58, 50]. However, these benchmarks still differ in tool availability and cost modeling, so cross-benchmark comparisons remain limited.

Taken together, the most actionable comparisons focus on how loop boundary choices interact with evaluation constraints. Works that emphasize explicit routing or control decisions can look stronger under cost-aware metrics because they reduce wasted actions, whereas more free-form loops may appear stronger under unconstrained "success if eventually solved" scoring. Therefore, a protocol-aware synthesis should ask: what actions are allowed, what is penalized (latency, tool calls, tokens), and what verification is required before an episode is considered complete [97, 94].

A key limitation is that richer action spaces enlarge the error surface. In security-relevant settings, a mis-specified action representation can turn a recoverable planning error into an irreversible tool abuse event, and benchmark scores may not reflect the real risk trade-off. Evidence from domain-specific agent studies (e.g., cybersecurity investigation workflows) underscores that the same loop design can behave very differently when the environment is adversarial or partially observed, so future work should report threat-model assumptions alongside the action space definition [19, 17].

One way to make these comparisons less hand-wavy is to treat the loop boundary as an experimental variable. If a benchmark exposes the same environment but changes what counts as an action (free-form steps vs validated tool calls), then improvements can be attributed more cleanly to planning or to interface choice. AgentSwif(t) explicitly leans on breadth—seven benchmarks across heterogeneous domains—which is useful precisely because it pressures the action space to be robust across different observation and feedback modalities [41]. By contrast, specialized benches can go deeper on protocol detail: they can specify tool schemas, episode resets, and scoring for intermediate steps, but they risk overfitting conclusions to one style of loop [50, 58].

For practitioners, a lightweight checklist helps: (i) make the action space explicit (what is callable, with what parameters), (ii) log enough state to support post-mortems, and (iii) choose evaluation metrics that separate "did it solve the task" from "did it waste actions or take unsafe actions." This is where controlled environments such as ToolGym are valuable: they let authors report success, cost, and error-recovery behavior under comparable protocols instead of relying on narrative demos [78, 94]. However, even with better harnesses, action-space design remains limited by mismatched assumptions about tool reliability and environment adversariality, which is why domain-specific studies remain important for stress testing loop definitions [19, 17].

Concretely, surveys can normalize comparisons by asking each paper to spell out an "episode contract": the allowed tool schemas, retry/error semantics, stop conditions, and what is logged for post-mortems. Benchmarks such as MCPAgentBench and ToolGym make these contracts more explicit, which is precisely why they are useful for isolating the impact of action representations and loop boundaries [50, 78]. With this contract view, classic reasoning–action splits like ReAct can be evaluated less as a prompting trick and more as a design choice about traceability and auditability under protocol constraints [85, 34].

After Agent loop and action spaces, Tool interfaces and orchestration makes the bridge explicit via function calling, tool schema, routing; tool interface (function calling, schemas, protocols), tool selection / routing policy, setting up a cleaner A-vs-B comparison.

3.2 Tool interfaces and orchestration

Tool interfaces are the "ground truth boundary" for LLM agents: they determine what actions can actually change the world, what observability the agent has, and what constraints (permissions, rate limits, and cost budgets) are enforced. A central tension is that richer tool APIs reduce friction for general problem solving, whereas tighter interface contracts make failures easier to detect and safer to contain under realistic evaluation protocols [16, 48]. This motivates treating interface design as a first-class comparison axis rather than an implementation

detail.

Concrete implementations of tool interfaces appear in Cheng et al. [10]; Xuan et al. [83]; Li et al. [42]; Li et al. [37]; and Xian et al. [81].

At the interface level, systems differ in whether they expose a small, curated function set or a large, open-ended tool catalog. AutoTool-style approaches emphasize automated tool selection and routing to reduce manual engineering, but this increases the need for reliable tool descriptions and robust fallback behavior when the selected tool fails [31]. In contrast, AnyTool-like framing highlights generality—covering broad tool families—but pushes more responsibility onto orchestration policies to keep action sequences valid and interpretable [16].

Orchestration becomes especially visible when tools include memory substrates. MemTool reports evaluation across 13+ LLMs on the ScaleMCP benchmark, and it explicitly studies stability over 100 consecutive uses, which makes "interface drift" observable rather than anecdotal [52]. However, memory-as-a-tool also blurs the boundary between state and action: writing to memory is an action that can silently change future behavior, which complicates attribution when metrics improve or degrade [44].

Because interface choices dominate failure modes, tool-use evaluation needs protocols that surface tool-specific errors (invalid parameters, permission violations, hallucinated tool names) and account for cost. Evaluation-oriented studies emphasize that many reported improvements are not comparable unless the protocol specifies tool availability, allowed retries, and what counts as a successful tool call versus a successful end-to-end task [54, 14]. In practice, a benchmark that scores only final success can hide systematic tool misuse, whereas a protocol that scores tool validity may penalize exploratory behavior that is acceptable in some deployments.

Self-improvement loops interact with tool interfaces in a subtle way. When agents learn to challenge or critique themselves, they often change not just the content of their reasoning but also their tool-use policies (when to call a tool, when to verify, and when to stop). Therefore, the same interface can yield different trade-offs depending on whether the system emphasizes conservative verification or aggressive exploration under a fixed budget [100, 48]. In contrast, purely static tool policies may look strong on narrow benchmarks but remain brittle when tool distributions shift.

A key limitation is that the interface contract is rarely fully specified in papers: permissions, sandboxing, and failure recovery policies are often implicit. This makes it hard to transfer conclusions from one tool suite to another, even when the underlying model is similar. Future work should report interface-level conditions—tool schemas, constraints, and logging hooks—as part of the experimental protocol, so that orchestration decisions can be compared without relying on "demo feel" as evidence [54, 44].

Moreover, "tool interface" includes more than the list of functions: it includes schemas, validation, error semantics, and permissioning. ToolScope-style analyses emphasize that agents can appear competent while silently relying on underspecified contracts (e.g., tools that never fail, or tools that return perfectly formatted outputs), which can collapse when moved to real deployments [48]. In contrast, evaluation harnesses that explicitly score tool-call validity and parameter correctness surface failure modes early, even if they make headline success rates look worse [14, 54].

From a systems standpoint, a practical orchestration policy should specify at least three things: selection (how a tool is chosen), execution (how errors are handled and retried), and verification (what evidence is required before an action is accepted). AutoTool-style routing helps on selection, but without strong verification it can amplify tool misuse; AnyTool-style breadth helps coverage, but it increases the burden on execution-time safeguards [31, 16]. MemTool's repeated-use setting is a useful stress test here because it makes "policy drift" observable over long runs rather than only in single episodes [52, 44].

Finally, interface comparisons need observability, not just API lists. Without structured logging of tool arguments, tool outputs, and verification signals, interface failures collapse into

"model errors" in post-hoc analysis. Tool-focused evaluations argue for reporting tool-call traces and schema validation outcomes as part of the benchmark artifact, because they reveal whether a method improves selection, execution robustness, or verification discipline [14, 48]. Long-run stress tests are particularly informative here: they can expose interface drift, caching pathologies, and subtle permission mismatches that single-episode benchmarks miss [52, 54].

4 Core Components (Planning + Memory)

Planning and memory are the two reusable components that most directly determine whether an agent can act coherently over long horizons. Planning governs how the system allocates deliberation and chooses actions; memory governs what the system treats as state and how it stays grounded when context is long or noisy [85, 74].

The key comparison axes in this chapter are protocol-level: cost budgets, tool availability, and evaluation metrics. Without holding these fixed, it is easy to conflate "better planning" with "more expensive planning," or to attribute gains to memory when they are actually due to different task setups [47, 69].

4.1 Planning and reasoning loops

Planning and reasoning loops are the control layer that turns an agent from a reactive tool caller into a system that can allocate deliberation, recover from errors, and trade off cost against success. A central tension is that deeper deliberation can improve long-horizon task completion, whereas stricter budgets and cost models penalize unnecessary reasoning and tool calls [85, 47]. This implies that planning methods should be compared under protocols that make budgets and stopping rules explicit.

Concrete system realizations of planning/control loop are described in Silva et al. [13]; Hatalis et al. [24]; Lu et al. [51]; Huang et al. [29]; Hu et al. [28]; and Yang et al. [84].

One route is to improve planning via learning signals rather than only via prompting. For example, Hu et al. (2025) report that a 1.5B-parameter model trained with single-turn GRPO can outperform larger baselines up to 14B parameters on a complex task planning benchmark, with success rates reported around 70% for long-horizon planning [27]. Moreover, this kind of result is only meaningful when the benchmark specifies the task distribution and metric definition; otherwise, "better planning" can collapse into "different evaluation."

A second route is to refine deliberation and guidance at inference time. In security-oriented settings, Nakano et al. (2025) report that a self-guided reasoning tool completes only 13.5%, 16.5%, and 75.7% of subtasks in different configurations and requires 86.2%, 118.7%, and 205.9% more model queries, highlighting that planning quality and cost are often tightly coupled [55]. However, the interpretation depends on protocol details: what counts as a subtask, whether retries are allowed, and how tools are exposed.

Beyond raw planning performance, loop design decisions affect what kinds of reasoning are even possible. CostBench (2025) explicitly studies cost-aware planning and reports that agents may fail to identify cost-optimal solutions in static settings, which makes "budget awareness" a measurable axis rather than an afterthought [47]. In contrast, protocol-free evaluations can hide whether an agent is truly planning or simply sampling many candidates until something works.

Taken together, the most informative evidence comes from comparisons that hold protocols fixed while varying loop mechanisms. Under a unified ReAct-style framework, Seo et al. (2025) evaluate 16 agents and report distinct capability and limitation profiles across models, suggesting that "planning loop" is not a monolith but a family of control patterns whose behavior depends on the environment and metric [63, 85]. Therefore, survey synthesis should emphasize which

control-loop assumptions are shared (planner/executor separation, search strategy, verification) and which are artifacts of evaluation.

A key limitation is that planning loops open new attack and failure surfaces. Reasoning-style vulnerabilities and process-oriented attacks can manipulate how an agent allocates steps, which may induce premature errors even when content filters are strong [98]. This motivates treating verification, logging, and bounded reasoning policies as part of planning design, not just as safety add-ons, especially when systems rely on memory retrieval or novelty-seeking mechanisms that can amplify spurious plans [15, 35, 34].

Viewed through a protocol lens, planning methods differ in what they optimize and what they assume. Some loops assume that extra deliberation is essentially free, so they optimize for success rate; others explicitly trade success against cost or latency, which turns planning into a constrained optimization problem. CostBench makes this distinction concrete by pushing cost-aware planning into the benchmark definition, while training-based approaches report gains under their own task distributions and reward specifications [47, 27]. In contrast, prompt-centric planning claims can be hard to interpret unless the paper reports the same constraints (tool access, retry limits, budget) alongside the metric.

For readers deciding "what to build," the most useful comparison is not CoT vs ToT by name, but whether a system has (i) an explicit planner/executor split, (ii) a verification step that can halt bad plans early, and (iii) a budgeted control policy that prevents runaway deliberation. Simulation-style evaluations such as SimuHome help because they unify multiple agents under a shared ReAct-style loop and reveal how planning behavior changes with environment complexity [63, 85]. However, because planning interacts with novelty-seeking and memory retrieval, conclusions can drift when the loop includes additional components; this is one reason to demand ablations and protocol reporting when papers claim general planning improvements [35, 15, 26].

One pragmatic synthesis is to separate "planner capacity" from "planner discipline." Capacity asks whether the loop can search or decompose tasks effectively; discipline asks whether it can stop, verify, and stay within budget when search is tempting. Benchmarks that expose both success and resource traces (queries/tool calls/latency) make this distinction measurable and reduce the chance that a method wins by simply spending more [47, 55]. For future surveys and evaluations, reporting a small set of standardized controls (budget, retry policy, and verification trigger) would make cross-paper planning claims far easier to compare than yet another taxonomy of prompting names [63, 27].

Memory and retrieval (RAG) follows naturally by turning Planning and reasoning loops's framing into retrieval, index, write policy; memory type (episodic / semantic / scratchpad), retrieval source + index (docs / web / logs)-anchored evaluation questions.

4.2 Memory and retrieval (RAG)

Memory and retrieval determine what an agent treats as state over time, which is why many "agent failures" are memory failures: missing context, stale summaries, or ungrounded assumptions. A central tension is that richer memory improves grounding in multi-turn settings, whereas aggressive compression and retrieval heuristics can introduce drift that is hard to detect under coarse evaluation metrics [74, 85]. This suggests that memory mechanisms should be evaluated with protocols that stress long-horizon consistency rather than only single-turn accuracy.

At the system level, memory/retrieval is realized in a range of implementations (e.g., Yu et al. [90]; Xia et al. [79]; Zhang et al. [96]; and Ye et al. [87]).

A practical way to organize memory work is by the unit of storage and the unit of retrieval. Some systems store free-form text summaries; others store structured traces, graphs, or tool call records that can be queried more deterministically. Memory-box style systems make the "state representation" explicit by treating memory modules as composable components that

can be swapped and benchmarked across tasks [69]. In contrast, more implicit memory can look simpler to implement but becomes difficult to audit when failures occur.

Structured memories also enable different evaluation lenses. AriGraph (2024) uses a graph-based representation that can support targeted retrieval and relational reasoning, which can change the failure modes compared to purely textual retrieval [3]. Moreover, task-specific memory policies often hinge on what the environment exposes (tool logs, web pages, code execution traces), which is why "memory" in agents is not identical to classic RAG in static QA settings [86].

Recent comparative studies make the protocol issue concrete. Wei et al. (2025) report implementing over ten representative memory modules and evaluating them across 10 diverse multi-turn, goal-oriented tasks, which helps isolate what memory contributes under a shared metric definition [74]. However, even such studies can be sensitive to evaluation details: how success is scored, how long context is truncated, and whether external tools are available.

In contrast to hand-designed memory modules, meta-level approaches emphasize learning or adapting retrieval policies. These routes can reduce manual engineering, but they also risk overfitting to benchmark artifacts or amplifying spurious correlations when memory content is noisy [70, 1]. Therefore, when comparing memory systems, it is useful to separate (i) what is stored, (ii) how it is retrieved, and (iii) how retrieval decisions are verified.

A key limitation is that memory introduces new security and robustness concerns: poisoning, prompt injection via stored content, and subtle state drift that only manifests many steps later. Without explicit provenance and verification hooks, a memory system can accumulate errors that look like "planning mistakes" downstream. This motivates evaluation protocols that include stress tests for long-horizon consistency and adversarial memory content, rather than treating memory as a purely helpful add-on [15, 86].

A complementary viewpoint is to treat retrieval as an autonomous subsystem with its own objectives and failure modes. MemR^3 frames memory retrieval as an agent-like process, which highlights that retrieval is not only about finding relevant text but also about deciding what evidence is trustworthy and when to refresh state [15]. In contrast, graph-based representations such as AriGraph make state relational and queryable, which can reduce some hallucination modes but introduces its own brittleness when the graph construction is noisy or incomplete [3].

In practice, memory evaluation should go beyond average task success and measure stability: does the agent's state drift across repeated runs, does retrieval amplify spurious correlations, and does summarization silently erase constraints needed for safe tool use. Comparative evaluations across 10 multi-turn tasks and many memory modules are helpful because they expose which improvements are consistent and which are benchmark-specific [74]. Still, a limitation is that long-horizon memory failures often look like planning failures; future protocols should include explicit memory stress tests (poisoned entries, contradictory evidence, and partial observability) so that memory and planning contributions can be disentangled [69, 86].

A useful reporting standard is to make memory operations first-class in the trace: what was retrieved, why it was retrieved (query), and how it changed the next action. Without such traces, it is hard to tell whether improved success comes from better state modeling or from accidental leakage (e.g., storing answers in memory) under a lax protocol. Modular frameworks like MemBox make it easier to swap storage/retrieval policies under identical tasks, while graph-based representations make state relational and queryable; both directions strengthen the case for treating memory as an auditable subsystem rather than a "black box context" [69, 3]. Even in abstract-first evidence, surveys can insist on this traceability contract: memory should be inspectable and verifiable, not merely available [15, 74].

5 Learning, Adaptation & Coordination

Adaptation mechanisms and multi-agent coordination are often introduced as ways to reduce brittleness: agents can critique and improve themselves over time, or multiple agents can divide labor and cross-check outputs. However, these mechanisms also introduce new failure modes—reward hacking, instability, and coordination overhead—that only become visible under long-horizon evaluation protocols [100, 59].

This chapter frames the landscape through two questions: how agents change with experience (reflection, optimization, or learned controllers), and how they coordinate (roles, protocols, aggregation). In both cases, the most informative comparisons connect mechanism choices back to measurable outcomes under a stated benchmark or metric [99, 11].

5.1 Self-improvement and adaptation

Self-improvement mechanisms aim to make agents less brittle by letting them adapt their behavior based on feedback, critique, or learned optimization signals. A central tension is that adaptation can improve robustness on multi-turn tasks, whereas it can also induce drift and reward hacking that is invisible under short-horizon metrics [100, 71]. This motivates evaluating adaptation as a protocol-aware system change, not as a cosmetic prompt tweak.

Concrete system realizations of learning/feedback are described in Xia et al. [80]; Sarukkai et al. [60]; Chen et al. [8]; He et al. [25]; Schneider et al. [62]; and Wang et al. [73].

One common pattern is "self-challenging" or critique loops that modify the agent's plan or tool-use policy after failures. Zhou et al. (2025) evaluate a self-challenging framework on two multi-turn tool-use agent benchmarks (M3ToolEval and TauBench), emphasizing that improvements should be measured under fixed task suites and scoring rules rather than via cherry-picked examples [100]. Moreover, such loops often change when the agent decides to verify or to stop, which can alter cost and failure modes under a budgeted protocol.

Another pattern is to learn controllers that adapt actions based on environmental feedback. RL-style approaches for agents can formalize adaptation objectives, but they inherit classical pitfalls: proxy objectives, instability, and sensitivity to evaluation distribution shift. Archer-style training (2024) illustrates this route by explicitly optimizing agent behavior rather than relying purely on prompt engineering, which can improve repeatability when protocols are well specified [99, 97]. In contrast, purely prompt-based adaptation may appear strong in demos but be hard to reproduce.

Adaptation also interacts with the rest of the agent stack. Systems that evolve or route behaviors over time can shift which planning and memory components are actually used, which makes ablations essential for interpretation [77, 94]. Therefore, a survey synthesis should treat "adaptation" as a coordination problem across components: what changes, what remains fixed, and what evidence supports the claimed improvement.

Taken together, adaptation gains are most convincing when they come with protocol-level anchors: benchmark names, metrics, and budget conditions. It is also helpful when papers link adaptation back to a clear failure analysis (what went wrong before, what changes after), because otherwise improvements can be indistinguishable from overfitting to the evaluation harness [100, 56]. This suggests that adaptation research should report both performance and stability metrics across repeated runs.

A key limitation is that self-improvement can amplify errors if the feedback loop is misspecified. Even when adaptation improves average performance, it may increase tail risk (rare catastrophic failures) or degrade safety properties if verification is relaxed. This motivates coupling adaptation with memory and monitoring: tracking what changed, under what evidence, and how the agent's policy can be rolled back when anomalies are detected [74, 4, 85].

Beyond critique loops, "adaptation" increasingly includes explicit policy evolution: systems that route between behaviors or evolve prompts/programs based on observed failures. Evolver-

style approaches make this a first-class mechanism rather than an ad-hoc retry, but they also raise the bar for evaluation: the protocol must specify what feedback is available, how many iterations are allowed, and how overfitting is detected [77]. Likewise, route-selection or evolution frameworks can change which tools are used and which failure modes appear, so it is essential that papers report not just final success but also how the policy changed over time [94, 97].

For safety and reliability, adaptation needs guardrails. A practical mitigation is to treat adaptations as versioned changes with explicit rollback and to couple improvements with verification policies that prevent "improving" by relaxing checks. This is especially important when adaptation interacts with memory: a self-improving agent can learn to exploit a memory shortcut that passes a benchmark but fails under distribution shift [74, 56]. Therefore, even in abstract-first evidence, a cautious survey should frame adaptation claims as conditional on protocol details and stability evidence, not as unconditional progress [4, 100].

Methodologically, adaptation papers would benefit from borrowing evaluation idioms from software changes: report a baseline, a delta (what changed), and a regression envelope (what got worse). If a self-challenging loop increases success, it is also important to report whether it increases tool-call count, latency, or failure severity under the same budget, and whether gains persist across benchmark variants rather than only the training-like distribution [100, 56]. Evolutionary or routing-based adaptation can be framed as a controlled search over policies; without clear limits on iterations and explicit overfitting checks, it becomes hard to separate genuine robustness from benchmark-specific tuning [77, 94].

Even lightweight audits help: track which tools and verification steps are used before and after adaptation, and report whether "improvements" come from better decisions or from relaxed checks. This makes adaptation results easier to reuse across settings and reduces the chance that protocol drift is mistaken for capability gains [100, 97].

Multi-agent coordination follows naturally by turning Self-improvement and adaptation's framing into roles, communication, debate; communication protocol + role assignment, aggregation (vote / debate / referee)-anchored evaluation questions.

5.2 Multi-agent coordination

Multi-agent systems extend the agent loop by distributing cognition across roles—planning, execution, verification, or critique—rather than forcing a single policy to do everything. A central tension is that specialization can reduce individual brittleness, whereas coordination introduces overhead, communication failures, and the risk of correlated errors [59, 11]. This implies that coordination protocols should be evaluated as first-class design choices.

In practice, the literature operationalizes coordination across multiple implementations (e.g., Papadakis et al. [57]; Wu et al. [76]; Chang et al. [7]; Zhang et al. [95]; Hassouna et al. [23]; and Li et al. [43]).

A practical coordination axis is whether agents communicate in free-form language or through more structured messages. Free-form debate can surface contradictions and improve coverage, but it can also devolve into persuasion rather than verification if incentives are misaligned [11]. In contrast, more structured collaboration protocols can be easier to audit, but they may limit the diversity of hypotheses explored.

Representative systems illustrate different points in this trade-off space. Voyager (2023) emphasizes long-horizon behavior through iterative skill acquisition and environment interaction, which naturally benefits from role separation across planning and execution [72]. Smaller-agent studies (e.g., 2024) suggest that coordination quality is not purely a function of model scale; protocol design and task decomposition matter at least as much as raw capability [66]. Moreover, coordination often changes evaluation: what counts as success for a team, and how credit is assigned.

Evaluation of multi-agent systems is still maturing. Work on evaluating multi-agent reasoning and collaboration highlights that aggregation rules (majority vote, arbitration, or hier-

archical control) can change outcomes even when the underlying models are fixed, which makes "team performance" sensitive to protocol details [88, 91]. Therefore, multi-agent benchmarks should specify communication bandwidth, turn limits, and stopping criteria, not just final task correctness.

Taken together, coordination mechanisms are best compared by the failure modes they prevent or introduce. Debate-style systems can catch some factual errors, but they may still share blind spots if all agents rely on the same tools or retrieval sources; conversely, tool diversity can increase robustness but also increases orchestration complexity [59, 52]. This suggests that practical systems may need mixed protocols: structured checks for safety-critical steps and freer collaboration for exploration.

A key limitation is that coordination can create new incentive problems and attack surfaces (collusion, sybil behaviors, and adversarial messaging). RL-based coordination policies may mitigate some issues, but they also require careful evaluation under distribution shift and adversarial conditions [6, 99]. Future work should report coordination assumptions explicitly and include stress tests where communication is noisy, tools are unreliable, or some agents are compromised.

Coordination overhead is not a footnote; it is a measurable part of system behavior. Communication bandwidth, turn limits, and shared tool access can dominate both cost and accuracy, which is why coordination protocols should be reported alongside task metrics. Shared memory or shared tool substrates can help agents stay aligned, but they also increase coupling and can make correlated errors more likely if all agents depend on the same retrieved evidence [52, 59].

From an evaluation standpoint, the field still lacks "calibrated" multi-agent benchmarks: many papers report team-level success without specifying how disagreement is resolved or how failures are attributed. Work that explicitly evaluates collaboration and aggregation highlights that arbitration rules and verification roles can flip outcomes even under identical underlying models, which suggests that coordination should be tested under controlled protocol variations [88, 91]. This remains limited by the lack of standardized settings for adversarial messaging, noisy communication, and partial tool failures, so results should be read as conditional rather than universal [6, 11].

From a design perspective, many practical patterns can be expressed as separation of concerns: one agent proposes, another critiques, and a third arbitrates under a fixed policy. This makes coordination closer to an executable protocol than a vague "teamwork" metaphor, and it naturally suggests ablations: swap the arbiter policy, restrict communication, or remove shared memory to see which component drives gains [88, 91]. A remaining open question is how these protocols scale with tool-use: as agents share tool access and memory, correlated errors become more likely, so coordination should be paired with diversity mechanisms (independent retrieval, heterogeneous tools) and explicit budgets on communication to avoid winning by simply "talking more" [52, 59].

As a result, survey comparisons should treat communication policy (who speaks, how often, with what evidence) as an evaluation variable rather than a narrative detail. Protocol choices like turn limits, arbitration rules, and messaging constraints often matter as much as model choice once systems scale beyond two roles [88, 6].

6 Evaluation & Risks

Evaluation and risk are the shared constraints that determine whether an agent design is meaningful beyond a demo. Benchmarks define what is measured and what is ignored; security and governance define what cannot be ignored in deployment, even if it is inconvenient to measure [54, 93].

We focus on protocol-aware evaluation: task suites, cost models, and reproducibility controls, alongside threat models such as prompt injection, data exfiltration, and tool abuse. The

goal is to make the evaluation layer explicit enough that conclusions can transfer across systems and settings [18, 32].

6.1 Benchmarks and evaluation protocols

Benchmarks and evaluation protocols are the common language that makes agent claims comparable, yet they are also a major source of disagreement: the same agent can look strong under an unconstrained success metric and weak under a budgeted or safety-aware protocol. A central tension is that end-to-end benchmarks capture realistic failure compounding, whereas component-level evaluations make it easier to attribute improvements and to reproduce results [54, 18]. This motivates treating protocol design as part of the agent research contribution.

Concretely, prior work instantiates tool interfaces in several systems (e.g., Ji et al. [30]; Zhan et al. [92]; Dagan et al. [12]; Zhu et al. [101]; and Lidayan et al. [46]).

Benchmark suites for agents are increasingly diverse. Some focus on interactive task completion, others on tool-use correctness, and others on safety stress tests. AgentSquare (2024) exemplifies the "suite" approach by packaging multiple scenarios under a consistent harness, which can reduce the temptation to overfit to a single task type [65]. However, suites still vary in tool availability and reset policies, so comparisons across suites remain limited unless protocols are standardized.

Reliability-focused work emphasizes that metrics must reflect what matters in deployment: not just whether an answer is correct, but whether it is obtained efficiently and safely under constraints. This leads to protocol elements such as cost budgets, latency caps, and repeated-trial stability checks, which can change which methods look preferable [32, 33, 40]. Moreover, explicit protocol reporting enables reviewers to identify when gains come from harness differences rather than from method differences.

Security-aware evaluation further complicates the picture. Zhang et al. (2025) propose a taxonomy of 12 agent attacks, including categories such as prompt injection and manipulation routes, which provides a concrete scaffold for threat-model-aware benchmarking rather than ad-hoc red teaming [93]. Systems like Progent-style evaluations connect these threat models to agent use cases and show that safety conclusions depend on both interface exposure and monitoring, not just on model alignment [67].

Taken together, the evaluation layer should be understood as a set of design choices with their own failure modes: leakage, non-determinism, and missing cost models can all bias conclusions. Therefore, survey comparisons should prefer studies that report protocol details (budgets, tool access, scoring rules) and that include ablations separating model capability from harness effects [9, 54].

A key limitation is that many benchmarks remain partial proxies for deployment. They may omit adversarial environments, long-tail tool failures, or governance constraints, leading to optimistic performance estimates. This motivates two directions: (i) richer benchmark reporting standards (what was allowed and what was measured), and (ii) cross-benchmark calibration studies that quantify how conclusions change when protocols differ [18, 32].

A useful mental model is that an agent benchmark is a bundle of hidden assumptions: tool reliability, availability, budget, and what counts as evidence. When these are not reported, evaluation becomes a moving target, and seemingly "better" methods may simply be evaluated under easier conditions. This is why evaluation meta-work argues for reporting standards and for separating harness effects from model effects, especially when systems involve tool calls and multi-turn state [54, 18].

For security-aware evaluation, protocol specificity is even more critical. Attack taxonomies (e.g., 12 attack categories) provide coverage targets, but they only become meaningful when a benchmark specifies attacker capabilities, tool permissions, and monitoring hooks [93]. Therefore, the most actionable direction is to treat benchmark design as an engineering artifact:

publish harnesses, specify randomness controls, and include cost models so that results can be reproduced and compared across labs [33, 32, 40].

Calibration is arguably the missing layer. When two benchmarks claim to measure similar abilities but differ in tool availability, reset rules, and scoring, headline results cannot be compared directly. Meta-evaluation work suggests that a small number of controlled protocol swaps (budget, tool set, verification rule) can quantify how brittle a method is to harness changes and can prevent leaderboard overfitting from being mistaken for progress [54, 18]. In this view, publishing harnesses and reporting randomness controls are not just reproducibility hygiene; they are the mechanism by which agent papers make claims that survive across labs [33, 40].

Another practical lesson is to report failure structure, not only success. For agents, a single score often conflates tool-call errors, planning failures, and safety violations. Protocol-aware evaluation encourages logging intermediate events (tool-call validity, verification outcomes, abort reasons) and reporting distributions across runs, which supports a more honest discussion of reliability and risk envelopes [32, 9]. For safety-aware suites, threat taxonomies become useful only when mapped to concrete attacker capabilities and monitoring hooks; otherwise "secure" can mean "not tested" [93, 67].

Rather than restarting, Safety, security, and governance carries forward the thread from Benchmarks and evaluation protocols and stresses it through threat model, prompt/tool injection, monitoring; threat model (prompt / tool injection, exfiltration), defense surface (policy, sandbox, monitoring).

6.2 Safety, security, and governance

Safety and security for tool-using agents are not separable from system design: the tool interface defines the attack surface, the logging surface, and the available mitigations. A central tension is that higher autonomy reduces human oversight costs, whereas it increases the risk of prompt injection, data exfiltration, and tool abuse under realistic deployment constraints [93, 67]. This implies that surveys should treat threat models as part of the agent design space, not as an external appendix.

Concrete implementations of security appear in Gasmi et al. [20]; Hadeliya et al. [22]; Luo et al. [53]; Sha et al. [64]; An et al. [2]; and Liu et al. [49].

Threat models for agents go beyond content manipulation. When an agent delegates actions to tools, attackers can target the decision process (e.g., steering tool selection) or the state (e.g., poisoning memory) rather than only the final output. Progent-style evaluations emphasize agent use cases and benchmarked attack conditions (e.g., AgentDojo/ASB/AgentPoison-style setups), illustrating that vulnerability depends on interface exposure and tool permissions as much as on model behavior [67]. However, reported results can be hard to compare without standardized protocols.

Mitigations often take the form of checks, monitors, and constrained execution. Systems that emphasize systematic checking aim to catch tool misuse and unsafe actions before they execute, but they also introduce latency and can create new failure modes when checks are incomplete or mis-specified [5, 36]. In contrast, lightweight guardrails may preserve usability but provide weaker guarantees, especially under adaptive attackers.

A governance-oriented view reframes safety as an engineering contract: what actions are allowed, how they are audited, and what rollback mechanisms exist. Reliability and evaluation work suggests that safety claims should be tied to measurable protocol conditions (budgets, tool access, logging), because otherwise "safe" can mean "safe under a different harness" [32, 18]. Therefore, practical deployments may require layered controls: sandboxing at the tool layer, provenance at the memory layer, and monitoring at the decision layer [45].

Taken together, agent safety research benefits from integrating threat models with evaluation. Attack taxonomies provide coverage targets, while benchmarks provide measurable failure rates under controlled conditions. This suggests that future evaluation suites should explicitly

parameterize adversarial strength, tool permissions, and the cost of verification, rather than treating these as hidden implementation details [93, 67].

Moreover, governance constraints are often external to research prototypes: real organizations impose compliance, privacy, and audit requirements that benchmarks rarely model. Bridging this gap likely requires collaboration between research and deployment communities, and a shift toward reporting "safety envelopes" (what the system can guarantee under what assumptions) rather than one-size-fits-all safety claims [38, 32].

A concrete governance view is to describe "what can go wrong" in the same language as "what can be done." If the action space includes file I/O, web access, or code execution, then the governance layer must specify which actions are permitted, how they are audited, and what escalation paths exist when anomalies are detected [45]. In contrast, purely content-level guardrails may miss tool-layer abuse, which is why safety work increasingly ties checks to execution events rather than only to generated text [5, 36].

Finally, governance constraints need to be made measurable. Reliability framing suggests reporting "safety envelopes" tied to protocol parameters: budgets, permissions, monitoring latency, and false positive/negative rates of checks [32, 18]. This remains limited by benchmark realism—organizational compliance and privacy constraints are rarely modeled—but the direction is clear: treat governance as a system contract with testable conditions, and report those conditions as part of the evaluation protocol rather than as a narrative disclaimer [38, 93].

At the system boundary, permissioning and provenance are the concrete levers that connect abstract threat models to implementable mitigations. If a benchmark treats tool calls as always-successful and side-effect free, it cannot meaningfully measure prompt injection, exfiltration, or tool abuse; conversely, realistic permissions and failure modes force methods to specify how they detect anomalies and when they escalate or refuse actions [93, 67]. This is also why checkers and monitors should be evaluated with their own operating characteristics (latency, false positives/negatives), since aggressive checking can shift risk rather than eliminate it [5, 36].

Governance becomes actionable when expressed as a testable contract: what events are logged, who can inspect them, and what interventions exist (rollback, sandboxing, human review). Reliability-focused framing encourages reporting these contracts alongside the evaluation harness, because monitoring latency and audit policies can change the effective threat model even if the underlying agent policy is unchanged [32, 18]. Practical deployments also motivate least-privilege tool design and clear escalation paths, so that when an agent fails it fails safely; this engineering detail deserves to be part of survey comparisons rather than an afterthought [45, 38].

7 Discussion

A recurring pattern across agent systems is that progress often comes from tightening contracts rather than adding new prompts. When the tool boundary is explicit (what can be called, with what permissions, and under what cost model), it becomes easier to reason about failure recovery and to compare planning or memory strategies under consistent protocols [16, 47]. In contrast, when the boundary is implicit, many improvements become hard to attribute: gains may come from better prompting, better tool routing, or simply different evaluation settings.

A second theme is that evaluation is still the bottleneck. Benchmarks are proliferating, but protocol details (cost budgets, tool availability, reset policies, and contamination controls) often vary enough that "higher score" does not translate into a clear engineering decision [54, 18]. This gap is especially visible for long-horizon tasks, where small differences in memory policy or error handling compound over time.

Finally, safety and governance are not add-ons; they are entangled with interface design. Prompt injection and tool abuse are easiest to study when systems expose a clear action space and logging surface, but these same interfaces also enlarge the attack surface if permissions

and provenance are under-specified [93, 67]. A practical implication is that agent research should report threat models and monitoring hooks as first-class experimental conditions, not as post-hoc disclaimers.

8 Conclusion

Evidence from recent tool-using agent work suggests that "agent performance" is not a single axis: interface contracts, planning/memory components, and adaptation/coordination schemes interact with evaluation protocols in ways that can reverse conclusions across benchmarks. As a result, survey taxonomies should be decision-oriented: they should help a reader predict which design choices change reliability, cost, and risk under a stated protocol.

Looking forward, the most valuable work may be less about inventing new loop variants and more about standardizing what it means to compare them: shared task suites, explicit budgets and tool access policies, and threat-model-aware evaluations that reflect real deployment constraints [54, 93].

References

[1] Pravallika Abbineni, Saoud Aldowaish, Colin Liechty, Soroosh Noorzad, Ali Ghazizadeh, and Morteza Fayazi. Muallm: A multimodal large language model agent for circuit design assistance with hybrid contextual retrieval-augmented generation. *arXiv preprint arXiv:2508.08137v1*, 2025. URL http://arxiv.org/abs/2508.08137v1.

[2] Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. Ipiguard: A novel tool dependency graph-based defense against indirect prompt injection in llm agents. *arXiv preprint arXiv:2508.15310v1*, 2025. URL http://arxiv.org/abs/2508.15310v1.

[3] Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Andrey Kravchenko, Mikhail Burtsev, and Evgeny Burnaev. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363v3*, 2024. URL http://arxiv.org/abs/2407.04363v3.

[4] Nikolas Belle, Dakota Barnes, Alfonso Amayuelas, Ivan Bercovich, Xin Eric Wang, and William Wang. Agents of change: Self-evolving llm agents for strategic planning. *arXiv preprint arXiv:2506.04651v2*, 2025. URL http://arxiv.org/abs/2506.04651v2.

[5] Vamshi Krishna Bonagiri, Ponnurangam Kumaragurum, 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.

[6] 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.

[7] Edward Y. Chang and Longling Geng. Alas: A stateful multi-llm agent framework for disruption-aware planning. *arXiv preprint arXiv:2505.12501v1*, 2025. URL http://arxiv.org/abs/2505.12501v1.

[8] 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.

[9] 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.

[10] Yize Cheng, Arshia Soltani Moakhar, Chenrui Fan, Parsa Hosseini, Kazem Faghih, Zahra Sodagar, Wenxiao Wang, and Soheil Feizi. Your llm agents are temporally blind: The misalignment between tool use decisions and human time perception. *arXiv preprint arXiv:2510.23853v2*, 2025. URL http://arxiv.org/abs/2510.23853v2.

[11] 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.

[12] Gautier Dagan, Frank Keller, and Alex Lascarides. Plancraft: an evaluation dataset for planning with llm agents. *arXiv preprint arXiv:2412.21033v2*, 2024. URL http://arxiv.org/abs/2412.21033v2.

[13] 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.

[14] Jia-Kai Dong, I-Wei Huang, Chun-Tin Wu, and Yi-Tien Tsai. Msc-bench: A rigorous benchmark for multi-server tool orchestration. *arXiv preprint arXiv:2510.19423v1*, 2025. URL http://arxiv.org/abs/2510.19423v1.

[15] Xingbo Du, Loka Li, Duzhen Zhang, and Le Song. Memr^3: Memory retrieval via reflective reasoning for llm agents. *arXiv preprint arXiv:2512.20237v1*, 2025. URL http://arxiv.org/abs/2512.20237v1.

[16] 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.

[17] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978v3*, 2025. URL http://arxiv.org/abs/2505.10978v3.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Tsimur Hadeliya, Mohammad Ali Jauhar, Nidhi Sakpal, and Diogo Cruz. When refusals fail: Unstable safety mechanisms in long-context llm agents. *arXiv preprint arXiv:2512.02445v1*, 2025. URL http://arxiv.org/abs/2512.02445v1.

[23] Amine Ben Hassouna, Hana Chaari, and Ines Belhaj. Llm-agent-umf: Llm-based agent unified modeling framework for seamless design of multi active/passive core-agent architectures. *arXiv preprint arXiv:2409.11393v3*, 2024. URL http://arxiv.org/abs/2409.11393v3.

[24] Kostas Hatalis, Despina Christou, and Vyshnavi Kondapalli. Review of case-based reasoning for llm agents: Theoretical foundations, architectural components, and cognitive integration. *arXiv preprint arXiv:2504.06943v2*, 2025. URL http://arxiv.org/abs/2504.06943v2.

[25] Yufei He, Ruoyu Li, Alex Chen, Yue Liu, Yulin Chen, Yuan Sui, Cheng Chen, Yi Zhu, Luca Luo, Frank Yang, and Bryan Hooi. Enabling self-improving agents to learn at test time with human-in-the-loop guidance. *arXiv preprint arXiv:2507.17131v2*, 2025. URL http://arxiv.org/abs/2507.17131v2.

[26] Joey Hong, Anca Dragan, and Sergey Levine. Planning without search: Refining frontier llms with offline goal-conditioned rl. *arXiv preprint arXiv:2505.18098v2*, 2025. URL http://arxiv.org/abs/2505.18098v2.

[27] 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.

[28] Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257v2*, 2025. URL http://arxiv.org/abs/2507.05257v2.

[29] Jiayuan Huang, Runlong He, Danyal Zaman Khan, Evangelos B. Mazomenos, Danail Stoyanov, Hani Marcus, Linzhe Jiang, Matthew J Clarkson, and Mobarak I. Hoque. Surgical ai copilot: Energy-based fourier gradient low-rank adaptation for surgical llm agent reasoning and planning. *arXiv preprint arXiv:2503.09474v2*, 2025. URL http://arxiv.org/abs/2503.09474v2.

[30] 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.

[31] Jingyi Jia and Qinbin Li. Autotool: Efficient tool selection for large language model agents. *arXiv preprint arXiv:2511.14650v1*, 2025. URL http://arxiv.org/abs/2511.14650v1.

[32] Neil Kale, Chen Bo Calvin Zhang, Kevin Zhu, Ankit Aich, Paula Rodriguez, Scale Red Team, Christina Q. Knight, and Zifan Wang. Reliable weak-to-strong monitoring of llm agents. *arXiv preprint arXiv:2508.19461v1*, 2025. URL http://arxiv.org/abs/2508.19461v1.

[33] 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.

[34] 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.

[35] Andrew Kiruluta. A novel architecture for symbolic reasoning with decision trees and llm agents. *arXiv preprint arXiv:2508.05311v1*, 2025. URL http://arxiv.org/abs/2508.05311v1.

[36] 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.

[37] Chuanhao Li, Runhan Yang, Tiankai Li, Milad Bafarassat, Kourosh Sharifi, Dirk Bergemann, and Zhuoran Yang. Stride: A tool-assisted llm agent framework for strategic and interactive decision-making. *arXiv preprint arXiv:2405.16376v2*, 2024. URL http://arxiv.org/abs/2405.16376v2.

[38] 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.

[39] Qiumeng Li, Chunhou Ji, and Xinyue Liu. From narrative to action: A hierarchical llm-agent framework for human mobility generation. *arXiv preprint arXiv:2510.24802v1*, 2025. URL http://arxiv.org/abs/2510.24802v1.

[40] 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.

[41] 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.

[42] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, Rui Kong, Yile Wang, Hanfei Geng, Jian Luan, Xuefeng Jin, Zilong Ye, Guanjing Xiong, Fan Zhang, Xiang Li, Mengwei Xu, Zhijun Li, Peng Li, Yang Liu, Ya-Qin Zhang, and Yunxin Liu. Personal llm agents: Insights and survey about the capability, efficiency and security. *arXiv preprint arXiv:2401.05459v2*, 2024. URL http://arxiv.org/abs/2401.05459v2.

[43] Yuanhao Li, Mingshan Liu, Hongbo Wang, Yiding Zhang, Yifei Ma, and Wei Tan. Draft-rl: Multi-agent chain-of-draft reasoning for reinforcement learning-enhanced llms. *arXiv preprint arXiv:2511.20468v1*, 2025. URL http://arxiv.org/abs/2511.20468v1.

[44] 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.

[45] 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.

[46] Aly Lidayan, Jakob Bjorner, Satvik Golechha, Kartik Goyal, and Alane Suhr. Abbel: Llm agents acting through belief bottlenecks expressed in language. *arXiv preprint arXiv:2512.20111v1*, 2025. URL http://arxiv.org/abs/2512.20111v1.

[47] 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.

[48] Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. Toolscope: Enhancing llm agent tool use through tool merging and context-aware filtering. *arXiv preprint arXiv:2510.20036v1*, 2025. URL http://arxiv.org/abs/2510.20036v1.

[49] 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.

[50] Wenrui Liu, Zixiang Liu, Elsie Dai, Wenhan Yu, Lei Yu, and Tong Yang. Mcpagentbench: 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.

[51] Keer Lu, Chong Chen, Xili Wang, Bin Cui, Yunhuai Liu, and Wentao Zhang. Pilotrl: Training language model agents via global planning-guided progressive reinforcement learning. *arXiv preprint arXiv:2508.00344v4*, 2025. URL http://arxiv.org/abs/2508.00344v4.

[52] 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.

[53] Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. Agrail: A lifelong agent guardrail with effective and adaptive safety detection. *arXiv preprint arXiv:2502.11448v2*, 2025. URL http://arxiv.org/abs/2502.11448v2.

[54] 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.

[55] Katsuaki Nakano, Reza Fayyazi, Shanchieh Jay Yang, and Michael Zuzak. Guided reasoning in llm-driven penetration testing using structured attack trees. *arXiv preprint arXiv:2509.07939v2*, 2025. URL http://arxiv.org/abs/2509.07939v2.

[56] 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.

[57] 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.

[58] Jielin Qiu, Zuxin Liu, Zhiwei Liu, Rithesh Murthy, Jianguo Zhang, Haolin Chen, Shiyu Wang, Ming Zhu, Liangwei Yang, Juntao Tan, Roshan Ram, Akshara Prabhakar, Tulika Awalgaonkar, Zixiang Chen, Zhepeng Cen, Cheng Qian, Shelby Heinecke, Weiran Yao, Silvio Savarese, Caiming Xiong, and Huan Wang. Locobench-agent: An interactive benchmark for llm agents in long-context software engineering. *arXiv preprint arXiv:2511.13998v1*, 2025. URL http://arxiv.org/abs/2511.13998v1.

[59] 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.

[60] Vishnu Sarukkai, Asanshay Gupta, James Hong, Michaël Gharbi, and Kayvon Fatahalian. In-context distillation with self-consistency cascades: A simple, training-free way to reduce llm agent costs. *arXiv preprint arXiv:2512.02543v1*, 2025. URL http://arxiv.org/abs/2512.02543v1.

[61] 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.

[62] Philipp J. Schneider, Lin Tian, and Marian-Andrei Rizoiu. Learning to make friends: Coaching llm agents toward emergent social ties. *arXiv preprint arXiv:2510.19299v1*, 2025. URL http://arxiv.org/abs/2510.19299v1.

[63] 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.

[64] Zeyang Sha, Hanling Tian, Zhuoer Xu, Shiwen Cui, Changhua Meng, and Weiqiang Wang. Agent safety alignment via reinforcement learning. *arXiv preprint arXiv:2507.08270v1*, 2025. URL http://arxiv.org/abs/2507.08270v1.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] Dehao Tao, Guoliang Ma, Yongfeng Huang, and Minghu Jiang. Membox: Weaving topic continuity into long-range memory for llm agents. *arXiv preprint arXiv:2601.03785v1*, 2026. URL http://arxiv.org/abs/2601.03785v1.

[70] 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.

[71] 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.

[72] 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:2305.16291v2*, 2023. URL http://arxiv.org/abs/2305.16291v2.

[73] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073v2*, 2025. URL http://arxiv.org/abs/2504.20073v2.

[74] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857v1*, 2025. URL http://arxiv.org/abs/2511.20857v1.

[75] Chunlong Wu, Ye Luo, Zhibo Qu, and Min Wang. Meta-policy reflexion: Reusable reflective memory and rule admissibility for resource-efficient llm agent. *arXiv preprint arXiv:2509.03990v2*, 2025. URL http://arxiv.org/abs/2509.03990v2.

[76] 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.

[77] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079v1*, 2025. URL http://arxiv.org/abs/2510.16079v1.

[78] 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.

[79] Siyu Xia, Zekun Xu, Jiajun Chai, Wentian Fan, Yan Song, Xiaohan Wang, Guojun Yin, Wei Lin, Haifeng Zhang, and Jun Wang. From experience to strategy: Empowering llm agents with trainable graph memory. *arXiv preprint arXiv:2511.07800v1*, 2025. URL http://arxiv.org/abs/2511.07800v1.

[80] Yu Xia, Yiran Shen, Junda Wu, Tong Yu, Sungchul Kim, Ryan A. Rossi, Lina Yao, and Julian McAuley. Sand: Boosting llm agents with self-taught action deliberation. *arXiv preprint arXiv:2507.07441v2*, 2025. URL http://arxiv.org/abs/2507.07441v2.

[81] R. Patrick Xian, Qiming Cui, Stefan Bauer, and Reza Abbasi-Asl. Measuring temporal effects of agent knowledge by date-controlled tool use. *arXiv preprint arXiv:2503.04188v2*, 2025. URL http://arxiv.org/abs/2503.04188v2.

[82] Jingao Xu, Shuoyoucheng Ma, Xin Song, Rong Jiang, Hongkui Tu, and Bin Zhou. Exemplar-guided planing: Enhanced llm agent for kgqa. *arXiv preprint arXiv:2510.15283v1*, 2025. URL http://arxiv.org/abs/2510.15283v1.

[83] 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.

[84] Wei Yang, Jinwei Xiao, Hongming Zhang, Qingyang Zhang, Yanna Wang, and Bo Xu. Coarse-to-fine grounded memory for llm agent planning. *arXiv preprint arXiv:2508.15305v1*, 2025. URL http://arxiv.org/abs/2508.15305v1.

[85] 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.

[86] Ye Ye. Task memory engine: Spatial memory for robust multi-step llm agents. *arXiv preprint arXiv:2505.19436v1*, 2025. URL http://arxiv.org/abs/2505.19436v1.

[87] Ye Ye. Task memory engine (tme): Enhancing state awareness for multi-step llm agent tasks. *arXiv preprint arXiv:2504.08525v4*, 2025. URL http://arxiv.org/abs/2504.08525v4.

[88] Yauwai Yim, Chunkit Chan, Tianyu Shi, Zheye Deng, Wei Fan, Tianshi Zheng, and Yangqiu Song. Evaluating and enhancing llms agent based on theory of mind in guandan: A multi-player cooperative game under imperfect information. *arXiv preprint arXiv:2408.02559v1*, 2024. URL http://arxiv.org/abs/2408.02559v1.

[89] Ziming You, Yumiao Zhang, Dexuan Xu, Yiwei Lou, Yandong Yan, Wei Wang, Huaming Zhang, and Yu Huang. Datawiseagent: A notebook-centric llm agent framework for adaptive and robust data science automation. *arXiv preprint arXiv:2503.07044v2*, 2025. URL http://arxiv.org/abs/2503.07044v2.

[90] 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.

[91] Rasoul Zahedifar, Sayyed Ali Mirghasemi, Mahdieh Soleymani Baghshah, and Alireza Taheri. Llm-agent-controller: A universal multi-agent large language model system as a control engineer. *arXiv preprint arXiv:2505.19567v1*, 2025. URL http://arxiv.org/abs/2505.19567v1.

[92] 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.

[93] 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.

[94] 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.

[95] Jusheng Zhang, Yijia Fan, Kaitong Cai, Xiaofei Sun, and Keze Wang. Osc: Cognitive orchestration through dynamic knowledge alignment in multi-agent llm collaboration. *arXiv preprint arXiv:2509.04876v1*, 2025. URL http://arxiv.org/abs/2509.04876v1.

[96] Xin Zhang, Lissette Iturburu, Juan Nicolas Villamizar, Xiaoyu Liu, Manuel Salmeron, Shirley J. Dyke, and Julio Ramirez. Large language model agent for structural drawing generation using react prompt engineering and retrieval augmented generation. *arXiv preprint arXiv:2507.19771v1*, 2025. URL http://arxiv.org/abs/2507.19771v1.

[97] 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.

[98] Xingfu Zhou and Pengfei Wang. Reasoning-style poisoning of llm agents via stealthy style transfer: Process-level attacks and runtime monitoring in rsv space. *arXiv preprint arXiv:2512.14448v1*, 2025. URL http://arxiv.org/abs/2512.14448v1.

[99] Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446v1*, 2024. URL http://arxiv.org/abs/2402.19446v1.

[100] 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.

[101] Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370v1*, 2025. URL http://arxiv.org/abs/2509.25370v1.