agent survey latex

January 25, 2026

Contents

Abstract

Large language models are increasingly deployed as agents: closed-loop systems that plan, act through external tools, observe feedback, and adapt across long-horizon tasks. This shift turns "prompting" into a systems problem, where interface contracts (what actions are executable) and evaluation protocols (what results are comparable) jointly determine whether reported capabilities transfer. Recent work has therefore expanded along two coupled fronts: richer tool-use and orchestration mechanisms (from textual action traces to structured tool schemas) [94, 69, 16], and more demanding benchmarks that surface long-horizon brittleness, cost sensitivity, and security risks [62, 22].

This survey organizes the agent literature through a protocol-first lens: we treat the agent boundary and tool interface as the primary contract, then examine how planning, memory, adaptation, and multi-agent coordination interact with evaluation assumptions. We emphasize evidence-backed contrasts, highlight where measurements are not commensurate across papers, and summarize representative approaches and benchmark settings in Appendix tables for rapid comparison.

1 Introduction

Large language models are increasingly embedded in agents that do more than respond: they decide, act, observe, and iterate under real constraints. A practical agent is therefore a closed-loop system whose behavior depends not only on the base model, but also on the action space it is allowed to execute, the feedback it can observe, and the protocol that couples these components over many steps [94, 83, 75].

A recurring lesson in recent work is that interfaces are not plumbing: they are contracts. Tool APIs, schemas, permissions, and observability determine what actions are even expressible and which failures are detectable. As the community moves from ad-hoc textual tool calls toward more structured orchestration and tool standards, claims that look similar on paper can become incomparable in practice unless the interface contract is made explicit [69, 16, 58, 42].

Beyond interface design, the agent loop itself has diversified. Planning and search style loops trade off deliberation depth against latency and cost, while memory and retrieval mechanisms trade off longer context against staleness, grounding, and citation hygiene [95, 41, 79]. Adaptation methods add another axis: self-improvement can raise performance, but it also changes what should be held fixed in evaluation (policy updates, tool access, or budget assumptions) [103, 17].

These dynamics make evaluation the central bottleneck. Many benchmarks measure end-task success, but the results can be dominated by protocol details: tool catalog design, budget and cost models, retry policies, environment stochasticity, and security assumptions. Security benchmarks further show that the same interface surface that enables tool use also enables prompt injection, tool abuse, and data exfiltration, which must be evaluated end-to-end rather than as isolated model vulnerabilities [62, 73, 22, 102].

This survey adopts a protocol-first organization. After defining the agent boundary and interface contract, we review the design space across (i) planning and reasoning loops, (ii) memory and retrieval, (iii) self-improvement and multi-agent coordination, and (iv) benchmarks and risk surfaces. Our goal is not to exhaustively summarize every paper, but to make comparisons auditable: each subsection is framed around concrete trade-offs, protocol anchors, and limitations, and the Appendix tables provide compact maps of representative approaches and benchmark settings.

We retrieved an arXiv candidate pool of 800 papers (809 before deduplication) using keyword-based queries focused on LLM agents and tool use, and curated a 220-paper core set for mapping and evidence extraction. Unless otherwise stated, our evidence is derived from metadata and abstracts rather than full-text annotation, so we report quantitative claims only when they are explicitly supported by the cited sources.

2 Related Work

Recent surveys and benchmarks have begun to systematize agent behavior, but the field is still fragmented across interface assumptions and evaluation protocols. Broad overviews of agent evaluation emphasize that reported gains are often inseparable from benchmark design, tool access, and cost constraints, motivating protocol-aware comparisons rather than "leaderboard-first" summaries [62, 43].

Our survey is aligned with this diagnosis, but differs in emphasis: we treat the agent boundary and tool interface as an explicit contract, then use evaluation protocols as the main lens for synthesis. This framing helps connect systems work on orchestration and replayability to evaluation work on comparability, and it makes cross-paper disagreements easier to attribute to concrete assumptions rather than to vague "implementation differences" [16, 58, 42].

On the interface side, tool use spans a spectrum from prompting patterns that interleave reasoning and actions to training and scaffolding methods that treat tool invocation as a structured prediction problem. Work on self-instructed tool use and action-trace prompting highlights how much behavior depends on the representation of actions and feedback [69, 94], while orchestration-focused benchmarks and standards highlight the need for explicit tool schemas and reproducible tool-call traces [16, 58, 42]. Tool-selection methods and protocol critiques further show that small changes in evaluation setup can inflate pass rates, making "tool use" a moving target unless the contract is fixed [17, 55].

Inside the agent loop, planning and search methods often resemble deliberative inference under budget constraints, where the main question is not whether the model can reason, but how reasoning is externalized, branched, and terminated. Deliberate search-style prompting and reflection-based adaptation illustrate this tension, but they also underscore that comparisons are brittle without shared environments and budgets [95, 75]. Memory and retrieval mechanisms add a complementary line of work that treats long-horizon performance as an information management problem, spanning compression, citation grounding, and retrieval-augmented execution [41, 79, 1]. Routing and adaptation methods then sit at the intersection, improving performance by changing which evidence and tools are used at each step, which again shifts the evaluation baseline [103].

Multi-agent work introduces additional degrees of freedom: role specialization, communication protocols, and aggregation mechanisms. Frameworks for multi-turn training and evaluation, as well as tool-augmented multi-agent settings, highlight that coordination quality depends on protocol details (who speaks, when tools are called, and how disagreement is resolved), not just on the base model [5, 72, 59].

Coordination also blurs into adaptation: multi-agent systems often change the effective policy by reallocating work across agents, changing the verification budget, or modifying tool access at run time. This makes cost normalization and protocol accounting central, since a coordination gain can reflect better division of labor, whereas it can also reflect a hidden increase in search or redundancy that would not be acceptable under a fixed deployment budget [103, 112, 5].

Finally, security and risk-focused lines of work argue that agent evaluation must include threat models and end-to-end tool-chain behavior. Security benchmarks and red-teaming workflows show that agents can remain "competent" on primary tasks while being highly vulnerable to tool-layer attacks, and that adversarial effectiveness can depend strongly on chained or protocol-aware attacks [22, 102, 39, 109]. Our survey integrates these strands by using interface contracts and evaluation protocols as the organizing lens, so that differences in reported results can be traced to concrete assumptions rather than treated as unexplained variance.

3 Foundations & Interfaces

Agent papers often disagree because they silently assume different interfaces: what actions exist, how tools are specified, what feedback is observable, and which failures are recoverable. Treating the interface as a contract helps separate model capability from systems design choices, and it also clarifies why "tool use" results can be brittle across environments and protocols [69, 94, 58].

We first define the agent loop and its action spaces as the basic abstraction that ties state, decisions, actions, and observations into a measurable protocol. We then examine tool interfaces and orchestration mechanisms, focusing on how schema design, routing, and replayability constraints shape both correctness and what evaluation claims can be compared across studies [16, 42].

3.1 Agent loop and action spaces

For system builders, the first decision is what counts as an action: whether an agent "acts" by emitting text, calling an API, manipulating a UI, or controlling an embodied interface. That choice fixes the action space and therefore determines what kinds of failures can be detected, retried, or recovered within a closed-loop protocol [94, 77, 76]. It also determines what "success" means: if an action is a structured tool call, then executability and schema compliance become measurable; if an action is free-form text, then correctness is mediated by downstream interpreters and implicit conventions.

A second decision is the scope of the loop: what is treated as state, what feedback is observable, and what resets between episodes. Multi-domain evaluations make this explicit by spanning environments with very different affordances (web, embodied, tool-use, games), which helps distinguish general loop assumptions from benchmark-specific quirks [71, 49]. In practice, two agents can share the same base model yet behave very differently if one has richer observability (tool traces, error codes, intermediate state) while the other operates with partial or noisy feedback.

Within this framing, "better agents" can come from very different sources: improved control policies, better representations of actions, or better search over agent designs. Automated agent search reports gains when evaluated across heterogeneous benchmarks, but those gains still hinge on how the search space encodes action primitives and observability [49, 86, 65]. This differs from hand-engineered agent scaffolds, where the design space is narrower but the interface contract is often tighter and easier to audit across runs.

These two routes also lead to different evaluation artifacts. Search-centric approaches treat the environment as an objective and optimize agent designs against it, whereas trajectory-centric approaches treat the environment as both a benchmark and a data generator that shapes what the agent can learn from interaction [49, 90].

The evaluation substrate can also be used as a data engine. Work that collects interaction trajectories to finetune tool-using models illustrates that the loop is not just a test harness: it shapes the data distribution the agent will later rely on, and it can amplify or dampen the apparent benefit of downstream training strategies [90, 21]. For example, collecting tool-use trajectories for finetuning can improve downstream behavior, but the gain is conditional on what the environment exposes as actions and feedback (e.g., which errors are surfaced versus silently absorbed) [90]. That blurs the line between "evaluation" and "training": the loop design that defines what is measured also defines what data becomes available for future agents.

Many "agent" failures are better read as mismatches between planning and execution under constraints: a model may generate plausible plans but fail at constraint following or low-level tool execution, especially when tool outputs are noisy or when the environment is partially observed. Comparative evaluations that separate planning from execution reinforce that action-space design and constraint specification are as important as the base model choice [90, 101].

Papers that report only end-task success can hide whether failures arise from poor decision policies or from brittle actuation under an underspecified tool contract.

Interface standards make the contract more legible. Benchmarks grounded in real tool definitions (rather than idealized APIs) highlight how much practical performance depends on tool schemas, permissions, and error semantics that are often hidden in paper summaries [58, 9, 10]. Standardization also makes negative results more actionable: when failures are tied to a named contract element (schema, permission, observability), downstream work can fix the right layer rather than iterating on prompts.

The same interface boundary also defines an attack surface. Comparative analyses of function-calling style architectures versus protocol-based tool interfaces suggest that differences in tool semantics and mediation can shift which vulnerabilities dominate and how easily attacks chain across the loop [24]. In particular, reported comparisons quantify large shifts in attack success under different mediation layers (e.g., higher success under function calling than under MCP in one comparative setup), and show that chained attacks can amplify success rates relative to simple attacks, which makes "secure by default" assumptions fragile unless the loop is explicitly hardened [24].

A recurring limitation is that loop design and evaluation cannot optimize all objectives at once: pushing for stronger task success, stronger safety guarantees, and stronger efficiency can create tensions that look like "model limits" but are often system-level trade-offs [103]. In contrast to single-benchmark improvements, cross-domain transfer often fails precisely because the action space and feedback contract change. Results in specialized domains (e.g., geometry or domain-specific optimization agents) further emphasize that transferring an action policy across domains typically requires rethinking the action space and protocol, not just swapping models [107, 25].

A practical reading rule emerges from this literature: when two papers report different "agent performance", first align their action spaces and observability assumptions, then compare policies under a shared protocol. Without that contract alignment, improvements can be real yet non-composable across settings, whereas apparent regressions can simply be artifacts of incomparable loops and evaluation assumptions [49, 90, 58, 23]. "Agent progress" is best tracked by tightening and standardizing loop contracts, not only by scaling models. The most informative comparisons make the loop contract explicit before attributing outcomes to model choice.

Once the agent loop defines how decisions translate into actions, the next question is how those actions are grounded in concrete tool interfaces and orchestration policies that shape reliability and risk.

3.2 Tool interfaces and orchestration

Tool interfaces are not only a way to "call APIs"; they define the language in which an agent can express intent, specify constraints, and interpret feedback. As a result, orchestration design largely determines whether an agent's behavior is auditable and whether evaluation results transfer across environments that expose different tools and error semantics [16, 42].

A useful distinction is between ad-hoc interfaces (where tools are described in free-form text and invoked via loosely structured prompts) and schema-first interfaces (where tool calls are typed, validated, and replayable). Whereas ad-hoc interfaces rely on natural language negotiation to encode constraints and tool semantics, schema-first designs shift work from "prompt cleverness" to contract engineering, making failures more attributable (interface ambiguity vs model behavior) and traces more replayable for audits [42, 93].

Benchmark design increasingly reflects this shift. Hierarchical, multi-hop orchestration benchmarks emphasize end-to-end execution rather than isolated tool selection, forcing agents to manage intermediate states, tool dependencies, and failure recovery across multiple calls [16]. This matters because the "hard part" is often not picking the right tool once, but composing

a tool chain under partial observability and brittle intermediate outputs. Orchestration also makes error semantics visible: whether the agent can inspect intermediate tool failures and recover determines whether a multi-hop plan is robust or brittle.

At the same time, protocol details can inflate apparent success. Critiques of common evaluation setups show that pass rates can be artificially high under permissive matching or underspecified constraints, which makes "tool-use capability" hard to compare unless the protocol explicitly defines what counts as a correct tool trace and which errors are recoverable [17, 46]. A recurring theme is that evaluation must specify both surface-form correctness (the tool call) and semantic correctness (the resulting state), otherwise agents can optimize the scoring shortcut rather than the intended capability. Replayability constraints also change what can be audited: if the same tool trace cannot be reproduced under the same schema and permissions, then "success" may not be a stable property of the agent at all [42].

Self-training and adversarial self-challenging approaches report improvements on multi-turn tool-use benchmarks, but their claims are meaningful only relative to the fixed tool catalog, turn budget, and scoring rules of the benchmark. In practice, this means that improvements should be read as "under this tool interface contract" rather than as generic tool-use competence, because the training signal is shaped by the same interface constraints the system is later evaluated under [112].

Tool routing and tool selection remain a common failure point, especially when the tool set is large or when tools are semantically overlapping. Methods that explicitly target tool selection accuracy can therefore improve reliability without changing the underlying reasoning loop, but they again depend on how tools are represented and on what negative examples the protocol includes [55, 27, 38, 89, 114]. In contrast to orchestration-heavy settings where execution is the bottleneck, selection-heavy settings can fail simply because the interface makes multiple tools look equivalent to the model.

Cost and robustness constraints provide another lens that typical accuracy-only leaderboards miss. Attacks that expand tool-use tasks into extremely long trajectories illustrate how token budgets, cost models, and tool output noise can dominate end-to-end viability even when nominal task success remains high [109]. Reported attacks on tool-use benchmarks quantify this gap by showing orders-of-magnitude cost inflation under adversarial prompting, which is hard to detect if evaluation reports only binary success and ignores resource accounting [109].

Orchestration complexity also interacts with coordination strategies. Multi-LLM or multi-agent settings can reduce single-model brittleness through specialization or verification, but they can also amplify failure modes when tool schemas, communication formats, or aggregation rules are not fixed as part of the protocol [72, 59]. Here again the key contrast is contractual: coordination helps when the protocol defines how agents share state and validate tool traces, whereas it can devolve into redundant search when the contract is implicit and costs are not normalized.

Tool interfaces and orchestration are best evaluated as contracts that jointly constrain correctness, auditability, and cost. Comparisons are most informative when they report (i) the tool schema and permissions model, (ii) the budget and retry policy, and (iii) whether tool traces are replayable under the same scoring rules, since these protocol details often explain variance that otherwise appears as "model differences" [42, 16, 112]. Confidence calibration and determinism signals can further help interpret tool traces, because an agent that knows when it is uncertain can route to verification rather than silently committing an invalid call [93]. This highlights a broader point: interface design and evaluation are coupled, so improving one without fixing the other often yields brittle, non-transferable gains.

4 Core Components (Planning + Memory)

Planning and memory determine how an agent behaves when a single prompt is not enough. Planning governs deliberation and control: how the agent decomposes tasks, explores alternatives, and decides when to commit actions under finite budgets. Memory governs state and evidence: what information the agent can reliably condition on, how it is retrieved, and how it is kept consistent over long horizons [95, 75, 41].

We begin with planning and reasoning loops, where the central trade-off is between deliberation depth and protocol constraints such as latency, tool failures, and termination rules. We then turn to memory and retrieval, emphasizing how retrieval policies and summarization choices affect grounding and comparability, especially when evaluation depends on citations, long contexts, or persistent state [79, 1].

4.1 Planning and reasoning loops

A key trade-off is that more deliberation can improve correctness, yet it increases latency, cost, and the surface area for tool failures. As a result, the question is rarely "can the model plan?", but rather "which planning protocol remains reliable under the environment and budget constraints we actually care about?" [31, 15, 29]. Planning results are most interpretable when they are reported alongside the budgets and failure recovery assumptions that govern the loop.

Prompting-based planners often externalize reasoning as action traces interleaved with tool calls, where the main design choice is how tightly reasoning is coupled to execution. This style can work well on interactive tasks, but it is sensitive to termination rules and to whether the agent can detect and recover from incorrect intermediate steps [94, 11].

Search-style planners push the same idea further by explicitly branching, scoring, and selecting candidate trajectories. These methods highlight that "planning" is as much about evaluation and selection as it is about generation, which makes the choice of scoring functions and constraints a first-class part of the protocol [110, 31, 45, 108]. The choice of scoring signal (self-consistency, tool feedback, or external verification) often determines whether extra deliberation helps or merely adds expensive noise.

Whereas prompting-based planners often resemble a linear "think-act" trace, search-style planners treat reasoning as a structured exploration problem: they explicitly allocate budget to branching, use scoring or verification to select candidates, and therefore make selection criteria part of the algorithm rather than an implicit side effect [94, 110]. In contrast, prompt-only planners can be cheaper and easier to deploy, but they can hide failure modes when intermediate steps are not verifiable or when termination is underspecified [11, 31].

A parallel line trains smaller or specialized models to plan more effectively, often using RL-style objectives that compress multi-step planning into a policy that is cheaper to run at inference time. This creates a different trade-off: trained planners can amortize deliberation cost into parameters, whereas search-based planners pay cost at inference time but can adapt their search to the specific instance and budget [32, 110].

Execution-aware evaluation reveals another gap: plausible plans are not necessarily executable tool traces. Measurements that directly score whether generated tool calls are valid (e.g., executable API calls) make this gap visible, and they often surface failure modes that are hidden by end-task success metrics alone [63, 36]. For instance, reporting the fraction of executable calls provides a concrete protocol anchor that distinguishes "good intentions" from reliable actuation under a fixed tool schema [63]. This also motivates reporting cost and recovery behavior: two planners can reach similar success rates, whereas one may rely on many retries or expensive verification steps that would be unacceptable under a deployment budget [31, 63].

Robust planning under adversarial or noisy tool settings further complicates comparisons. Security-aware evaluations suggest that planning strategies can change attack success rates,

not only by changing what the agent tries, but also by changing how the agent interprets tool outputs and error states [73, 44]. Here the key contrast is between planners that treat tool outputs as trusted observations and planners that treat them as adversarially contaminated signals requiring verification or redundancy, which leads to different failure recovery behavior under the same threat model [73].

Testing frameworks and agent benchmarks provide a complementary lens by turning planning failures into reproducible test cases. This matters because without systematic testing, improvements can be overfit to the quirks of a single benchmark environment rather than to the underlying decision-making problem [36, 15, 113]. Test suites can also make it clear which failure recovery behaviors are expected (retry, backtrack, defer), which closes the gap between "planning quality" and deployable reliability.

Domain-specific agents illustrate the same theme: strong results often come from aligning the planning loop with domain constraints and data representations. For example, task-oriented agents for structured domains require planners that can operate over domain-appropriate state abstractions rather than over generic text-only action traces [74, 51].

Planning and reasoning loops are easiest to interpret when they are reported as protocols: action representation, scoring and selection rules, budgets, retry policies, and failure recovery mechanisms. When those protocol details are made explicit, comparisons across prompt-based, search-based, and trained planners become more interpretable, and limitations (e.g., brittle execution, unclear termination) can be traced to contract mismatch rather than treated as unexplained variance [31, 110, 32, 36]. This framing also makes it easier to identify what must be standardized for results to transfer across environments. A simple reporting norm follows: treat the planning loop itself as the evaluated object, not just the final answer.

Planning determines how actions are chosen over time, whereas memory determines what evidence those choices can reliably condition on under a fixed protocol.

4.2 Memory and retrieval (RAG)

Memory is the subsystem that turns a single interaction into a persistent process: it determines what the agent can condition on, what it can retrieve, and how it maintains consistency across long horizons. The core tension is that richer memory can improve grounding and reduce hallucination, yet it can also introduce staleness, retrieval errors, and citation drift that are invisible if evaluation only measures end-task success [79, 34, 86].

Most agent memory mechanisms can be read as RAG variants embedded in a loop: an index over external information, a retrieval policy, and a write/update policy that decides what gets stored. This framing makes it clear why "more context" is not always better: the protocol must specify how evidence is selected and how it is used during action selection [100, 99, 92]. Once memory is part of the loop, errors can compound: a single mistaken write can bias future retrieval and turn a local mistake into a persistent behavioral drift.

One pragmatic direction is to make memory cheaper. Distillation-style approaches suggest that smaller models can inherit retrieval- or memory-conditioned behavior from larger systems, but the resulting gains depend on the training distribution and on the evaluation tasks used to define "memory usefulness" [41].

Whereas distillation amortizes memory behavior into parameters, another direction is to make memory more active: retrieval can be conditioned on the agent's current plan, and the agent can iteratively refine what it searches for. Evaluations that explicitly stress context-intensive problem instances help expose when performance is driven by retrieval quality versus by reasoning or tool execution, which is crucial for interpreting gains as "better memory" rather than as accidental benchmark fit [82, 47, 41, 78].

Benchmarking memory requires metrics that do not collapse into generic "accuracy". Custom datasets that separate retrieval/citation behavior from reasoning behavior illustrate that agents

can be strong at retrieval but weak at attributing evidence correctly, or vice versa, depending on how the protocol scores citations and intermediate steps [1, 33, 54].

This separation also clarifies what should be measured during agent execution. A memory system can retrieve the right document yet still degrade decisions if the evidence is not integrated into action selection, and conversely a strong reasoner can still fail if retrieval returns stale or misattributed evidence. Protocols that score intermediate citations and retrieval traces therefore provide more diagnostic signal than end-task accuracy alone [1, 33].

Compression-based memories highlight a related trade-off: summaries can make long contexts tractable, but they also risk erasing details that later steps depend on. Systems that integrate summarization into the retrieval loop therefore need protocols that measure not just task success, but also whether compressed memories preserve the evidence needed for later decisions [79, 67]. Whereas summary-based memories reduce context length and stabilize inputs, retrieval-heavy memories preserve source granularity but can fail through missed retrieval or spurious matches; hybrid designs effectively choose a point on this spectrum per step. Making that choice explicit (when to summarize, when to retrieve raw sources) is often more informative than simply reporting a single RAG score [79, 82]. Naming the retrieval trigger and the write policy (what gets stored, when it is updated) can make memory claims far easier to reproduce than reporting aggregate accuracy alone. It also provides a clear hook for ablations: changing the write policy while holding retrieval fixed (or vice versa) isolates where the gain actually comes from.

Domain-specific memory settings make the same issues concrete. When the environment includes structured artifacts (codebases, documents, or technical diagrams), the memory interface must define what representations are stored and how retrieval results are grounded back to the environment, otherwise evaluation can conflate representation mismatch with model limitations [104, 100].

Limitations often arise from evaluation shortcuts: retrieval systems can look strong when the benchmark implicitly rewards lexical overlap, and they can look weak when the benchmark penalizes any mismatch without accounting for protocol uncertainty. In contrast, task designs that vary evidence availability, scoring strictness, and citation requirements make it harder to overfit to surface cues and therefore provide a more reliable basis for claims about memory robustness [97, 96, 1].

Memory and retrieval are most usefully treated as a contract with explicit protocol elements (index scope, retrieval policy, write policy, and citation grounding). When these elements are reported, it becomes easier to compare memory mechanisms across agents and to diagnose whether failures come from retrieval, reasoning, or execution rather than from an opaque mixture of all three [94, 34, 99]. This implies that memory papers are most reusable when they report not only "what was retrieved", but also "how retrieval affected subsequent actions" under a fixed protocol.

5 Learning, Adaptation & Coordination

Learning and coordination turn an agent from a fixed policy into a system that changes with feedback. Self-improvement methods can raise performance, but they also introduce new failure modes: reward hacking, instability, and protocol dependence (what is held fixed, what is updated, and under which budget) [75, 103].

We first review adaptation mechanisms that modify prompts, policies, or routing based on experience, and we highlight how to read reported gains through the lens of evaluation and cost constraints. We then examine multi-agent coordination, where roles, communication protocols, and aggregation rules determine whether multiple models help verification and coverage or simply add variance and cost [5, 72, 59].

5.1 Self-improvement and adaptation

Adaptation methods promise to turn agents from static prompt programs into systems that improve with experience. The central tension is that learning can raise task success, yet it can also change what is held fixed in evaluation (policy updates, tool access, budgets), making gains hard to interpret unless the protocol specifies the learning loop explicitly [62, 103, 6, 3]. Adaptation claims are most comparable when papers state what was updated (policy, prompts, routing), when updates occur, and how costs and data are accounted.

One family of approaches treats adaptation as reflection and revision: the agent uses its own traces to critique failures, refine plans, or update internal heuristics. In practice, these loops blur the line between "reasoning" and "learning", since the improvement mechanism is itself mediated by the model and by the feedback channel it can access [94, 81, 80, 111]. Whereas reflection-based adaptation often changes the agent's behavior by changing prompts, rules, or memory, it typically does so without changing the tool contract or the benchmark itself. Reflection loops can be brittle when the feedback channel is ambiguous (e.g., sparse reward vs rich traces), because the agent can learn to rationalize failures rather than to correct them.

Another family treats adaptation as data generation and self-training under fixed tools. Self-challenging methods report gains on multi-turn tool-use benchmarks using self-generated training data, but these gains remain conditional on the benchmark's tool catalog, scoring, and budget assumptions [112, 17]. In contrast to reflection loops that reshape behavior at inference time, self-training reshapes the policy via data and optimization, making the "training protocol" part of what must be reported for results to be comparable.

A protocol-first comparison therefore asks: what is being updated, on what signal, and under what accounting? Two methods can both claim "self-improvement", yet one may be updating only a routing policy while another updates the underlying policy parameters; without this distinction, benchmark-to-benchmark transfer results are hard to interpret [62, 112].

Routing-based adaptation emphasizes a different lever: rather than changing the base model, the system changes which tools, prompts, or sub-policies are used in a given context. Results on agentic benchmarks suggest that such routing can improve performance when integrated into existing systems, but it also increases protocol complexity because the routing policy becomes part of the evaluation contract [103, 88]. Whereas weight-updating methods pay their cost during training, routing methods often pay cost during inference (additional calls, verification, or search), so cost normalization becomes decisive.

Evolution-style adaptation pushes this further by explicitly searching over policies, prompts, or routes under an evaluation signal. These methods can be effective, but they also make overfitting risks more acute: when the search objective is tied to a benchmark, the system may learn to exploit the scoring procedure unless constraints and held-out protocols are part of the evaluation design [88, 64]. Evolution-style methods also make reporting burdens higher: without clear accounting of search budget and selection criteria, it is hard to separate algorithmic improvement from brute-force exploration.

When adaptation is viewed through an optimization lens, a persistent failure mode is reward hacking: the system can learn shortcuts that exploit benchmark scoring without improving the underlying capability. This risk is amplified when protocols under-specify constraints or when intermediate traces are not audited, since the learning loop can overfit to hidden degrees of freedom [64, 62]. Held-out protocols and explicit constraint reporting are often more important for adaptation papers than small average score gains.

Safety and reliability constraints further complicate adaptation. Methods that prioritize sandboxing or fault containment highlight that an "improved" policy can still be unacceptable if it increases the likelihood of unsafe tool invocations or makes behavior harder to monitor, which again requires protocol-level reporting rather than anecdotal success stories [91, 64, 30, 18].

Adaptation is also domain-sensitive. Work that demonstrates autonomous performance on specialized scientific or engineering tasks illustrates that "autonomy" depends on domain

interfaces and evaluation settings, and that transfer across domains often requires redesigning the feedback loop rather than simply scaling the same adaptation mechanism [48, 81].

A practical limitation is that many benchmarks treat agents as stationary policies, while adaptation methods explicitly violate that assumption. This makes it easy to conflate learning progress with leakage, implicit curriculum effects, or unreported protocol changes unless the evaluation explicitly accounts for learning dynamics (when updates occur, what data is used, and under which budgets) [62, 112].

Self-improvement results are easiest to compare when papers make three elements explicit: the feedback channel (what signals are available), the update rule (what changes in the agent), and the accounting rule (how cost and data are counted). Making these elements explicit enables fair comparison across reflection-style, self-training, routing, and evolution-style methods, and it makes the remaining limitations (stability, reward hacking, safety) easier to diagnose and reproduce [103, 88, 17, 64]. A simple rule of thumb follows: if an adaptation method cannot state its update and accounting rules precisely, its improvements should be treated as protocol-specific until independently reproduced.

Self-improvement adapts an agent across episodes, but coordination couples multiple agents whose roles and communication protocols introduce new trade-offs and failure modes.

5.2 Multi-agent coordination

A key trade-off is that coordination can improve coverage and robustness, yet it also introduces new failure modes (miscommunication, incentive misalignment, error amplification) and higher cost. Multi-agent systems extend the agent loop by adding communication and aggregation, with the hope that specialization and verification can reduce single-model brittleness, but the gains are meaningful only under an explicit coordination protocol [72, 87, 66, 12].

Most coordination schemes can be decomposed into three primitives: role assignment (who does what), communication protocol (what is exchanged), and aggregation rule (how outputs are combined). Small changes in these primitives can turn "multi-agent" from helpful verification into redundant chatter, which is why coordination results are difficult to compare unless the protocol details are stated explicitly [14, 87, 28, 98]. Whereas simple ensembling aggregates independent answers, role-based coordination tries to create complementarity by constraining who explores, who critiques, and who verifies, which changes both the error model and the cost model [87]. Coordination papers are most comparable when they state role definitions, message formats, and aggregation rules as first-class evaluation metadata rather than as implementation details.

Training and evaluation frameworks for long-horizon coordination emphasize efficiency constraints. Multi-turn training settings highlight that coordination quality depends on asynchronous scheduling, batching, and tool integration, not just on the base models' reasoning ability [5, 15]. The infrastructure layer can change coordination outcomes by changing latency and synchronization, so "algorithmic" comparisons need to control for these protocol-level confounds [5].

Tool- and memory-mediated coordination provides another angle: agents may coordinate by sharing tool state, selectively removing tools, or enforcing shared memory policies. Such interventions can be evaluated as controlled protocol changes that expose how coordination interacts with tool availability and how quickly failures propagate through the group [59, 33]. Shared-state coordination also creates a natural place to enforce policies (e.g., tool gating and consistency checks), which can make the system more auditable than purely conversational coordination.

Tree-structured coordination and deliberation strategies highlight a complementary mechanism: rather than relying on a single trajectory, the system explores multiple branches and uses aggregation to select or verify candidates. This shifts the bottleneck to evaluation and selection, making scoring functions and stopping rules a key part of the coordination contract

[35, 14]. This style trades off breadth for accounting: tree search can reduce single-trajectory brittleness, whereas it can also behave like implicit ensembling if the protocol does not normalize the number of branches, verification calls, and tool invocations [35, 51].

A useful contrast is how coordination shares state. Some systems coordinate primarily through message passing (each agent maintains its own context), whereas others coordinate through shared tools or shared memory policies that enforce a common state and reduce divergence. These choices affect not only accuracy, but also failure propagation: shared-state designs can prevent inconsistent actions, but they can also amplify a single poisoned memory item across the group [59, 33].

Evaluation remains the main constraint. Coordination can look strong when the benchmark rewards diversity of attempts, and it can look weak when the benchmark penalizes any disagreement without modeling uncertainty. Protocol-aware benchmarks and testing suites therefore matter as much as coordination algorithms, because they determine whether the observed gains reflect genuine complementarity or accidental protocol fit [15, 33, 52, 57]. Many apparent coordination gains disappear once cost and disagreement are scored under a stricter protocol.

Domain-specific coordination examples illustrate that multi-agent benefits are often mediated by domain structure. In specialized settings (e.g., circuit optimization or geometry problem solving), coordination can represent decomposed subproblems and verification, but only if the communication format aligns with domain representations and constraints [25, 107].

A limitation of many multi-agent reports is accounting: improvements are often reported without clear cost normalization (number of agents, turns, tool calls), which makes it hard to distinguish algorithmic gains from brute-force search or ensembling. In contrast to single-agent settings where cost is often approximated by token count, coordination costs include cross-agent communication, redundant tool calls, and verification overhead, so a "better" system may be worse under realistic deployment budgets unless accounting is explicit [51, 5].

Multi-agent coordination is best treated as a protocol layer that sits above tool interfaces and planning loops. The most useful comparisons specify roles, communication bandwidth, aggregation rules, and cost normalization, so that coordination gains can be interpreted as decision-relevant trade-offs rather than as artifacts of a particular prompting or infrastructure setup [72, 87, 14, 59]. This highlights why coordination results often fail to transfer: the coordination contract, not the base model, is what changes between implementations.

6 Evaluation & Risks

Evaluation is where the agent field either becomes cumulative or collapses into irreconcilable demos. Benchmarks and protocols determine what "success" means, which costs are counted, and which assumptions are comparable across papers; risk evaluations further determine whether an agent is merely capable or plausibly deployable under realistic threat models [62, 22].

We first map the benchmark landscape and the protocol details that most often drive variance (tool catalogs, budgets, environment stochasticity, replayability). We then survey safety, security, and governance work, focusing on tool-chain vulnerabilities and monitoring strategies that require end-to-end evaluation rather than isolated model checks [102, 109, 39].

6.1 Benchmarks and evaluation protocols

Agent evaluation is hard for the same reason agent design is interesting: success depends on a coupled protocol of tools, environments, budgets, and stopping rules rather than on a single model forward pass. Benchmark scores can be high-signal only when the protocol details that govern comparability are made explicit and stable across studies [62, 37, 105].

One useful way to organize the benchmark landscape is by what is held fixed: some suites fix the environment and vary the agent scaffold, while others fix the scaffold and vary tool catalogs

or task distributions. Multi-domain suites make protocol dependence visible by spanning very different action spaces, highlighting where an agent is robust versus where it is simply well-matched to one environment's affordances [71, 84].

Another axis is openness. Some benchmarks assume a closed world (fixed tool catalogs and constrained environments), whereas others move toward open-web or open-ended settings where tools, observations, and intermediate states are less controlled. This difference changes what is being measured: closed settings emphasize interface compliance and orchestration reliability, while open settings emphasize error recovery and robustness under distribution shift [56, 8, 84]. This is one reason cross-benchmark comparisons are often fragile even when the underlying model family is held constant.

Protocol details often dominate variance. Tool access (which APIs exist), budget models (tokens, latency, monetary cost), and retry policies can each change outcomes without any change in the underlying model. Benchmarks that model noisy tool outputs and dynamic environments therefore provide a stronger substrate for claims about deployable agents than idealized API settings [43, 73, 60, 13]. When protocols omit these details, comparisons implicitly reward systems that exploit the evaluation blind spot rather than systems that are genuinely more reliable.

Whereas idealized API settings can make success look like pure model capability, noisy and dynamic settings often reveal that "agent performance" is a joint property of tool mediation, error handling, and accounting. In contrast to single-number leaderboards, protocol-aware reporting (budgets, retries, tool formats, and stochasticity) makes it possible to explain why two systems disagree and what would need to be held fixed to reproduce a claim [84, 43].

Security benchmarks reinforce this point by extending the protocol to include threat models. End-to-end suites that stress the full tool-use pipeline (planning, invocation, response handling) show that agents can be simultaneously "capable" and highly vulnerable, and that attack success rates depend on protocol details such as tool formats and environment assumptions [22, 73]. Security evaluations also surface failure modes (data exfiltration, unsafe tool invocation) that are orthogonal to task success, so they require explicit metrics rather than narrative claims.

Adversarial settings also reveal a second confound: agents may appear robust on primary tasks while leaking sensitive information or violating policies. Evaluations that explicitly measure privacy leakage or attack success under realistic tool-use scenarios therefore complement task-success metrics and help prevent "silent failures" from being treated as benign edge cases [61, 22].

Benchmarking effort increasingly includes meta-evaluation: measuring how evaluation itself behaves under changes in prompts, tool schemas, and scoring rules. This line of work is essential for survey synthesis because it distinguishes methodological variance from genuine capability differences, especially in fast-moving settings where models and scaffolds change quickly [62, 84, 2].

Agent benchmarks also vary in the degree to which they probe system-level behaviors (coordination, memory, recovery) rather than isolated skills. Suites that explicitly characterize agents as systems, rather than as single-turn predictors, make it easier to evaluate long-horizon behavior and to connect benchmark outcomes to design decisions in earlier chapters [56, 8].

Limitations remain substantial. Many benchmarks under-specify costs and protocol details, making it hard to reproduce results or to interpret improvements as anything beyond benchmark fit. Even when evaluation is careful, reporting can omit the accounting needed for fair comparison (e.g., tool calls, retries, failure recovery), which complicates synthesis across papers [19, 37].

The most reusable contribution of an evaluation paper is often not a single leaderboard number, but a protocol: a clear description of tasks, metrics, constraints, and threat models that makes results comparable. A recurring implication across evaluation work is that protocol details should be treated as first-class evidence: score improvements are meaningful when they

hold under comparable budgets, tool access, and threat models, whereas improvements that rely on hidden protocol changes are better interpreted as benchmark fit [62, 43, 22, 50, 26, 9]. A practical review heuristic follows: if a result cannot be reproduced from the protocol description alone, it should be treated as an implementation-specific finding.

Evaluation protocols define what success means, yet the same interface and budget assumptions also determine what can go wrong in deployment, motivating a closer look at safety, security, and governance.

6.2 Safety, security, and governance

Tool use turns language models into systems that can act on external resources, which expands the threat model from "unsafe text" to end-to-end tool-chain behavior. A central tension is that interface contracts therefore define both the attack surface and the monitoring leverage available to defenders [102, 22, 70, 68].

A practical entry point is a threat taxonomy. Benchmarks that systematize attack categories (e.g., prompt injection embedded in tool descriptions, out-of-scope parameter requests, tool-transfer exploits) illustrate that many failures arise from mismatched assumptions about who controls tool metadata, what inputs are trusted, and how tool errors are handled [102, 106].

This taxonomy lens also clarifies where defenses should intervene. Some failures are model-centric (e.g., susceptibility to instruction hijacking), whereas others are system-centric (e.g., unsafe parameter binding, privilege escalation through tool permissions, or replayable tool-trace abuse). This highlights that, as a result, evaluations can conflate "model safety" with "system safety" and obscure which layer actually needs redesign [102, 24].

Security benchmarks also emphasize scale and protocol realism. Suites that map large numbers of attack tasks to common weakness categories and implement tools in multiple formats make it harder to dismiss failures as contrived edge cases, and they stress the full pipeline rather than isolated model prompts [22, 102]. Reported suites quantify this realism by enumerating large attack sets (e.g., thousands of attack tasks across dozens of test cases) and by measuring not only attack success, but also how attacks degrade task completion under the same nominal agent scaffold [22]. Complementary MCP-specific suites organize attacks at the interface layer (e.g., tool-description prompt injection, tool-transfer, and mixed strategies), which helps link failures back to concrete contract assumptions rather than to generic "jailbreak" narratives [102].

Monitoring and red-teaming workflows highlight a second constraint: defenses are themselves protocol-dependent. If the monitor lacks situational awareness, or if the adversary can exploit tool outputs and intermediate traces, then guardrails that look strong in controlled settings can fail in deployment-like loops [39, 22].

Architecture choices can measurably shift which vulnerabilities dominate. Comparative evaluations of function-calling style tool mediation versus protocol-based tool interfaces suggest that different tool mediation layers can change attack success rates and the balance between system-centric and model-centric exposures, especially under chained attacks [24, 102].

Behavioral safety issues also include seemingly "benign" failure modes such as quitting or refusal behaviors that can be triggered by adversarial prompts or tool failures. Systematic evaluation of such behaviors matters because they can interact with recovery policies and make long-horizon agents brittle even when single-step safety checks pass [4, 19].

Governance and enforcement mechanisms often reduce to permissions and policy compliance at the interface layer: what the agent is allowed to call, what parameters are admissible, and how violations are detected and handled. These controls are most effective when they are enforced by the tool contract rather than by post-hoc prompt policies alone [40, 20]. This makes it important to name the enforcement points and permission assumptions explicitly, since they determine what can be monitored and what can be blocked.

This means separating ”monitoring” from ”enforcement”. Monitoring can detect anomalous tool traces and suspicious intermediate states, whereas enforcement constrains what actions are executable in the first place (least privilege, schema validation, parameter guards). Robust governance typically needs both: monitoring without enforcement is reactive, while enforcement without monitoring can be bypassed if the threat model includes tool-chain manipulation [39, 40].

Adversarial evaluation further shows that ”safe” behavior cannot be inferred from task success. Attacks can preserve primary task completion while leaking data or violating constraints, so evaluations must include explicit security metrics and threat-model reporting alongside utility metrics [85, 7, 53].

Security and governance are best treated as protocol design problems. The most reusable results specify the threat model, the tool interface assumptions, and the monitoring/enforcement points in the loop, so that defenses can be compared as systems rather than as isolated prompt patches [102, 22, 39]. Put differently, security claims are interpretable only relative to an explicit threat model and a concrete tool contract.

7 Discussion

Protocol drift is the most common hidden confound in agent results. Across the literature, ”tool use” can mean different action representations, tool catalogs, retry rules, and observability assumptions; ”success” can mean different stopping conditions and cost models; and ”robustness” can mean different threat models. Without a shared protocol surface, even careful ablations can fail to transfer, because the claim is anchored to an implicit contract rather than to a comparable evaluation setting [62, 58, 42].

This also reframes how to read progress on planning, memory, and adaptation. Deliberative prompting and reflection can look like algorithmic improvements, but their benefit often depends on budget and termination policies; similarly, retrieval and memory can look like capability gains, but much of the variance comes from what evidence is retrievable and how it is cited or validated in the loop [95, 75, 79]. A practical implication is that future benchmarks should report key protocol details (tool access, budget, environment stochasticity) as first-class metadata, so that comparisons can be interpreted as ”under the same contract” rather than ”under similar intent.”

Security results reinforce the same point. Many attacks operate at the tool boundary: prompt injection in tool descriptions, unsafe parameter requests, or tool-transfer exploits. These failures can be invisible if evaluation only measures end-task completion, since a system can remain functional while leaking data or violating policies. End-to-end benchmarks that exercise the full tool-use pipeline therefore provide a stronger substrate for claims about deployable agents than isolated safety checks [22, 102, 39].

Finally, tables and taxonomies are most valuable when they compress decisions, not when they enumerate papers. We therefore treat the Appendix as a protocol map: representative approaches are grouped by their loop and interface assumptions, and benchmarks are grouped by the task, metric, and protocol constraints that govern comparability. This design is intended to support reuse: practitioners can locate the relevant protocol assumptions first, then read the associated sections as decision-relevant contrasts rather than as a narrative catalog.

8 Conclusion

LLM agents are best understood as closed-loop systems whose behavior is jointly determined by the model, the interface contract, and the evaluation protocol. Across tool use, planning, memory, adaptation, and multi-agent coordination, the most persistent source of disagreement

Table 1: Representative agent designs (loop + interface assumptions).

Work	Core idea	Loop and interface assumption	Key refs
AgentSwift	Automated agent discovery across diverse tasks	Closed-loop agent evaluated across multiple benchmark families	[49]
AgentSquare	Unified evaluation across web, embodied, tool, and game settings	Environment/task suite treated as a shared action-space substrate	[71]
ReAct	Interleave reasoning traces with actions	Prompted loop that alternates deliberation and tool/environment steps	[94]
Structured Cognitive Loop (SCL)	Modularize cognition into explicit phases	Loop decomposed into named stages (e.g., retrieval → reasoning → action)	[44]
Progent	Defense wrapper for tool-using agents	Plugs into existing agents without changing internals (minimal integration)	[73]
Agent Distillation	Transfer full agent behavior to smaller models	Student learns task-solving behavior beyond single-step reasoning traces	[41]
SAFE (fact-checking)	Retrieval-augmented agent for long-form verification	Loop couples retrieval with structured extraction/evaluation steps	[34]
Self-Challenging (tool use)	Stress-test + improve tool-use robustness	Iterative interaction where the agent generates and resolves harder tool calls	[112]
MemTool	Autonomous memory management for tool use	Memory operations are explicit actions; evaluated over long interaction traces	[59]
SkyRL-Agent	Multi-turn, long-horizon agent training/evaluation	Training and evaluation framed as long-horizon interaction episodes	[5]

in the literature is not whether a technique "works", but under which protocol assumptions the reported gains are interpretable and transferable [94, 62, 58].

Looking forward, progress on agents will be easier to audit if papers make key protocol details explicit (tool catalog, budget/cost model, environment stochasticity, threat model) and if benchmarks evaluate the full tool-use pipeline rather than isolated components. A protocol-first reporting norm would narrow the gap between impressive demos and deployable systems, and make cross-paper synthesis a structural property of the field rather than an after-the-fact editorial exercise [22, 102, 42].

A Tables

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.

Table 2: Benchmarks and evaluation protocols (task + metric + constraints).

Benchmark / setting	Task + metric (short)	Key protocol constraints	Key refs
MCPAgentBench	MCP tool-use capability; task success	Real-world MCP definitions; tool schema fidelity	[58]
ETOM	Multi-hop end-to-end tool orchestration; success rate	Hierarchical MCP ecosystem; multi-step orchestration	[16]
ToolBench	Tool-use tasks; task completion	Tool availability + API behavior; model/tool mismatch	[109]
BFCL	Function-calling/tool-use; correctness	Long trajectories possible; cost/energy sensitivity under attacks	[109]
M3ToolEval	Multi-turn tool use; task success	Multi-turn interaction; tool calling under constraints	[112]
TauBench	Multi-turn tool use; task success	Multi-turn protocol; robustness across turns	[112]
ScaleMCP	Memory management in tool use; accuracy + tool removal ratio	Long interaction horizon; explicit memory operations	[59]
AgentDojo, ASB, and AgentPoison	Agent security; attack success vs utility	Threat-model assumptions; defense integration cost	[73]
RAS-Eval	Safety attack suite; robustness under attacks	CWE-mapped attack categories; tool-implemented adversaries	[22]
MSB (MCP Security Benchmark)	MCP-specific attacks; resistance	Name-collision/prompt-injection style attacks via tool metadata	[102]

[2] Kushal Agrawal, Frank Xiao, Guido Bergman, and Asa Cooper Stickland. Why do language model agents whistleblow? *arXiv preprint arXiv:2511.17085v2*, 2025. URL http://arxiv.org/abs/2511.17085v2.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] Binger Chen, Tacettin Emre Bök, Behnood Rasti, Volker Markl, and Begüm Demir.

Remsa: An llm agent for foundation model selection in remote sensing. *arXiv preprint arXiv:2511.17442v1*, 2025. URL http://arxiv.org/abs/2511.17442v1.

[8] 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.

[9] Jizhou Chen and Samuel Lee Cong. Agentguard: Repurposing agentic orchestrator for safety evaluation of tool orchestration. *arXiv preprint arXiv:2502.09809v1*, 2025. URL http://arxiv.org/abs/2502.09809v1.

[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] 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.

[12] 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.

[13] Enfang Cui, Yujun Cheng, Rui She, Dan Liu, Zhiyuan Liang, Minxin Guo, Tianzheng Li, Qian Wei, Wenjuan Xing, and Zhijie Zhong. Agentdns: A root domain naming system for llm agents. *arXiv preprint arXiv:2505.22368v1*, 2025. URL http://arxiv.org/abs/2505.22368v1.

[14] Zikun Cui, Tianyi Huang, Chia-En Chiang, and Cuiqianhe Du. Toward verifiable misinformation detection: A multi-tool llm agent framework. *arXiv preprint arXiv:2508.03092v1*, 2025. URL http://arxiv.org/abs/2508.03092v1.

[15] 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.

[16] 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.

[17] 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.

[18] Andreas Einwiller, Kanishka Ghosh Dastidar, Artur Romazanov, Annette Hautli-Janisz, Michael Granitzer, and Florian Lemmerich. Benevolent dictators? on llm agent behavior in dictator games. *arXiv preprint arXiv:2511.08721v1*, 2025. URL http://arxiv.org/abs/2511.08721v1.

[19] Gal Engelberg, Konstantin Koutsyi, Leon Goldberg, Reuven Elezra, Idan Pinto, Tal Moalem, Shmuel Cohen, and Yoni Weintrob. Sola-visibility-ispm: Benchmarking agentic ai for identity security posture management visibility. *arXiv preprint arXiv:2601.07880v1*, 2026. URL http://arxiv.org/abs/2601.07880v1.

[20] Junfeng Fang, Zijun Yao, Ruipeng Wang, Haokai Ma, Xiang Wang, and Tat-Seng Chua. We should identify and mitigate third-party safety risks in mcp-powered agent systems. *arXiv preprint arXiv:2506.13666v1*, 2025. URL http://arxiv.org/abs/2506.13666v1.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] Yuzhi Hao and Danyang Xie. A multi-llm-agent-based framework for economic and public policy analysis. *arXiv preprint arXiv:2502.16879v1*, 2025. URL http://arxiv.org/abs/2502.16879v1.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] Ziniu Hu, Ahmet Iscen, Chen Sun, Kai-Wei Chang, Yizhou Sun, David A Ross, Cordelia Schmid, and Alireza Fathi. Avis: Autonomous visual information seeking with large language model agent. *arXiv preprint arXiv:2306.08129v3*, 2023. URL http://arxiv.org/abs/2306.08129v3.

[34] 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.

[35] Yuxiang Ji, Ziyu Ma, Yong Wang, Guanhua Chen, Xiangxiang Chu, and Liaoni Wu. Tree search for llm agent reinforcement learning. *arXiv preprint arXiv:2509.21240v2*, 2025. URL http://arxiv.org/abs/2509.21240v2.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Minki Kang, Jongwon Jeong, Seanie Lee, Jaewoong Cho, and Sung Ju Hwang. Distilling llm agent into small models with retrieval and code tools. *arXiv preprint arXiv:2505.17612v2*, 2025. URL http://arxiv.org/abs/2505.17612v2.

[42] Raffi Khatchadourian. Replayable financial agents: A determinism-faithfulness assurance harness for tool-using llm agents. *arXiv preprint arXiv:2601.15322v1*, 2026. URL http://arxiv.org/abs/2601.15322v1.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] Jia Li, Xianjie Shi, Kechi Zhang, Ge Li, Zhi Jin, Lei Li, Huangzhao Zhang, Jia Li, Fang Liu, Yuwei Zhang, Zhengwei Tao, Yihong Dong, Yuqi Zhu, and Chongyang Tao. Graphcodeagent: Dual graph-guided llm agent for retrieval-augmented repo-level code generation. *arXiv preprint arXiv:2504.10046v2*, 2025. URL http://arxiv.org/abs/2504.10046v2.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] Yuxuan Li, Sauvik Das, and Hirokazu Shirado. What makes llm agent simulations useful for policy? insights from an iterative design engagement in emergency preparedness. *arXiv preprint arXiv:2509.21868v1*, 2025. URL http://arxiv.org/abs/2509.21868v1.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] Tianming Liu, Jirong Yang, Yafeng Yin, Manzi Li, Linghao Wang, and Zheng Zhu. Aligning llm agents with human learning and adjustment behavior: a dual agent approach. *arXiv preprint arXiv:2511.00993v1*, 2025. URL http://arxiv.org/abs/2511.00993v1.

[58] Wenrui Liu, Zixiang Liu, Elsie Dai, Wenhan Yu, Lei Yu, Tong Yang, Jinjun Han, and Hong Gao. Mcpagentbench: A real-world task benchmark for evaluating llm agent mcp tool use. *arXiv preprint arXiv:2512.24565v3*, 2025. URL http://arxiv.org/abs/2512.24565v3.

[59] 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.

[60] 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.

[61] Kanghua Mo, Li Hu, Yucheng Long, and Zhihao Li. Attractive metadata attack: Inducing llm agents to invoke malicious tools. *arXiv preprint arXiv:2508.02110v2*, 2025. URL http://arxiv.org/abs/2508.02110v2.

[62] 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.

[63] 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.

[64] 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.

[65] Humza Nusrat, Luke Francisco, Bing Luo, Hassan Bagher-Ebadian, Joshua Kim, Karen Chin-Snyder, Salim Siddiqui, Mira Shah, Eric Mellon, Mohammad Ghassemi, Anthony Doemer, Benjamin Movsas, and Kundan Thind. Automated stereotactic radiosurgery planning using a human-in-the-loop reasoning large language model agent. *arXiv preprint arXiv:2512.20586v1*, 2025. URL http://arxiv.org/abs/2512.20586v1.

[66] 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.

[67] Hoang Pham, Thuy-Duong Nguyen, and Khac-Hoai Nam Bui. Agent-unirag: A trainable open-source llm agent framework for unified retrieval-augmented generation systems. *arXiv preprint arXiv:2505.22571v3*, 2025. URL http://arxiv.org/abs/2505.22571v3.

[68] Ron F. Del Rosario, Klaudia Krawiecka, and Christian Schroeder de Witt. Architecting resilient llm agents: A guide to secure plan-then-execute implementations. *arXiv preprint arXiv:2509.08646v1*, 2025. URL http://arxiv.org/abs/2509.08646v1.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Jieyu Zhang, Hang Wu, Yuanda Zhu, Joyce Ho, Carl Yang, and May D. Wang. Ehragent: Code empowers large language models for few-shot complex tabular reasoning on electronic health records. *arXiv preprint arXiv:2401.07128v3*, 2024. URL http://arxiv.org/abs/2401.07128v3.

[75] 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.

[76] Abdelrahman Soliman, Ahmed Refaey, Aiman Erbad, and Amr Mohamed. Intagent: Nwdaf-based intent llm agent towards advanced next generation networks. *arXiv preprint arXiv:2601.13114v1*, 2026. URL http://arxiv.org/abs/2601.13114v1.

[77] 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.

[78] Dehao Tao, Guoliang Ma, Yongfeng Huang, and Minghu Jiang. Membox: Weaving topic continuity into long-range memory for llm agents. *arXiv preprint arXiv:2601.03785v2*, 2026. URL http://arxiv.org/abs/2601.03785v2.

[79] 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.

[80] Elizaveta Tennant, Stephen Hailes, and Mirco Musolesi. Moral alignment for llm agents. *arXiv preprint arXiv:2410.01639v4*, 2024. URL http://arxiv.org/abs/2410.01639v4.

[81] 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.

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

[83] 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.

[84] Wenhao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, and Siheng Chen. Mcp-flow: Facilitating llm agents to master real-world, diverse and scaling mcp tools. *arXiv preprint arXiv:2510.24284v2*, 2025. URL http://arxiv.org/abs/2510.24284v2.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N. Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *arXiv preprint arXiv:2406.11200v3*, 2024. URL http://arxiv.org/abs/2406.11200v3.

[90] 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.

[91] 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.

[92] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110v11*, 2025. URL http://arxiv.org/abs/2502.12110v11.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] 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.

[100] Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. Large language model-brained gui agents: A survey. *arXiv preprint arXiv:2411.18279v12*, 2024. URL http://arxiv.org/abs/2411.18279v12.

[101] Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. Datascibench: An llm agent benchmark for data science. *arXiv preprint arXiv:2502.13897v1*, 2025. URL http://arxiv.org/abs/2502.13897v1.

[102] 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.

[103] 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.

[104] 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.

[105] Zehua Zhang, Ati Priya Bajaj, Divij Handa, Siyu Liu, Arvind S Raj, Hongkai Chen, Hulin Wang, Yibo Liu, Zion Leonahenahe Basque, Souradip Nath, Vishal Juneja, Nikhil Chapre, Yan Shoshitaishvili, Adam Doupé, Chitta Baral, and Ruoyu Wang. Buildbench: Benchmarking llm agents on compiling real-world open-source software. *arXiv preprint arXiv:2509.25248v1*, 2025. URL http://arxiv.org/abs/2509.25248v1.

[106] 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.

[107] 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.

[108] Yuheng Zhao, Junjie Wang, Linbin Xiang, Xiaowen Zhang, Zifei Guo, Cagatay Turkay, Yu Zhang, and Siming Chen. Lightva: Lightweight visual analytics with llm agent-based task planning and execution. *arXiv preprint arXiv:2411.05651v2*, 2024. URL http://arxiv.org/abs/2411.05651v2.

[109] Kaiyu Zhou, Yongsen Zheng, Yicheng He, Meng Xue, Xueluan Gong, Yuji Wang, and Kwok-Yan Lam. Beyond max tokens: Stealthy resource amplification via tool calling chains in llm agents. *arXiv preprint arXiv:2601.10955v1*, 2026. URL http://arxiv.org/abs/2601.10955v1.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

[114] Yakun Zhu, Shaohang Wei, Xu Wang, Kui Xue, Xiaofan Zhang, and Shaoting Zhang. Menti: Bridging medical calculator and llm agent with nested tool calling. *arXiv preprint arXiv:2410.13610v3*, 2024. URL http://arxiv.org/abs/2410.13610v3.