Reliability Patterns for LLM-Based Multi-Agent Systems: A Synthesis of Published Evidence

Linus Teklenburg
Independent Researcher
hey@linus-teklenburg.de

Abstract

Large language model agents are increasingly deployed in multi-step, tool-bound configurations, yet practical guidance on how to make such systems reliable remains fragmented across conference papers, framework documentation, and engineering blogs. This paper synthesises the current state of published evidence for six engineering patterns commonly invoked to improve reliability in language-model-based multi-agent systems: structured tool schemas, explicit validator agents, retry with structured reflection, least-privilege tool access, explicit failure propagation, and deterministic output formatting. Each pattern is mapped to the strongest available controlled comparisons, benchmark results, or system ablations in the academic literature, and evidence is graded as strong, moderate, weak, or absent. The synthesis finds that three patterns-structured output enforcement, verifier- or judge-in-the-loop validation, and reflection with an external success signal-are supported by multiple independent controlled comparisons. Three further patterns are widely recommended in framework documentation but are supported primarily by system-paper ablations rather than by controlled isolation studies. A recurring cross-pattern finding is that reliability gains depend on the presence of an external grounding signal; patterns that rely purely on self-assessment by the same generator tend to regress on reasoning-heavy tasks. Gaps are enumerated, including the near-total absence of controlled studies on pattern interaction effects and on cross-model generalisation. The paper concludes with a graded set of recommendations and open research questions.

Keywords: large language model agents; multi-agent systems; Model Context Protocol; reliability; tool use; structured output; evaluation

1. Introduction

Language-model-based agents are software systems in which a large language model (LLM) decides, on each step, which external tool to invoke, what argument to pass, and how to incorporate the returned observation into its ongoing reasoning (Yao et al., 2022; Schick et al.,

2023; Xi et al., 2023). In the last two years such agents have moved from research prototypes into production deployments across software engineering, customer operations, scientific workflows, and enterprise automation, with frameworks such as the Model Context Protocol, the OpenAI Agents SDK, LangGraph, and AutoGen providing standardised interfaces for tool binding and inter-agent coordination (Anthropic, 2024b; OpenAI, 2025; LangChain, 2024; Wu et al., 2023).

The engineering community has responded with a rapidly growing body of advice on how to make these systems reliable. Vendor guidance recommends structured tool schemas, role-specialised agents, explicit validators, retry loops with reflection, and tight scoping of tool permissions (Anthropic, 2024a, 2024b). Conference papers have proposed individual patterns and demonstrated them on benchmarks (Shinn et al., 2023; Madaan et al., 2023; Gou et al., 2023; Wu et al., 2023). Framework documentation codifies these patterns as reusable abstractions (LangChain, 2024; OpenAI, 2025). The resulting literature, however, is difficult to navigate: individual patterns are described in isolation, evidence quality varies, and there is no single source that maps each commonly recommended pattern to the controlled studies that actually support it.

This paper offers such a synthesis. It identifies six reliability-relevant engineering patterns for LLM-based multi-agent systems, catalogues the strongest published evidence for each, and grades that evidence on a four-level scale: strong, moderate, weak, or absent. The contribution is not a new pattern, a new benchmark, or a new experimental result. The contribution is a consolidated evidence map and an honest identification of where practitioner recommendations are empirically supported and where they currently outrun the published record.

The paper proceeds as follows. Section 2 provides a compact background on LLM agents, the Model Context Protocol, and the reliability benchmarks that operationalise terms such as "task success" and "hallucination". Section 3 introduces the six-pattern taxonomy with its theoretical motivation. Section 4, the core of the paper, maps each pattern to its strongest published evidence and grades that evidence. Section 5 discusses how the patterns compose across three canonical architectures: single-agent-with-tools, sequential multi-agent, and parallel multi-agent with shared context. Section 6 enumerates the most important evidence gaps. Section 7 offers a graded set of recommendations together with open research questions. Section 8 concludes.

Two scope boundaries should be stated. First, the paper covers language-model agents that invoke external tools via structured protocols; it does not cover classical symbolic multi-agent systems, reinforcement-learning agents trained end-to-end, or pure chatbot deployments without tool use. Second, "reliability" here refers to the narrow, task-level sense in which the benchmark literature uses the term: success rate on bounded tasks, rate of tool-call correctness, resistance to hallucination, and robustness under repetition. Broader notions of reliability that include long-horizon drift, safety, alignment, and adversarial robustness are touched on only where the evidence explicitly intersects.

2. Background

2.1 LLM agents and tool binding

An LLM agent is a control loop in which a language model is prompted with the state of a task, produces a tool call or a final answer, and, on a tool call, receives the tool's output and is re-prompted (Yao et al., 2022; Xi et al., 2023; L. Wang et al., 2023). Tool binding is the mechanism by which the model is told which tools are available, what each one does, and how to call it. Early work in this line, including ReAct (Yao et al., 2022) and Toolformer (Schick et al., 2023), demonstrated that a model can interleave reasoning traces with tool invocations and achieve substantial gains over reasoning-only baselines.

Multi-agent configurations add one further dimension. Instead of a single model handling every step, multiple agents-each with a distinct role, tool budget, or instruction-coordinate over a shared task. Canonical roles include planner, executor, critic, and validator (Wu et al., 2023; Li, Hammoud, Itani, Khizbullin & Ghanem, 2023). Communication can be sequential (one agent produces output consumed by the next), hierarchical (a supervisor dispatches to workers), or parallel (multiple agents operate concurrently on shared state).

The Model Context Protocol (MCP) is a recent standard for tool binding that specifies a JSON-RPC based transport and three primitives: model-controlled tools, application-controlled resources, and user-controlled prompts (Anthropic, 2024b). MCP and its peers (the OpenAI Agents SDK, LangGraph, AutoGen) do not themselves guarantee reliability; they provide the scaffolding within which reliability patterns are implemented.

2.2 Operational definitions of reliability

The agent-evaluation literature has converged on several quantitative notions of reliability. *Task success rate* is the fraction of benchmark tasks on which the agent reaches the correct terminal state; this is the dominant metric in SWE-bench (Jimenez et al., 2023; OpenAI, 2024), GAIA (Mialon et al., 2023), AgentBench (Liu et al., 2023), WebArena (Zhou et al., 2023), VisualWebArena (Koh et al., 2024), and MLE-bench (Chan et al., 2024). *Tool-call correctness* measures whether the right tool is selected with the right arguments; ToolBench and its evaluation script operationalise this directly (Qin et al., 2023). *Hallucination rate* quantifies the frequency of unfounded factual claims, typically with reference to a grounded source (Ji et al., 2023; L. Huang et al., 2023; Zhang et al., 2023).

A complementary notion, introduced by τ-bench (Yao, Shinn, Razdaibiedina & Narasimhan, 2024), is *reliability under repetition*. The τ-bench authors report the metric pass^k, the probability that an agent solves the same task correctly k times in a row. On the retail subset, frontier models achieve pass^1 below 50% and pass^8 below 25%-a direct quantitative statement that, even when average accuracy is moderate, agent outputs are far from deterministically correct.

Absolute reliability numbers across these benchmarks are sobering. On GAIA, GPT-4 with

tool plugins achieves roughly 15% against a human baseline of 92% (Mialon et al., 2023). On WebArena the best GPT-4 agent achieves 14.41% against a human baseline of 78.24% (Zhou et al., 2023). On VisualWebArena the best evaluated vision-language model achieves 16.4% against a human baseline of 88.7% (Koh et al., 2024). On MLE-bench, the best reported configuration obtains a Kaggle bronze medal or better in 16.9% of competitions (Chan et al., 2024). On SWE-bench Verified, GPT-4o resolves 33.2% of curated issues (OpenAI, 2024). These gaps motivate the practitioner search for reliability patterns: the baseline LLM agent, without engineering scaffolding, is unreliable on any realistic task distribution.

2.3 Failure-mode literature

Parallel to the benchmark literature, a body of work catalogues the ways in which agent systems fail. Cemri and colleagues introduce the Multi-Agent System Failure Taxonomy (MAST) based on 1,600 annotated execution traces across seven multi-agent frameworks, identifying fourteen failure modes in three categories: specification issues, inter-agent misalignment, and task-verification failures, with an inter-annotator agreement of $\kappa = 0.88$ (Cemri et al., 2025). The hallucination surveys provide taxonomies of the factuality failure mode at a lower level of abstraction (Ji et al., 2023; L. Huang et al., 2023; Zhang et al., 2023). Together, this literature supplies both the vocabulary and the empirical grounding for Section 4's evidence mapping.

3. A Taxonomy of Reliability Patterns

The six patterns below were selected by scanning the framework documentation of MCP, the OpenAI Agents SDK, LangGraph, and AutoGen (Anthropic, 2024b; OpenAI, 2025; LangChain, 2024; Wu et al., 2023), the Anthropic engineering note on effective agents (Anthropic, 2024a), and the system-paper and survey literature on LLM agents (Xi et al., 2023; L. Wang et al., 2023). Patterns that occur in at least three of these sources, and for which at least one controlled comparison exists in the peer-reviewed or arXiv-indexed literature, were retained. Patterns that are ubiquitous in blog posts but have no academic footprint (for example, "add more examples to the prompt") were excluded.

3.1 Pattern 1: Structured tool schemas with strict validation

The first pattern prescribes that every tool a model can call should be declared through a typed schema-JSON Schema, a function signature, or an equivalent grammar-and that the model's output when invoking the tool should be validated against this schema before execution. The motivation is twofold. First, a typed schema disambiguates what a valid call looks like, reducing the space of possible hallucinated invocations. Second, validation at the boundary prevents malformed calls from reaching the tool implementation. The pattern is foundational in Toolformer (Schick et al., 2023), ToolLLM (Qin et al., 2023), and Gorilla (Patil, Zhang, Wang &

Gonzalez, 2023), and is the default interface in every major production framework (Anthropic, 2024b; OpenAI, 2025; LangChain, 2024).

3.2 Pattern 2: Explicit validator agents in the execution chain

The second pattern inserts a dedicated validation step between the generator's output and the downstream action. The validator may be a separate model call, a different role played by the same model, an external tool (for instance, a unit-test runner or a type checker), or a majority vote over samples. The theoretical motivation traces to verifier-reranking in mathematical reasoning (Cobbe et al., 2021) and self-consistency (X. Wang et al., 2022), and is instantiated in modern form by LLM-as-a-judge (Zheng et al., 2023), CRITIC (Gou et al., 2023), and the planner–executor–critic configurations documented in AutoGen and CAMEL (Wu et al., 2023; Li et al., 2023).

3.3 Pattern 3: Retry with structured reflection

The third pattern replaces naive resampling on failure with an intermediate reflection step in which the model is prompted to analyse what went wrong and to store the result in a working memory that informs the next attempt. Reflexion (Shinn et al., 2023) is the canonical instance. Related work includes Self-Refine (Madaan et al., 2023), Self-Debug (Chen, Lin, Schärli & Zhou, 2023), and Recursive Criticism and Improvement (Kim, Baldi & McAleer, 2023). The pattern's claimed advantage over naive retry is that the reflection transforms the memory of the failure into a conditioning signal for subsequent generation, rather than merely redrawing from the same distribution.

3.4 Pattern 4: Least-privilege tool access scoping

The fourth pattern limits the set of tools available to each agent role to the smallest set the role requires. The motivation has two distinct strands. The accuracy strand is that LLM tool selection degrades as the candidate tool menu grows, producing an accuracy argument for minimising the visible tool graph (Qin et al., 2023; Patil et al., 2023; Y. Huang et al., 2023; Lu et al., 2024). The security strand is that, even where accuracy would not suffer, exposing unnecessary tools enlarges the attack surface and makes the agent vulnerable to prompt-injection-driven misuse of tools that should never have been reachable.

3.5 Pattern 5: Explicit failure propagation and handling

The fifth pattern requires that when an agent or tool fails-through a thrown exception, a returned error code, a schema-invalid output, or a semantic failure detected downstream-the failure be propagated as a typed, structured signal rather than silently swallowed. The downstream agent

or controller is then expected to handle the failure explicitly: by retry, by escalation, by switching strategy, or by terminating with a clear error. Cemri and colleagues show, via the MAST taxonomy, that silent or miscategorised failures are a dominant cause of cascaded breakdowns in multi-agent systems (Cemri et al., 2025).

3.6 Pattern 6: Deterministic output formatting

The sixth pattern constrains the model's final output to a deterministic format, typically via grammar-constrained decoding, finite-state-machine masking, or a library such as Outlines, LMQL, or XGrammar (Willard & Louf, 2023; Beurer-Kellner, Fischer & Vechev, 2023; Dong et al., 2024; Geng, Josifoski, Peyrard & West, 2023; Geng, Josifoski, Peyrard, West et al., 2025). The motivation is that downstream consumers of agent output-tools, databases, user interfaces-require a parseable format, and that grammar-constrained generation achieves format compliance by construction rather than by prayer.

Patterns 1 and 6 overlap but are not identical. Pattern 1 concerns the schema of tool invocations during execution; Pattern 6 concerns the format of final or intermediate outputs more broadly. Both rely on grammar-level enforcement, but the theoretical and empirical literatures on them differ, so they are treated separately.

4. Published Evidence per Pattern

This section is the core of the paper. For each of the six patterns it reports the strongest available controlled evidence, extracts effect sizes where these are reported, and notes conflicting findings where they exist. Table 1 summarises the evidence grade and the primary sources at the end of the section.

4.1 Pattern 1: Structured tool schemas with strict validation

Evidence grade: **moderate to strong**, with a documented trade-off against reasoning-heavy tasks.

The benefits of structured tool schemas divide into two measurable effects. The first is structural compliance: the rate at which model outputs parse under the specified schema. The second is downstream task success: the rate at which a correctly parsed call also semantically succeeds.

On structural compliance, the evidence is uniformly strong. Geng and colleagues benchmark six structured-decoding frameworks (Guidance, Outlines, llama.cpp, XGrammar, and the OpenAI and Gemini structured-output APIs) on ten thousand real-world JSON schemas and report near-100% schema conformance for constrained-decoding approaches against 60–80% for unconstrained prompting (Geng et al., 2025). Willard and Louf show that finite-state-machine-guided decoding achieves 100% compliance by construction with near-constant per-token overhead

(Willard & Louf, 2023). Dong and colleagues report 100% structural validity on comparable schemas for XGrammar with up to two orders of magnitude lower overhead than prior grammar-constrained engines (Dong et al., 2024). These results establish that the compliance benefit of Pattern 1 is essentially free in engineering terms.

On downstream task success the evidence is also positive but less uniform. Schick and colleagues show that a model trained to emit typed API calls more than doubles zero-shot performance on the Toolformer task suite and that a 6.7B model so trained matches a 175B model without typed calls (Schick et al., 2023). Qin and colleagues show on ToolBench that fine-tuning with structured API descriptions and an evaluator over typed calls lifts open-model pass rates into parity with GPT-3.5 on unseen APIs (Qin et al., 2023). Patil and colleagues show that retrieval-aware fine-tuning on typed APIs reduces hallucinated API calls as the dominant failure mode on APIBench (Patil et al., 2023).

A critical conflicting result is reported by Tam and colleagues (Tam et al., 2024). In a controlled comparison across GSM8K, Last-Letter, and several classification tasks, strict JSON output requirements reduce reasoning accuracy by up to 27 percentage points on GSM8K relative to free-form reasoning followed by extraction. The degradation is monotonic in constraint strictness and holds across GPT-3.5-turbo, Claude, Gemini, and open-weight models. On classification the effect reverses: strict format sometimes improves accuracy. The implication for Pattern 1 is that structured schemas should be applied at the tool-invocation boundary and to the final output, but reasoning should be allowed to proceed in free text before the constrained emission step. This reconciliation is consistent with the two-step pattern recommended in the OpenAI structured-outputs documentation and the XML-tagged chain-of-thought guidance from Anthropic (OpenAI, 2025; Anthropic, 2024a).

4.2 Pattern 2: Explicit validator agents

Evidence grade: **strong**, with an important boundary condition.

The canonical controlled evidence for validator-in-chain predates the LLM era. Cobbe and colleagues show that a trained verifier reranking best-of-N samples raises GSM8K accuracy from roughly 27% to roughly 55%, a two-fold gain approximately equivalent in effect to a thirty-fold increase in model size (Cobbe et al., 2021). Wang and colleagues report that majority-vote self-consistency-a validator in the degenerate form of a voting ensemble-improves GSM8K by 17.9, SVAMP by 11.0, AQuA by 12.2, StrategyQA by 6.4, and ARC-c by 3.9 percentage points over greedy chain-of-thought (X. Wang et al., 2022).

In the modern LLM-agent era, Zheng and colleagues establish that GPT-4 as a judge reaches over 80% agreement with human expert evaluators on MT-Bench, an agreement rate equal to inter-human agreement (Zheng et al., 2023). Gou and colleagues, in CRITIC, show that a tool-grounded validator loop delivers consistent gains of several F1 points on ambiguous QA, improves program-synthesis pass rates, and reduces toxicity in generation (Gou et al., 2023). Wu and colleagues report that planner–executor–critic configurations in AutoGen out-

perform single-agent baselines on mathematics, coding, and QA, and document a GAIA result in which the AutoGen configuration places first across difficulty levels (Wu et al., 2023). Li and colleagues, in CAMEL, show qualitative gains from role-specialised cooperation though with weaker controlled effect sizes (Li et al., 2023).

The critical boundary condition is supplied by Huang and colleagues (J. Huang et al., 2023). They re-examine the claim that LLMs can reliably self-correct and show that when the validator is the same model as the generator, without access to an external signal, the self-correction loop often degrades performance relative to the first-attempt answer. The reported gains in earlier self-correction papers, they argue, depend on oracle signals (unit tests, ground-truth labels, tool output) being available to the validator. Valmeekam and colleagues reach a parallel conclusion on planning benchmarks: self-verification without an external planner is unreliable (Valmeekam, Marquez, Sreedharan & Kambhampati, 2023). The implication is precise: Pattern 2 yields reliability gains when the validator has access to an external grounding signal or is from a different distribution than the generator. Pure intrinsic-judge loops on the same model should not be relied upon.

4.3 Pattern 3: Retry with structured reflection

Evidence grade: **strong for tasks with an executable success signal; weak to moderate otherwise**.

Shinn and colleagues report the headline effect: Reflexion with GPT-4 raises HumanEval pass@1 from 80% to 91%, an 11-point gain; on AlfWorld the gain is 22 points; on HotpotQA it is approximately 20 points over ReAct (Shinn et al., 2023). Within-paper ablation shows that replacing structured reflection with naive retry recovers baseline performance, isolating the contribution of the reflection memory. Madaan and colleagues, in Self-Refine, demonstrate an average preference-gain of around 20% across seven tasks without parameter updates (Madaan et al., 2023). Chen and colleagues, in Self-Debug, report up to 12-percentage-point gains on TransCoder and MBPP and show that debugging with execution feedback can match the effect of a tenfold increase in sampling (Chen et al., 2023). Kim and colleagues, with Recursive Criticism and Improvement, exceed reinforcement-learning and behaviour-cloning baselines on MiniWoB++ with substantially fewer samples (Kim et al., 2023).

As with Pattern 2, the critical boundary is external grounding. Huang and colleagues demonstrate that on reasoning tasks without an oracle signal, reflection can degrade first-attempt performance (J. Huang et al., 2023). Valmeekam and colleagues report analogous negative results on planning (Valmeekam et al., 2023). The structural picture is clear: Reflexion-style reflection delivers reliability gains precisely when the retry loop has access to an objective signal-compilation errors, unit-test results, tool exceptions, environment rewards. On tasks where the model grades itself without ground truth, expected gain is near zero and variance is high. Effect sizes are also strongest on frontier models; smaller models sometimes regress under reflection, plausibly owing to drift in the reflection step itself (J. Huang et al., 2023).

4.4 Pattern 4: Least-privilege tool access scoping

Evidence grade: **moderate to strong on the accuracy strand; primarily design-based on the security strand**.

The accuracy strand is supported by several controlled and quasi-controlled studies. Qin and colleagues introduce ToolBench with 16,464 APIs and demonstrate that feeding the full API pool to a model is infeasible; a neural retriever that pre-filters the menu to a small relevant subset is required to obtain usable performance (Qin et al., 2023). Patil and colleagues document hallucinated API calls as the dominant failure mode when models face large API sets, and show that retrieval-aware fine-tuning-in effect, a narrowing of the candidate tool set-reduces hallucination and produces gains over GPT-4 on APIBench (Patil et al., 2023). Huang and colleagues introduce MetaTool, a dedicated benchmark for tool selection in the presence of similar distractors, and document that selection accuracy degrades sharply as distractor tools are added (Y. Huang et al., 2023). Lu and colleagues, in AppBench, evaluate nine LLMs including GPT-4o on graph-structured planning tasks over approximately ten apps and twenty APIs and show that success rate collapses as graph complexity grows; neither in-context learning nor fine-tuning closes the gap (Lu et al., 2024).

The effect is consistent across GPT-3.5, GPT-4, and open-weight tool-use fine-tunes, but several observations complicate the simple "smaller menu is always better" reading. First, the degradation is model-dependent: frontier agentic models with better tool retrieval and longer context windows partly overcome the menu-size penalty, suggesting the effect is weakening as models improve. Second, restricting the tool menu too aggressively removes tools the agent may later need, producing an inverse failure mode in which the agent correctly selects from its reduced menu but cannot in principle succeed because the necessary tool was withheld. Least-privilege scoping is thus best understood as a context-dependent trade-off calibrated to the task distribution, not as a universal prescription.

The security strand is supported by design-oriented rather than controlled-comparison work. Framework documentation across the major agent platforms recommends per-role or per-invocation permission scoping (Anthropic, 2024b; OpenAI, 2025; LangChain, 2024; Anthropic, 2024a); the security-benefit claim, however, rests on standard principle-of-least-privilege arguments rather than on published empirical demonstrations that scoped tool access reduces incidence of prompt-injection-induced tool misuse in production. This is a gap discussed further in Section 6.

4.5 Pattern 5: Explicit failure propagation and handling

Evidence grade: **moderate**, with strong observational grounding and thinner controlled-comparison evidence.

The observational grounding is robust. Cemri and colleagues, via the MAST taxonomy, annotate 1,600 execution traces across seven multi-agent frameworks and report that failure-

handling issues-misclassified errors, silent failures, unpropagated exceptions, and cascaded misunderstandings-account for a large share of observed failure cases, with an inter-annotator κ of 0.88 (Cemri et al., 2025). This study is, as of the date of writing, the most comprehensive published audit of multi-agent failure modes, and its vocabulary underpins most current discussion of Pattern 5.

Controlled ablations of failure-propagation mechanisms are rarer. Framework documentation describes typed error returns, supervisor-level error routing, and speculative or selective re-execution, but published studies that isolate the effect of switching between silent failure and structured propagation on reliability are thin. The available controlled evidence is largely indirect: the gains reported for retry-with-reflection (Pattern 3) presuppose that failure is detected and propagated to the reflection step, so those studies implicitly demonstrate the value of propagation. Pattern 5 as a stand-alone intervention-turning silent failures into typed signals without any downstream reflection or retry-has not, to the best of the available literature, been benchmarked in isolation.

The structural argument for Pattern 5 is nevertheless strong: no downstream reliability pattern can fire on a failure it cannot see. The evidence-quality verdict of "moderate" reflects the mismatch between the strength of the structural argument and the relative scarcity of isolated empirical tests.

4.6 Pattern 6: Deterministic output formatting

Evidence grade: **strong for structural reliability, with a well-documented reasoning trade-off**.

The case for constrained decoding on structural grounds is effectively settled. Willard and Louf establish that finite-state-machine-guided decoding achieves 100% compliance by construction with near-constant per-token overhead (Willard & Louf, 2023). Beurer-Kellner and colleagues, in LMQL, report token-level inference masks that retain or improve accuracy while reducing cost by 26–85% across several tasks (Beurer-Kellner et al., 2023). Dong and colleagues, in XGrammar, report up to a hundred-fold speed-up over prior grammar-constrained engines with near-zero end-to-end overhead, and the engine has been adopted as the default backend by vLLM, SGLang, TensorRT-LLM, and MLC-LLM (Dong et al., 2024). Geng and colleagues, in the JSONSchemaBench evaluation across ten thousand real schemas and six frameworks, report near-100% structural validity for constrained decoders and substantially lower success rates for unconstrained prompting, with closed-source grammar engines trailing open-source implementations (Geng et al., 2025). Ogundepo and colleagues document a large capability gap between models on format-following alone: Gemini 1.5 Pro reaches 93.4% structural success on StructuredRAG tasks while a comparably sized Llama 3 8B instruction-tuned model reaches 71.7% (Ogundepo, Shorten & Team, 2024).

The conflicting evidence concerns the reasoning side of the ledger. Tam and colleagues report a controlled comparison across three regimes-JSON-mode constrained decoding, format-

restricting instructions, and natural-language-to-format post-processing-and find that strict format coupling reduces reasoning accuracy, with the effect strongest on mathematical and symbolic tasks and weaker or absent on classification (Tam et al., 2024). On GSM8K the reported reduction is 27.3 percentage points relative to free-form reasoning followed by extraction.

The reconciliation that emerges across the two evidence lines is consistent. Constrained decoding on the *formatting* step is close to free; coupling constrained decoding to the *reasoning* step damages reasoning. The engineering implication is a two-step pattern: reason in free text, then emit a constrained structured wrapper for the final answer. This matches the reasoning-then-format guidance in the OpenAI structured-outputs documentation and the XML-tagged chain-of-thought pattern documented by Anthropic (OpenAI, 2025; Anthropic, 2024a), but the empirical case for it in the academic literature rests primarily on Tam and colleagues and should be treated as a single-source result pending independent replication.

4.7 Evidence summary

Table 1 summarises the evidence grade and the strongest source for each pattern.

#	Pattern	Grade	Strongest sources (selected)
1	Structured tool schemas	Moderate–strong	Geng et al. (2025); Schick et al. (2023); Qin et al. (2023); conflict: Tam et al. (2024)
2	Explicit validator agents	Strong	Cobbe et al. (2021); X. Wang et al. (2022); Zheng et al. (2023); Gou et al. (2023); boundary: J. Huang et al. (2023)
3	Retry with reflection	Strong (exec. signal); weak otherwise	Shinn et al. (2023); Madaan et al. (2023); Chen et al. (2023); boundary: J. Huang et al. (2023); Valmeekam et al. (2023)
4	Least-privilege tool access	Moderate–strong (accuracy); design-based (security)	Qin et al. (2023); Patil et al. (2023); Y. Huang et al. (2023); Lu et al. (2024)
5	Explicit failure propagation	Moderate (observational)	Cemri et al. (2025); indirect via Pattern 3
6	Deterministic output formatting	Strong (structure); conflicting (reasoning)	Willard and Louf (2023); Dong et al. (2024); Geng et al. (2025); conflict: Tam et al. (2024)

Table 1: Evidence summary for the six reliability patterns. Grades follow the rubric defined in Section 3.

A cross-pattern observation is worth recording. In every case where a pattern's evidence is contested-intrinsic self-correction in Pattern 2, reasoning degradation under strict format in Pat-

terns 1 and 6, reflection without oracle in Pattern 3-the common factor is the absence or presence of an external grounding signal. Reliability gains from these patterns appear to depend on injection of information from outside the generator: a separate judge, a tool's execution output, a compilation result, a retrieval result, or a ground-truth test. Patterns that operate purely on self-assessment by the same generator are fragile. This is consistent with the broader hallucination literature, which converges on the same structural diagnosis: unfounded claims arise precisely where no external grounding is available (Ji et al., 2023; L. Huang et al., 2023).

5. Architectural Compositions

The six patterns do not stand alone; they are applied within architectures. Three canonical architectures recur in the literature and in the framework documentation, and each foregrounds a different subset of patterns.

5.1 Single agent with tools

A single LLM controls the loop and invokes tools through a bound schema. This is the architecture of ReAct (Yao et al., 2022), the baseline in most tool-use benchmarks, and the default entry point of the Model Context Protocol (Anthropic, 2024b). Patterns 1 and 6 carry most of the reliability load here: the tool-invocation schema and the output format jointly determine whether the model can interact with its environment without malformed actions. Pattern 4 becomes relevant as soon as the tool menu grows past a handful of options; the evidence of Qin and colleagues, Patil and colleagues, and Huang and colleagues (Qin et al., 2023; Patil et al., 2023; Y. Huang et al., 2023) directly targets this setting. Pattern 3 is applicable wherever the environment returns an executable signal, and enjoys the strongest evidence in precisely this context (Shinn et al., 2023; Chen et al., 2023). Patterns 2 and 5 can be added but often in reduced form: a validator step without a separate agent identity, an error return without role-based propagation.

5.2 Sequential multi-agent (planner–executor–validator)

Two or more agents communicate in sequence, typically as a planner that decomposes the task, one or more executors that carry out subtasks with tool access, and a validator that checks the output before return. The architecture is documented in AutoGen (Wu et al., 2023) and recurs across the agent-framework literature (Xi et al., 2023; L. Wang et al., 2023). Patterns 2 and 5 are foregrounded: the validator is the explicit instantiation of Pattern 2, and the inter-agent boundary is where Pattern 5 either holds or fails (Cemri et al., 2025). Pattern 4 applies at the role boundary-the executor's tool menu should be scoped to the subtask; the planner and validator may have no tool access at all. Pattern 1 applies inside each executor's tool boundary. Pattern 3 is typically implemented by looping back from the validator to the executor with a

reflection step; this composition is precisely the configuration for which the Reflexion evidence is strongest (Shinn et al., 2023).

5.3 Parallel multi-agent with shared context

Multiple agents operate concurrently on a shared state. The architecture is native to frameworks that support dispatch-and-aggregate patterns, and appears in the AutoGen 0.4 async redesign (Wu et al., 2023). Patterns 5 and 6 dominate: failures must be propagated across concurrent branches without silent loss, and outputs from parallel branches must be structurally compatible for aggregation. Pattern 2 naturally becomes distributed-a judge or vote over parallel branches, which is the same structural device as self-consistency but at the agent level rather than the sample level (X. Wang et al., 2022). Empirical evidence specific to this architecture is thinner than for the sequential case; much of the published evaluation of parallel multi-agent systems is at the system-paper level rather than through controlled isolation of individual patterns (Wu et al., 2023; Li et al., 2023).

Architecture	Dominant patterns	Principal evidence
Single agent with tools	1, 6, 4, 3	Yao et al. (2022); Shinn et al. (2023); Qin et al. (2023); Geng et al. (2025)
Sequential multi-agent	2, 5, 4, 3	Wu et al. (2023); Shinn et al. (2023); Cemri et al. (2025)
Parallel multi-agent	5, 6, 2	Wu et al. (2023); X. Wang et al. (2022); Cemri et al. (2025)

Table 2: Pattern salience across three canonical architectures. Ordering within a row is approximately from most to least dominant.

6. Gaps in the Evidence

An honest synthesis must state not only what is known but what is not. The following gaps recur across the reviewed literature and are, in the author's reading, the most consequential for practitioners.

Interaction effects between patterns are almost entirely unstudied. The six patterns are rarely applied in isolation; production systems combine several at once. Whether the reliability gain from Pattern 3 is additive with, dependent on, or cancelled by Pattern 2 in a combined planner–executor–reflector–validator configuration is, to the best of the reviewed literature, an open empirical question. System-paper ablations typically remove one component at a time and do not map the joint design space.

Cross-model generalisation is under-studied. Most controlled comparisons are performed on a single frontier model family, typically the GPT-4 series, with occasional replication on Claude or Gemini. Whether a pattern that delivers a 20-point gain on GPT-4 delivers a 20-point gain,

a 5-point gain, or a regression on a 7B open model is not reliably known across patterns. The JSONSchemaBench evaluation is a partial exception in that it compares multiple engines and models within a single benchmark (Geng et al., 2025), but this is the exception rather than the rule. The hallucination-survey literature flags the same issue for factuality evaluations (L. Huang et al., 2023).

Long-horizon reliability is largely absent from the evidence base. Almost all the cited benchmarks measure success on single bounded tasks of minutes-to-hours scope. τ-bench's pass^k metric is a partial answer on the repetition dimension (Yao et al., 2024), but drift over long-running sessions, cumulative context degradation, and multi-session reliability are poorly benchmarked. Practitioner guidance on session length, context resets, and working-memory management is therefore largely extrapolative.

Failure-propagation (Pattern 5) lacks isolated controlled comparison. The MAST observational audit is the strongest piece of evidence available (Cemri et al., 2025), but a study that takes a fixed architecture and varies only the failure-handling mechanism-silent versus typed propagation with no other changes-does not appear in the reviewed literature. Given how frequently practitioner guides recommend explicit error channels, this is a surprisingly thin empirical foundation.

Security properties of least-privilege scoping (part of Pattern 4) are argued rather than measured. The accuracy strand is well supported (Qin et al., 2023; Patil et al., 2023; Y. Huang et al., 2023; Lu et al., 2024). The security strand rests on principle-of-least-privilege arguments familiar from classical systems security and on framework design, but published studies that quantify the reduction in successful prompt-injection-driven tool misuse attributable to scoped tool access are rare.

The Tam et al. reasoning-degradation result on strict formatting is a single-source finding that warrants replication. (Tam et al., 2024) The effect size (up to 27 percentage points on GSM8K) is too large, and the engineering implication too significant, to rest on one study. Replication across additional reasoning benchmarks and additional model families would either consolidate or temper the current advice to decouple reasoning from formatting.

Benchmark saturation and benchmark specificity are known hazards. SWE-bench Verified was retired from frontier evaluation in 2026 (OpenAI, 2024), and the benchmark literature continues to churn rapidly. Pattern-effectiveness results tied to a retired benchmark may age poorly; this argues for evaluating patterns across multiple contemporaneous benchmarks rather than anchoring on a single number.

7. Recommendations and Open Questions

The following graded recommendations follow from Section 4. The grading tracks the evidence grade rather than the intuitive appeal of the pattern, and practitioners should weight recommendations accordingly.

Default: Pattern 6 at the output boundary, Pattern 1 at the tool boundary-provided reasoning is not coupled to the constrained decoding step. The evidence for structural reliability gains is strong and the engineering cost is low (Willard & Louf, 2023; Dong et al., 2024; Geng et al., 2025). Reasoning should be permitted to proceed in free text before the constrained emission step, consistent with the conflict reported by Tam and colleagues (Tam et al., 2024).

Default: Pattern 2 with an external grounding signal. A validator agent, judge, or voting ensemble should be placed between generation and commitment whenever such a signal is available (unit tests, tool output, retrieved documents, a secondary model). Purely intrinsic self-judge loops on the same model should not be treated as reliability-adding under the current evidence (J. Huang et al., 2023; Valmeekam et al., 2023).

Default on tasks with an executable signal: Pattern 3. Reflection-plus-retry carries strong evidence in software-engineering, tool-use, and game-like environments where the environment returns an objective error (Shinn et al., 2023; Chen et al., 2023; Kim et al., 2023). On tasks without such a signal, reflection should be treated as an optional intervention whose expected gain is near zero and whose variance is non-trivial.

Context-dependent: Pattern 4. Tool-menu scoping should be calibrated to the task distribution. Very large tool menus degrade selection accuracy reliably (Qin et al., 2023; Patil et al., 2023; Y. Huang et al., 2023; Lu et al., 2024); excessively narrow menus remove capability. A retrieval layer that dynamically selects a relevant subset per task is the most evidence-supported middle path. The security-side case for strict scoping is strong on principle but weakly quantified in the literature.

Treat as a structural default rather than an evidence-driven one: Pattern 5. Explicit failure propagation is recommended by every framework and is structurally necessary for any downstream reliability pattern to fire on a failure it cannot see (Cemri et al., 2025; Anthropic, 2024b; LangChain, 2024; OpenAI, 2025). The empirical isolation of its effect is not, however, currently available in the literature.

Open questions worth funding or studying. Five specific empirical questions would materially advance the field. First, a full-factorial or fractional-factorial ablation of Patterns 1–6 on a fixed task suite and a fixed model, to quantify interaction effects. Second, a cross-model replication of the Tam et al. reasoning-format study across a range of open-weight models and reasoning benchmarks (Tam et al., 2024). Third, a controlled comparison of silent versus typed failure propagation with all other variables held constant, isolating Pattern 5. Fourth, a long-horizon reliability benchmark that extends τ-bench's repetition metric to multi-session and multi-day agent deployments (Yao et al., 2024). Fifth, a security-evaluation benchmark for Pattern 4 that measures successful prompt-injection-driven tool misuse against scoped and unscoped tool access.

8. Conclusion

Six engineering patterns recur across the practitioner literature on reliable LLM-based multi-agent systems. Three of them-validator-in-chain with external grounding, reflection-and-retry on executable signals, and deterministic output formatting at the emission boundary-are supported by multiple independent controlled studies with reported effect sizes. Three further patterns-structured tool schemas, least-privilege tool scoping, and explicit failure propagation-are widely recommended and partly supported, but rest on thinner controlled evidence than their prevalence in framework documentation suggests. A cross-cutting theme is that reliability gains from these patterns depend on the injection of an external grounding signal; patterns that rely on self-assessment by the same generator are fragile. The evidence base is growing quickly, and several of the gaps identified here would be straightforward to close with dedicated ablation studies. Until such studies are available, practitioners are advised to grade their own adoption decisions by the evidence grade recorded here rather than by the uniformity of the recommendation across framework documentation.

Conflict of Interest Statement

The author declares no competing interests.

License

References

Anthropic. (2024a). *Building effective agents.* `https://www.anthropic.com/engineering/building-effective-agents`. (Engineering guidance; accessed April 2026)

Anthropic. (2024b). *Model context protocol specification.* `https://modelcontextprotocol.io/`. (Protocol specification; accessed April 2026)

Beurer-Kellner, L., Fischer, M. & Vechev, M. (2023). Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages (PACMPL)*, *7*(PLDI). (arXiv:2212.06094)

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R. et al. (2025). Why do multi-agent LLM systems fail? *arXiv preprint arXiv:2503.13657*.

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E. et al. (2024). MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*.

Chen, X., Lin, M., Schärli, N. & Zhou, D. (2023). Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L. et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Dong, Y., Ruan, C. F., Cai, Y., Lai, R., Xu, Z., Zhao, Y. & Chen, T. (2024). XGrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*.

Geng, S., Josifoski, M., Peyrard, M. & West, R. (2023). Grammar-constrained decoding for structured NLP tasks without finetuning. *arXiv preprint arXiv:2305.13971*.

Geng, S., Josifoski, M., Peyrard, M., West, R. et al. (2025). JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*.

Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N. & Chen, W. (2023). CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X. & Zhou, D. (2023). Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H. et al. (2023). A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*.

Huang, Y., Shi, J., Li, Y., Fan, C., Wu, S., Zhang, Q. et al. (2023). MetaTool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y. et al. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, *55*(12), 1–38.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O. & Narasimhan, K. (2023). SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*.

Kim, G., Baldi, P. & McAleer, S. (2023). Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M. C., Huang, P.-Y., ... Fried, D. (2024). VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

LangChain. (2024). *LangGraph documentation.* `https://github.com/langchain-ai/langgraph`. (Framework documentation; accessed April 2026)

Li, G., Hammoud, H. A. A. K., Itani, H., Khizbullin, D. & Ghanem, B. (2023). CAMEL: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H. et al. (2023). AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*.

Lu, H., Wang, Y., Yang, D., Zhuang, Y., Zhang, N., Li, B., ... others (2024). AppBench: Planning of multiple APIs from various APPs for complex user instruction. *arXiv preprint*

arXiv:2410.19743.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S. et al. (2023). Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Mialon, G., Fourrier, C., Swift, C., Wolf, T., LeCun, Y. & Scialom, T. (2023). GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*.

Ogundepo, O., Shorten, C. & Team, W. (2024). StructuredRAG: JSON response formatting with large language models. *arXiv preprint arXiv:2408.11061*.

OpenAI. (2024). *Introducing SWE-bench Verified.* `https://openai.com/index/introducing-swe-bench-verified/`. (Technical report; accessed April 2026)

OpenAI. (2025). *OpenAI agents SDK documentation.* `https://openai.github.io/openai-agents-python/`. (Framework documentation; accessed April 2026)

Patil, S. G., Zhang, T., Wang, X. & Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y. et al. (2023). ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., ... Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K. & Yao, S. (2023). Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*.

Tam, Z. R., Wu, C.-K., Tsai, Y.-L., Lin, C.-Y., Lee, H.-y. & Chen, Y.-N. (2024). Let me speak freely? a study on the impact of format restrictions on performance of large language models. *arXiv preprint arXiv:2408.02442*.

Valmeekam, K., Marquez, M., Sreedharan, S. & Kambhampati, S. (2023). On the planning abilities of large language models: A critical investigation. *arXiv preprint arXiv:2305.15771*.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J. et al. (2023). A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., ... Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Willard, B. T. & Louf, R. (2023). Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E. et al. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Xi, Z., Chen, W., Guo, X., He, W., Ding, Y., Hong, B. et al. (2023). The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Yao, S., Shinn, N., Razdaibiedina, P. & Narasimhan, K. (2024). τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. & Cao, Y. (2022). ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T. et al. (2023). Siren's song in the AI ocean: A

survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y. et al. (2023). Judging LLM-as-a-judge with MT-Bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A. et al. (2023). WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.