Beyond Tool Selection: Diagnosing Downstream Failures in MCP Tool-Using LLM Agents

Sungho Bae [1] **Hyunwoo Park** [1]

Abstract

Tool-using agents are often assumed to fail because they select the wrong tools. We show that this explanation is insufficient in realistic MCP environments. We introduce a stage-aware failure taxonomy that assigns failed trajectories to No-tool, Routing, Workflow, Request/Parameter, Evidence Use, or Runtime failures, separating tool-selection errors from downstream execution failures. Across five agents on 427 MCP-Atlas tasks, restricting agents to gold tools yields only modest gains and leaves many failures in workflow continuation, request grounding, evidence use, and runtime stability. We then use evidence-gap and tool-path guidance as diagnostic probes. Proactive guidance is more effective than reactive post-final feedback, but many failures remain or shift to downstream stages. A trace-level case study further shows that agents can use the same relevant tool families yet diverge depending on query precision, evidence linking, and workflow persistence. Our results indicate that gold-tool access is necessary but not sufficient: reliable MCP agents require evidence-state tracking, query repair, and workflow-level verification beyond better tool selection.

1. Introduction

Large language model agents are increasingly expected to solve tasks by interacting with external tools, APIs, databases, files, and execution environments. The Model Context Protocol (MCP) has accelerated this trend by standardizing how agents connect to heterogeneous external capabilities (Anthropic, 2024). In such settings, agent reliability depends not only on whether a model can generate fluent responses, but also on whether it can select appropriate tools, issue grounded tool calls, interpret observations, and continue multi-step workflows until sufficient evidence has been collected.

A common explanation for tool-use failure is wrong tool selection. This view is intuitive: if an agent chooses an irrelevant tool, the task is unlikely to be completed. As a result, much prior work has focused on tool retrieval, API matching, schema grounding, or improving the validity of tool calls. However, real MCP tasks often require more than selecting the right tool identity. Even when a relevant tool is available, an agent may stop after only part of the workflow, call the right tool with poorly grounded arguments, ignore evidence returned by the tool, or fall into repeated loops and runtime failures. These failures are difficult to diagnose from end-to-end success rates alone.

This paper studies where MCP tool-using agents fail. We frame tool use as a staged execution process and introduce a stage-aware failure taxonomy that separates failures before relevant tool use from failures that occur after a relevant tool path has been entered. The taxonomy assigns failed trajectories to T0 No-tool, T1 Routing, T2 Workflow, T3 Request/Parameter, T4 Evidence Use, or TX Runtime. This allows us to ask whether failures are primarily due to tool selection, or whether substantial failures remain in downstream execution stages after agents are given or guided toward relevant tools.

We evaluate five agents on MCP-Atlas, a benchmark built on real MCP servers with multi-step tasks and claims-based scoring (Bandi et al., 2026). We first compare a standard BASE setting, where the full enabled tool set is available, with a gold-tool setting, where the agent receives only the tools used in the reference trajectory. This counterfactual reduces tool-selection ambiguity while leaving the agent responsible for tool-call ordering, argument construction, evidence interpretation, and final synthesis. We then analyze stage-level failure distributions and use guidance interventions as diagnostic probes to test which failures are repaired, shifted, or left unresolved.

Our results show that good tool access is necessary but not sufficient. Restricting agents to reference tools improves some models, but the gains remain modest and many failures

[1]Graduate School of Data Science, Seoul National University, Seoul, South Korea. Correspondence to: Hyunwoo Park <hyunwoopark@snu.ac.kr>.

Accepted at ICML 2026 Workshop on Failure Modes in Agentic AI (FAGEN), Seoul, South Korea. Copyright 2026 by the author(s).

persist. More importantly, the remaining failures are concentrated in downstream stages: agents often enter a relevant tool path but fail to complete the workflow, ground tool requests, use returned evidence, or maintain stable execution. These findings suggest that improving MCP agent reliability requires more than better tool retrieval or tool selection; it also requires mechanisms for workflow persistence, request grounding, evidence use, and runtime robustness.

In summary, this paper makes three contributions. First, we propose a stage-aware taxonomy for diagnosing MCP tool-use failures at the trajectory level. Second, we use gold-tool access as a diagnostic counterfactual to show that relevant tool identity alone does not resolve MCP failures. Third, we analyze guidance interventions as diagnostic probes, revealing which failure stages are sensitive to evidence or tool-path guidance and which remain persistent across models.

2. Related Work

2.1. Tool-Using LLM Agents and Trace-Level Evaluation

Large language models are increasingly deployed as agents that interact with external tools, APIs, databases, and execution environments. Early work such as Toolformer (Schick et al., 2023) showed that language models can learn when and how to invoke external APIs, while ReAct (Yao et al., 2023) introduced interleaved reasoning-and-acting trajectories for multi-step tool use. More recently, the Model Context Protocol (MCP) has provided a standardized interface between AI applications and external tools or data sources (Anthropic, 2024), making it easier to connect agents to heterogeneous external capabilities.

A broad line of benchmarks evaluates tool-using and interactive LLM agents. ToolLLM/ToolBench (Qin et al., 2023), API-Bank (Li et al., 2023), and Gorilla (Patil et al., 2024) focus on API retrieval, tool invocation, and documentation-grounded API use. Other benchmarks move toward broader interactive settings: AgentBench (Liu et al., 2023) evaluates agents across diverse environments, WebArena (Zhou et al., 2023) studies web-based task completion, τ-bench (Yao et al., 2025) evaluates tool-agent-user interaction in realistic service domains, and AppWorld (Trivedi et al., 2024) tests agents in simulated app environments with executable APIs and state-based evaluation. MCP-Atlas (Bandi et al., 2026) is especially relevant to our setting because it evaluates agents on real MCP servers with multi-step tasks and claims-based scoring. These benchmarks have substantially advanced tool-use evaluation, but their analyses are often centered on end-to-end success or capability-specific scores. Consequently, they provide limited visibility into where a failed trajectory breaks down.

Closer to our diagnostic goal, T-Eval (Chen et al., 2024) decomposes tool use into sub-capabilities such as planning, retrieval, reasoning, and understanding. ToolEmu (Ruan et al., 2024) studies agent failures and risks through emulated tool execution, and MAST (Cemri et al., 2025) provides a taxonomy of failures in multi-agent LLM systems. Our work is complementary to these studies. We focus on single-agent MCP trajectories and assign failures to stage-level categories, separating no-tool behavior, routing failures, workflow failures, request or parameter failures, evidence-use failures, and runtime failures.

2.2. Tool Selection and Diagnostic Interventions

A large portion of tool-use research aims to help agents identify, retrieve, or invoke the right tool. This includes API retrieval, schema matching, documentation grounding, and training models to generate valid tool calls (Qin et al., 2023; Li et al., 2023; Patil et al., 2024). These directions are important because tool selection is a prerequisite for successful tool use. However, tool selection alone does not determine whether an MCP task is completed. Even after a relevant tool path is available, agents may terminate a workflow prematurely, issue poorly grounded requests, fail to use returned evidence, or fall into non-productive runtime behavior. We therefore treat tool selection as one stage in a longer execution process and ask which failures persist after agents are guided toward good tools.

Prior work has also explored scaffolding and feedback mechanisms for improving reasoning and agent behavior. Chain-of-thought prompting (Wei et al., 2022) encourages intermediate reasoning, while Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use verbal or self-generated feedback to improve subsequent outputs. Such methods show that additional guidance can improve agent performance, but they do not directly isolate which parts of MCP tool-use trajectories are repaired or left unresolved. Our study therefore uses controlled guidance interventions as diagnostic probes. Instead of proposing a new agent method, we compare how guidance changes success rates and failure stages, thereby identifying which failures are associated with tool selection and which failures remain after good tools are made available.

3. Diagnostic Study Design

This work is an empirical diagnostic study of where agents fail in real MCP tool-use environments. We ask whether failures persist in downstream execution stages even after agents are given, or guided toward, relevant tools. To answer this question, we analyze MCP-Atlas trajectories, assign failed runs to a stage-aware attribution scheme, and use tool-access counterfactuals and guidance interventions as diagnostic probes.

3.1. Benchmark, Agents, and Evaluation

We evaluate agents on MCP-Atlas (Bandi et al., 2026), a benchmark built on real MCP servers with multi-step tasks and claims-based scoring. Each task provides a user instruction, an enabled tool set, a reference trajectory, and a set of atomic claims used for evaluation. This setting requires agents not only to select tools, but also to issue grounded tool calls, interpret observations, and synthesize a final answer from gathered evidence.

We represent each task i as

$$(x_i, \mathcal{T}_i, \mathcal{G}_i, \mathcal{C}_i),$$

where x_i is the user instruction, $\mathcal{T}_i$ is the enabled tool set, $\mathcal{G}_i \subseteq \mathcal{T}_i$ is the set of tools used in the reference trajectory, and $\mathcal{C}_i$ is the set of atomic claims that the final answer should satisfy. An agent produces a trajectory

$$\tau_i = (x_i, a_{i1}, o_{i1}, \ldots, a_{iT_i}, o_{iT_i}, y_i),$$

where a_{it} is a tool call or final-answer action, o_{it} is the corresponding observation, and y_i is the final response.

We follow the benchmark's claims-based evaluation protocol. Each final answer is judged against the atomic reference claims using Grok-3 (xAI, 2025), with each claim scored independently. The final response is scored by the fraction of claims in $\mathcal{C}_i$ that are satisfied, yielding a coverage score $s_i \in [0, 1]$. We report mean coverage and pass rate under a fixed threshold:

$$\text{Pass}_i = \mathbb{I}[s_i \geq \theta],$$

where $\theta = 0.75$ in our main experiments. Coverage measures partial task completion, while our trace-level taxonomy analyzes where the trajectory breaks down. We evaluate GPT-4o (OpenAI, 2024), Claude Sonnet 4 (Anthropic, 2025), Gemma 4-31B (Google, 2026), Qwen3-30B-A3B (Yang et al., 2025), and Mistral Small 3.2 (Mistral AI, 2025) to compare failure patterns across agents with different tool-use behaviors.

3.2. Stage-Aware Failure Taxonomy

End-to-end success rates do not reveal where a failed trajectory breaks down. We therefore organize failures according to the stages of MCP tool-use execution. An agent must first initiate tool use, route to relevant tools, continue the required workflow, ground its requests and parameters, and finally use the retrieved evidence in its answer. Our taxonomy follows this staged process and assigns each failed run to the stage where the observed trajectory first becomes insufficient under our attribution rules.

This staged view separates failures that occur before relevant tool use from failures that remain after a relevant tool path has been entered. T0 and T1 capture pre-relevant-tool failures: either no tool is called, or the agent routes to an irrelevant tool family. T2–T4 capture downstream execution failures in workflow continuation, request grounding, and evidence use. TX captures runtime failures such as timeouts, repeated loops, or tool/server errors. Detailed attribution rules are provided in the Appendix A.

Table 1. Stage-aware failure taxonomy. Buckets follow the staged process of MCP tool use, from tool initiation and routing to workflow execution, request grounding, evidence use, and runtime stability.

Bucket	Stage	Primary diagnostic question
Selection-stage failures		
T0	No-tool	Did the agent fail to initiate tool use?
T1	Routing	Did the agent enter an irrelevant tool family?
Downstream execution failures		
T2	Workflow	Did the agent stop before completing the workflow?
T3	Req./Param.	Were requests or parameters poorly grounded?
T4	Evidence Use	Did the agent obtain evidence but fail to use it?
Runtime failures		
TX	Runtime	Was the run dominated by timeout, looping, or tool/runtime errors?

3.3. Tool-Access Counterfactuals

To separate tool selection from downstream execution failures, we compare two tool-access conditions. In the BASE condition, the agent receives the full enabled tool set $\mathcal{T}_i$ provided by the benchmark. This reflects the standard MCP setting, where the agent must identify relevant tools among multiple candidates and decide how to use them over multiple turns.

In the gold-tool condition, the agent receives only the reference tool set $\mathcal{G}_i$ used in the reference trajectory. This condition is not intended as a deployable method, but as a diagnostic counterfactual. It reduces tool-selection ambiguity while leaving execution decisions to the agent: the gold-tool condition does not provide the correct tool-call order, arguments, queries, evidence values, or final synthesis.

This comparison asks whether good tool access is sufficient for task completion. If failures primarily come from tool selection, restricting available tools to $\mathcal{G}_i$ should remove most failures. If failures remain in T2 Workflow, T3 Request/Parameter, T4 Evidence Use, or TX Runtime, then relevant tool access is necessary but not sufficient: agents still need to execute the workflow, construct grounded requests, use returned evidence, and avoid non-productive runtime behavior.

3.4. Guidance Interventions as Diagnostic Probes

Tool-access counterfactuals show what happens when the candidate tool set is restricted, but they do not show how

trajectories respond to different forms of guidance. We therefore use controlled guidance interventions as diagnostic probes. The goal is not to propose a new prompting method, but to perturb the trajectory in controlled ways and observe which failure stages are repaired, shifted, or left unresolved.

We organize guidance along two axes: timing and content. Reactive guidance is injected after an insufficient final answer, whereas proactive guidance is injected before the first model response. Guidance content ranges from a minimal sufficiency signal to evidence-gap information, tool-path information, and an integrated signal containing both. We also include Forced Tool Control, which forces the next tool identity after a reactive intervention, as a stronger control over tool selection. To avoid answer leakage, evidence-gap guidance masks answer values with `[value]` and provides only the structure of missing evidence.

These interventions support two diagnostic questions. First, does guidance merely extend execution after a premature stop, or does it redirect the trajectory from the beginning? Second, what failure stages remain even after the agent is guided toward useful evidence or tools? By comparing how each intervention changes both success rates and stage-level failure distributions, we identify which failures are associated with tool selection and which failures persist after good tools are made available or explicitly suggested. Detailed construction of the guidance messages, sanitized evidence slots, gate-firing logic, and leakage-control procedure is provided in Appendix B.

4. Results

4.1. Relevant Tool Access Is Necessary but Not Sufficient

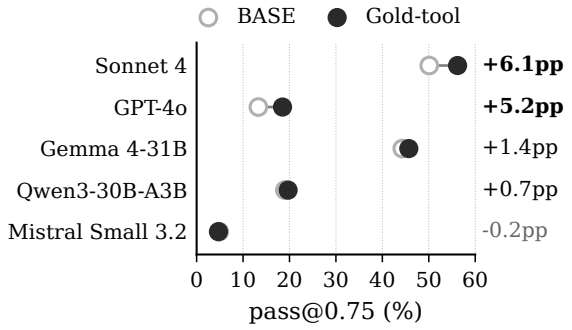

Figure 1. Pass@0.75 under BASE and gold-tool access across five MCP agents. Gold-tool access improves some models, but the gains remain modest and several agents still have low absolute success rates, indicating that relevant tool identity alone does not resolve MCP failures.

We first ask whether MCP agent failures can be explained mainly by tool selection. We compare the BASE condition, where the agent receives the full enabled tool set, with the gold-tool condition, where the agent receives only the tools used in the reference trajectory. If wrong tool selection were the main bottleneck, restricting the tool set to reference tools should remove most failures.

Figure 1 shows that gold-tool access helps some models but does not resolve MCP failures. Claude Sonnet 4 improves from 50.1% to 56.2%, and GPT-4o improves from 13.3% to 18.5%. However, the effect is much smaller for Gemma 4-31B and Qwen3-30B-A3B, and Mistral Small 3.2 does not improve. Thus, relevant tool access is useful, but it is not enough to explain or eliminate most failures.

This result suggests that tool identity is a prerequisite rather than a complete solution. Gold-tool access reduces tool-selection ambiguity, but it does not provide the correct tool-call order, arguments, evidence values, or final synthesis. We therefore next examine where the remaining failures occur. We report the full per-model BASE and gold-tool breakdowns in Appendix C.

4.2. Failures Persist in Workflow, Request, Evidence, and Runtime Stages

Figure 2 compares stage-level outcome distributions under BASE and gold-tool access. By construction, gold-tool access removes T1 Routing failures because the available tools are restricted to the reference set. However, these runs do not simply become successful. Many failures remain after the agent has entered a relevant tool path.

The largest remaining failures are workflow continuation and request/parameter grounding. GPT-4o still has many T2 Workflow failures under gold-tool access (188 in BASE and 171 in gold-tool), while its T3 Request/Parameter failures increase from 33 to 66. Mistral Small 3.2 shows an even stronger workflow bottleneck: T2 failures increase from 196 to 230 under gold-tool access. These runs often call part of the relevant tool chain but stop before the task is complete. In contrast, Gemma 4-31B is dominated by request grounding: its T3 failures increase from 84 to 106, indicating that the model often reaches the right tool family but issues poorly grounded queries, filters, entities, or dates.

Runtime failures form a separate bottleneck. Qwen3-30B-A3B is dominated by TX Runtime failures in both conditions, with 153 in BASE and 147 under gold-tool access. Claude Sonnet 4 achieves the highest pass rate, but still retains 88 TX failures under gold-tool access. These cases show that some failures are not mainly about finding the right tool, but about maintaining stable execution without repeated loops, timeouts, context overflows, or tool/server errors.

T4 Evidence Use failures are smaller under our strict attribution rule, but they still remain across models. Because

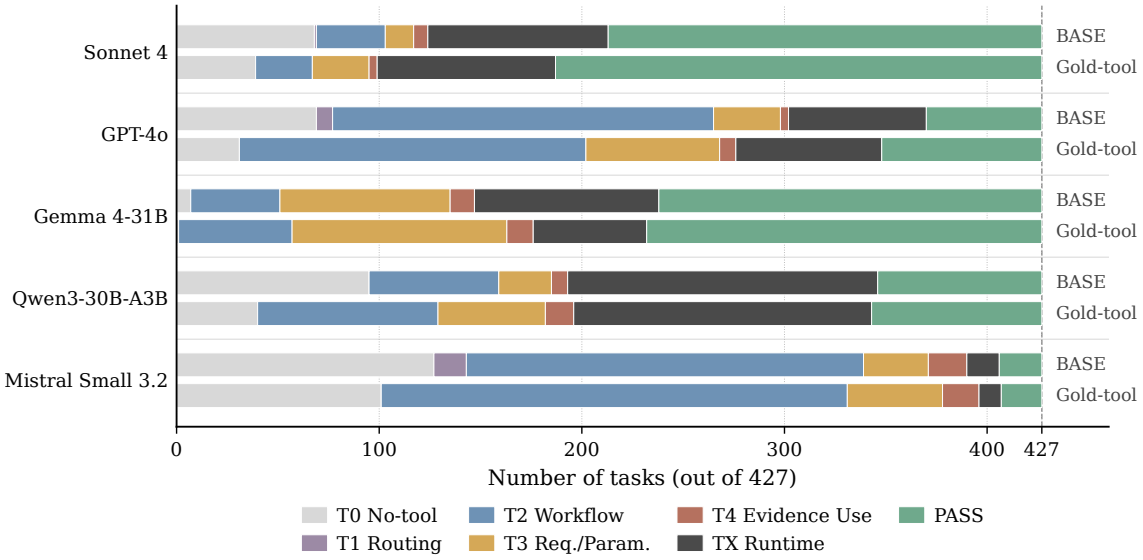

Figure 2. Stage-level outcome distributions under BASE and gold-tool access. Gold-tool access removes routing failures by construction, but failures remain concentrated in downstream execution stages, including workflow continuation, request/parameter grounding, evidence use, and runtime behavior.

T4 requires full reference-tool coverage and answer-bearing evidence in the observations, it is a conservative estimate of synthesis-stage failures. Even under this conservative definition, some agents retrieve relevant evidence but fail to use it in the final answer.

Overall, the stage-level breakdown reveals model-specific failure signatures that end-to-end pass rates alone would hide. GPT-4o and Mistral are dominated by partial workflow completion, Gemma by request/parameter grounding, Qwen by runtime instability, and Sonnet by a mixture of high pass rate and persistent runtime failures. More importantly, the failures that remain under gold-tool access are concentrated in downstream execution stages. Reliable MCP tool use therefore requires more than selecting the relevant tool set: agents must continue the workflow, ground each request, use returned evidence, and remain stable during execution.

4.3. Guidance Effects Depend on Timing and Content

We next examine whether guidance can repair the downstream failures observed above. Unlike gold-tool access, which only restricts the available tools, guidance provides information about missing evidence and, in some variants, relevant tools. We interpret these interventions as diagnostic probes rather than deployable agents: their role is to reveal which stages are sensitive to evidence or tool-path information, and which failures are recovered, shifted, or left unresolved.

Figure 3 shows that guidance effects depend strongly on timing and content. Proactive guidance is generally more effective than reactive guidance. Qwen3-30B-A3B gains +14.29 percentage points under Proactive Integrated guidance, and Mistral Small 3.2 gains +6.32 points. Gemma

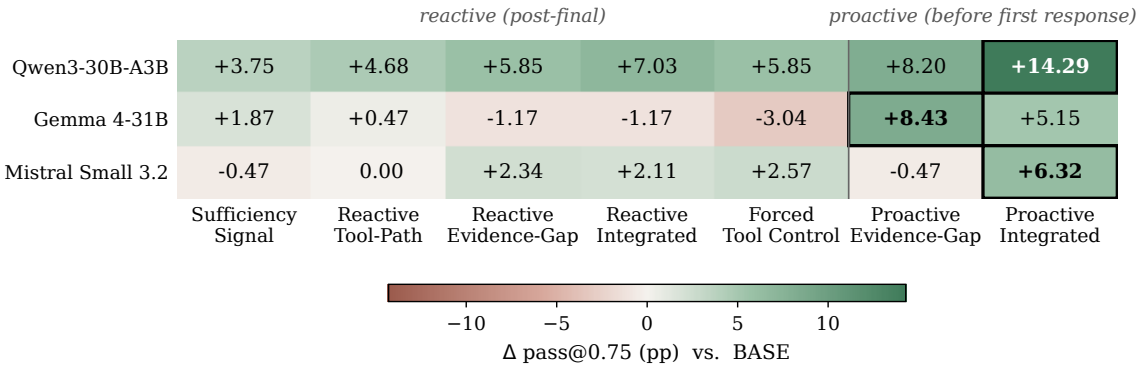

Figure 3. Change in pass@0.75 relative to BASE across guidance conditions. Proactive guidance generally yields larger gains than reactive guidance, but the best content differs by model. Qwen and Mistral benefit most from integrated evidence-gap and tool-path guidance, whereas Gemma benefits most from evidence-gap guidance alone.

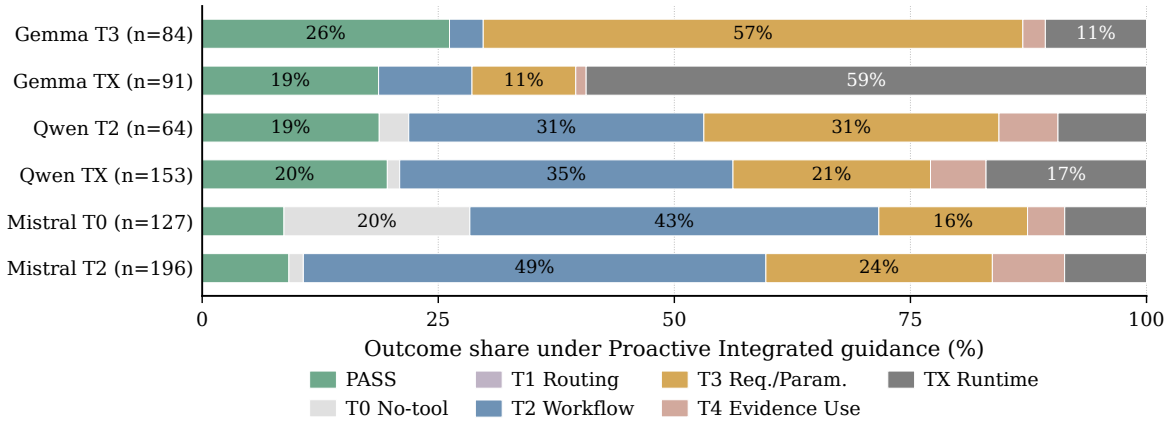

Figure 4. Selected failure-bucket migrations from BASE to Proactive Integrated guidance. Each row shows how tasks in a given BASE failure bucket are redistributed under Proactive Integrated guidance. Guidance recovers some failures to PASS, but many tasks remain in the same bucket or shift to downstream execution failures.

4-31B shows a different pattern: it performs best under Proactive Evidence-Gap guidance, gaining +8.43 points, while adding tool-path information gives a smaller improvement. This suggests that useful guidance is model-dependent. Qwen and Mistral benefit from explicit tool-path information, whereas Gemma benefits more from knowing which evidence is missing.

The timing effect indicates that many failures are not fixed well by post-hoc continuation alone. Reactive guidance is injected only after an insufficient final answer, whereas proactive guidance can influence the first few tool-use decisions before the agent commits to an incomplete path or premature final answer.

Pass-rate gains alone do not reveal how failures change. Figure 4 analyzes same-task bucket migrations from BASE to Proactive Integrated guidance. The main pattern is that only a minority of failures become PASS. For Gemma, 26% of BASE T3 Request/Parameter failures become PASS, but 57% remain T3. Similarly, 59% of Gemma TX Runtime failures remain TX. Thus, even when the model receives information about missing evidence, request grounding and runtime instability remain difficult to repair.

Qwen and Mistral show that guidance often shifts failures rather than removing them. For Qwen, many BASE T2 and TX failures move into T2 or T3, suggesting that guidance can reduce some unproductive behavior but still leave the agent unable to complete the workflow or issue grounded requests. Mistral shows a clear downstream shift: many BASE T0 No-tool failures move to T2 Workflow under Proactive Integrated guidance. In other words, strong guidance can make Mistral start using tools, but it often still stops before completing the required tool sequence. Mistral's BASE T2 failures are also sticky, with nearly half remaining T2 after guidance.

Forced Tool Control provides a complementary probe. It forces an unused relevant tool to be called in the next turn after reactive feedback, aiming to push the agent toward a tool-use direction it did not consider on its own. However, Forced Tool Control is not the best condition for any model. It improves Qwen and Mistral modestly, remains below Proactive Integrated guidance, and hurts Gemma relative to BASE. This suggests that opening a new tool path can help in some cases, but a single forced tool call does not solve downstream execution. After the forced call, the agent still needs to formulate correct arguments, continue the remaining workflow, decide whether enough evidence has been collected, and use the observations in the final answer.

Overall, guidance improves MCP agent behavior, but its effect is stage-limited. Proactive guidance can improve early tool-use decisions, yet many failures remain in workflow continuation, request/parameter grounding, and runtime behavior. Reliable MCP tool use therefore requires more than selecting or suggesting the next relevant tool: agents must sustain multi-step execution and ground each tool call in the evidence required by the task.

5. Trace-Level Analysis

Aggregate results show that gold-tool access and some guidance interventions improve agent performance, but they do not explain why failures remain after relevant tools are available. We therefore inspect paired task traces across BASE, gold-tool, and Proactive Integrated guidance.

Table 2 shows a representative Mistral case. The task asks for the publication year of an article on G-quadruplex propensity, the programming language of Lacroix's repository named after the algorithm in that article, and the last repository update before 2025. Solving it requires connecting evidence across PubMed and GitHub: the agent must

Table 2. Representative trace case. The same relevant tool families can lead to failure or success depending on query precision and evidence linking.

Condition	Outcome	Cov.	Turns/Calls	Tool families	Key behavior
BASE	Fail	0.000	7 / 6	GitHub, PubMed	Retrieves the PubMed article, but repeats broad GitHub queries such as `G4Hunter lacroix` and fails to identify the repository.
Gold-tool	Fail	0.000	2 / 1	GitHub	Makes one GitHub query, `lacroix g-quadruplex`, gets no result, and stops without trying the remaining PubMed path.
Proactive Integrated	Pass	1.000	2 / 3	GitHub, PubMed	Uses masked evidence slots and tool-path guidance to query PubMed, search `user:LacroixLaurent G4Hunter`, and inspect commits in parallel.

identify the article, infer the algorithm name, find the corresponding GitHub repository, and inspect its metadata and commit history.

In BASE, the agent does call the relevant tool families. It retrieves the PubMed article and the publication year, but then repeatedly searches GitHub with broad queries such as family-name or keyword searches. After failing to find the repository, it asks the user for more information and stops. The failure is therefore not simply a wrong-tool problem. The agent finds part of the evidence but fails to turn it into a precise repository query.

The gold-tool condition also fails. Even with the relevant tools available, the agent makes one GitHub query, receives an empty result, and stops without trying the PubMed path. In contrast, Proactive Integrated guidance gives the model masked evidence slots and a tool-path hint before the first response. The agent then queries PubMed, searches GitHub with the precise repository query, and checks the commit history in parallel, recovering all required values.

This case shows why tool access alone is not sufficient. BASE and Proactive Integrated use the same two tool families, GitHub and PubMed, but only the guided run succeeds. The difference is not the tool set, but the order of evidence acquisition, the precision of the GitHub query, and the ability to connect article-level evidence to repository-level evidence. Gold-tool access also does not prevent premature stopping: the agent still gives up after an empty first query instead of reformulating the request.

Although this example shows a successful guided recovery, Proactive Integrated guidance is not a complete solution. Appendix D provides a detailed breakdown of recurring failure sub-patterns within each bucket, highlighting the remaining challenges in workflow continuation, request grounding, evidence use, and runtime stability.

6. Discussion

6.1. Design Implications

Our results suggest that improving MCP agent reliability requires more than better tool retrieval or a narrower tool set. Gold-tool access reduces ambiguity in tool selection, but it does not ensure that the agent will write the right query, connect evidence across multiple tool calls, or continue the workflow until enough evidence has been collected. The trace-level analysis also shows that agents can use the same relevant tool families but still diverge depending on query precision and evidence linking.

First, agents need evidence-state tracking with completion verification. An agent should keep track of which pieces of evidence are required for the task and which tool observations support them. Before producing a final answer, it should verify whether all required evidence has actually been grounded. This directly addresses T2 Workflow failures, where agents collect only part of the evidence and then stop too early.

Second, agents need query repair and request validation. When a search returns an empty or incomplete result, the agent should not immediately stop. Instead, it should revise the query using entities, dates, identifiers, or other clues found in earlier observations. This is especially important for T3 Request/Parameter failures: even with the right tool, a task can fail if the query, filter, entity, date, or aggregation scope is wrong. Agents should therefore check whether each tool request matches the evidence requirement before or after making the call.

Future MCP agent designs should therefore focus less on stronger tool-selection control alone and more on evidence-state management, request-level repair, and workflow completion verification.

6.2. Limitations

This study has several limitations. First, our guidance interventions are diagnostic probes rather than deployable agent methods. Proactive Integrated guidance uses masked evi-

dence slots and reference tool names, which are not directly available in ordinary deployment settings. We therefore use guidance to study how failures change, not to propose a new prompting method.

Second, our failure taxonomy assigns each failed trajectory to a single primary label. This makes the analysis reproducible and easy to compare across models, but some failures can involve multiple stages. For example, an agent may stop early while also using a poorly grounded query, or retrieve partial evidence but fail to express it correctly in the final answer. Evidence Use failures may also be undercounted because we assign this label only when the required evidence clearly appears in tool observations but is missing from the final answer.

Third, our experiments are based on MCP-Atlas. The benchmark is well suited to our goal because it includes real MCP servers, heterogeneous tools, and multi-step tasks that require agents to gather and combine evidence across tool calls. However, it is still one benchmark with a fixed set of tasks, servers, and evaluation rules. Failure distributions may differ in other MCP environments with longer workflows, noisier tools, or different agent harnesses.

Taken together, our results show that reliable MCP tool use requires execution-level support beyond tool retrieval or tool-choice control. We hope this stage-aware view helps future work move beyond end-to-end success rates and toward more precise diagnosis and repair of agentic tool-use failures.

7. Conclusion

This paper diagnoses where MCP tool-using agents fail. We introduced a stage-aware taxonomy that separates tool-selection failures from downstream execution failures, including Workflow, Request/Parameter, Evidence Use, and Runtime failures. Across five agents on MCP-Atlas, gold-tool access improves some models but does not resolve most failures. Many agents still stop after partial workflows, issue poorly grounded requests, miss retrieved evidence, or enter unstable runtime behavior even when relevant tools are available.

We further used guidance interventions as diagnostic probes. Proactive guidance generally improves performance more than reactive post-final feedback, showing that early evidence and tool-path signals can change the agent's trajectory before it commits to an incomplete path. However, guidance is not a complete solution. Many failures remain in the same stage or migrate into downstream stages, indicating that suggesting or forcing relevant tools does not guarantee correct execution.

Taken together, our results show that reliable MCP tool use requires more than better tool retrieval or stronger tool-choice control. Agents need mechanisms to track missing evidence, repair failed or imprecise requests, continue multi-step workflows, and verify that retrieved observations support the final answer. We hope this stage-aware view helps future work move beyond end-to-end success rates and toward more precise diagnosis and repair of agentic tool-use failures.

References

Anthropic. Introducing the model context protocol. https://www.anthropic.com/news/model-context-protocol, 2024. Accessed: 2026-05-09.

Anthropic. System card: Claude Opus 4 & Claude Sonnet 4, 2025. Anthropic system card.

Bandi, C., Hertzberg, B., Boo, G., Polakam, T., Da, J., Hassaan, S., Sharma, M., Park, A., Hernandez, E., Rambado, D., Salazar, I., Cruz, R., Rane, C., Levin, B., Kenstler, B., and Liu, B. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

Cemri, M., Pan, M. Z., Yang, S., Agrawal, L. A., Chopra, B., Tiwari, R., Keutzer, K., Parameswaran, A., Klein, D., Ramchandran, K., Zaharia, M., Gonzalez, J. E., and Stoica, I. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Chen, Z., Du, W., Zhang, W., Liu, K., Liu, J., Zheng, M., Zhuo, J., Zhang, S., Lin, D., Chen, K., and Zhao, F. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Google. Gemma 4 model overview, 2026. Google AI for Developers documentation.

Li, M., Zhao, Y., Yu, B., Song, F., Li, H., Yu, H., Li, Z., Huang, F., and Li, Y. Api-bank: A comprehensive benchmark for tool-augmented llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., Huang, M., Dong, Y., and Tang, J. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang,

Y., Welleck, S., Majumder, B. P., Hermann, K., Yazdanbakhsh, A., and Clark, P. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

Mistral AI. Mistral Small 3.2 24B Instruct 2506, 2025. Hugging Face model card.

OpenAI. GPT-4o system card, 2024. arXiv:2410.21276.

Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, 2024.

Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Ruan, Y., Dong, H., Wang, A., Pitis, S., Zhou, Y., Ba, J., Dubois, Y., Maddison, C. J., and Hashimoto, T. Identifying the risks of lm agents with an lm-emulated sandbox. In *International Conference on Learning Representations*, 2024.

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

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Trivedi, H., Khot, T., Hartmann, M., Manku, R., Dong, V., Li, E., Gupta, S., Sabharwal, A., and Balasubramanian, N. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

xAI. Grok 3 Beta: The age of reasoning agents, 2025. xAI announcement.

Yang, A., Li, A., Yang, B., et al. Qwen3 technical report, 2025. arXiv:2505.09388.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Yao, S., Shinn, N., Razavi, P., and Narasimhan, K. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Bisk, Y., Fried, D., Alon, U., and Neubig, G. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

A. Failure Attribution Rules

The main text defines the failure taxonomy at a conceptual level. Here we describe the deterministic attribution rules used to assign one primary label to each failed trajectory. The goal is not to claim that each failure has only one cause, but to make the labeling procedure reproducible across models and experimental conditions.

For each task, we compare the model trajectory with the reference trajectory. We extract the set of reference tool families F_{ref} and the set of tool families called by the model F_{called}. We define tool-family coverage as

$$\text{coverage}_{tool} = \frac{|F_{\text{called}} \cap F_{\text{ref}}|}{|F_{\text{ref}}|}.$$

This measures whether the agent entered the required tool path, without requiring exact agreement at the individual tool-function level.

Each failed trajectory is assigned a single label using the priority rules in Table 3. Passing runs are reported separately as PASS. Runtime-dominated trajectories, such as timeouts, repeated loops, context overflows, max-turn failures, or tool/server errors, are assigned to TX. The remaining trajectories are labeled based on tool use, overlap with reference tool families, tool-family coverage, and whether the trajectory retrieves and uses the evidence required by the benchmark claims.

Table 3. Operational attribution rules for assigning one primary failure label to each failed trajectory.

Bucket	Operational rule
T0 No-tool	No tool call is made before the run ends.
T1 Routing	The agent calls tools, but none of the called tool families overlap with the reference tool families.
T2 Workflow	The agent calls at least one relevant tool family, but does not cover the reference tool-family set. These runs usually correspond to partial workflows, premature stopping, or asking the user for missing information instead of continuing with available tools.
T3 Request/Parameter	The agent covers the reference tool families, but the required evidence is not correctly retrieved because of a poorly grounded query, filter, entity, date, identifier, aggregation scope, or request format.
T4 Evidence Use	The agent retrieves the evidence required for the task, but the final answer omits, misuses, or fails to synthesize that evidence.
TX Runtime	The run is dominated by timeout, repeated tool loops, max-turn termination, context overflow, or tool/server/runtime errors.

Because the taxonomy assigns only one primary label, it should be interpreted as dominant-stage attribution rather than a complete causal explanation. Some trajectories contain multiple errors. For example, an agent may stop early after issuing a weak query, or retrieve partial evidence but fail to express it correctly in the final answer. In such cases, the assigned label identifies the main stage where the trajectory becomes insufficient under our deterministic rules.

The Evidence Use category is conservative. We assign T4 only when the trace shows that the required evidence was retrieved but was not properly reflected in the final answer. This avoids treating incomplete workflows as synthesis failures, but it also means that some evidence-use errors may be counted under Workflow or Request/Parameter when the evidence was only partially retrieved.

B. Guidance Intervention Details

This appendix describes how the guidance interventions in Section 3.4 and Section 4.3 are constructed. The interventions are used as diagnostic probes rather than as a new deployment method. Their purpose is to test how guidance timing and content affect MCP agent failures while preventing the executing agent from seeing the actual answer values.

B.1. Masked Evidence Slots and Answer-Value Control

Each MCP-Atlas task provides a set of reference answer requirements: atomic facts that the final answer should satisfy. These requirements contain the true answer values, so they cannot be shown directly to the executing agent. To build safe guidance messages, we convert each reference requirement into a masked evidence slot.

A masked evidence slot keeps the structure of the required information while replacing answer values with `[value]`. For example, the reference requirement

> AC Milan won their last Champions League title in 2007.

is converted into

> AC Milan won their last Champions League title in `[value]`.

This tells the agent what information must be retrieved, but not the value it must output. Values already specified in the user request are kept, since they are task context rather than hidden answers.

We generate masked evidence slots offline using Claude Opus 4.6. After automatic masking, a human manually checks each masked requirement to verify that answer values have been removed and that the remaining text still describes the required evidence. Throughout all experiments, the original reference answer requirements are used only for final evaluation and hidden support checks. The executing agent receives only masked evidence slots. Thus, guidance can indicate what information is missing without directly revealing the answer.

B.2. Guidance Conditions

We evaluate seven guidance conditions. They vary along two axes: when guidance is given and what information it contains. Reactive guidance is given after the agent attempts an insufficient final answer. Proactive guidance is inserted before the first model response. Guidance content ranges from a minimal "not complete" signal to masked evidence slots, unused reference tool names, or both.

Table 4. Guidance conditions used as diagnostic probes. Missing-evidence guidance provides masked evidence slots. Tool-path guidance provides unused reference tool names without arguments or execution order. Forced Tool Control fixes the next tool name, while leaving the arguments to the model.

Condition	Timing	Missing evidence	Tool information
Sufficiency Signal	Reactive	No	No
Reactive Tool-Path	Reactive	No	Tool names
Reactive Evidence-Gap	Reactive	Yes	No
Reactive Integrated	Reactive	Yes	Tool names
Forced Tool Control	Reactive	Yes	Forced next tool
Proactive Evidence-Gap	Proactive	Yes	No
Proactive Integrated	Proactive	Yes	Tool names

Tool names are provided only as names of unused reference tools. We do not provide tool arguments, the correct call order, retrieved evidence values, or the final answer. Thus, even in the integrated conditions, the agent must still decide how to call tools, how to continue the workflow, and how to use the observations.

All guidance conditions use the same task set, agent loop, maximum-turn limit, and timeout setting. They differ only in feedback timing, feedback content, and whether the next tool name is forced.

B.3. Reactive and Proactive Injection

Reactive guidance is triggered when the agent tries to finish without enough tool-supported evidence. When the agent produces a final answer instead of a tool call, a hidden support check compares the tool observations collected so far against the reference answer requirements. If required information is still missing, the system appends a feedback message to the conversation and lets the agent continue. Each task can receive at most one reactive guidance message.

Proactive guidance follows a different procedure. Instead of waiting for an incomplete final answer, the feedback message is inserted before the first model response. The agent therefore starts the task with masked evidence slots and, in the integrated variant, with a small set of unused reference tool names. No additional post-final support check is performed in the proactive setting; the initial message is the intervention.

This distinction is important for interpretation. Reactive guidance tests whether an agent can recover after it has already attempted to stop. Proactive guidance tests whether giving the agent missing-evidence information at the start changes its early tool-use decisions.

B.4. Feedback Templates and Forced Tool Control

All guidance messages are built from four simple templates. The templates differ only in what information is shown to the agent.

Table 5. Feedback templates used to construct guidance messages.

Template	Information shown to agent	Used by
`minimal`	Incomplete-answer signal	Sufficiency Signal
`evidence-only`	Masked evidence slots	Evidence-Gap conditions
`tool-only`	Unused reference tool names	Reactive Tool-Path
`integrated`	Evidence slots + tool names	Integrated conditions

The minimal template only tells the agent that the answer is not complete. The evidence-only template lists masked evidence slots and instructs the agent not to guess values. The tool-only template lists unused reference tool names. The integrated template combines masked evidence slots and unused tool names.

A representative integrated feedback message is:

```
[Observation] The current answer is not yet supported by sufficient tool evidence.
Missing evidence slots:
- The walking time from The Salty Donut to Vizcaya Museum is [value].
Consider using these tools (not yet called):
- ddg-search_search
- ddg-search_fetch_content
Do not guess values; retrieve them from tools.
```

This message tells the agent what evidence is missing and which tools may be useful, but it does not reveal the answer values, the tool arguments, or the correct execution order.

Forced Tool Control uses the same integrated feedback template as Reactive Integrated guidance, but additionally forces the next tool name through the tool-choice interface. This tests whether an agent can be moved into a relevant tool-use path that it did not select on its own. Only the next tool name is forced. The model still generates the arguments, interprets the observation, decides whether more tool calls are needed, and produces the final answer. Therefore, Forced Tool Control isolates the effect of opening a new relevant tool path while leaving request grounding, workflow continuation, evidence use, and final synthesis to the agent.

C. Additional Quantitative Results

This appendix reports the full numerical results behind the aggregate figures in Section 4. All tables use the same 427 MCP-Atlas tasks and the same pass threshold, $\theta = 0.75$, as the main text.

C.1. Full BASE and Gold-Tool Results

Table 6 reports the full outcome counts under BASE and gold-tool access. Gold-tool access removes routing failures by construction, but it does not eliminate downstream failures. In particular, T2 Workflow and T3 Request/Parameter failures remain frequent across models, and TX Runtime failures remain large for Qwen and Sonnet.

Table 6. Full BASE and gold-tool results. Counts are over 427 tasks. Pass is computed at $\theta = 0.75$.

Model	Arm	T0	T1	T2	T3	T4	TX	PASS	Pass	Mean cov.
Sonnet 4	BASE	68	1	34	14	7	89	214	50.12	0.592
Sonnet 4	Gold-tool	39	0	28	28	4	88	240	56.21	0.653
GPT-4o	BASE	69	8	188	33	4	68	57	13.35	0.270
GPT-4o	Gold-tool	31	0	171	66	8	72	79	18.50	0.336
Gemma 4-31B	BASE	7	0	44	84	12	91	189	44.26	0.561
Gemma 4-31B	Gold-tool	1	0	56	106	13	56	195	45.67	0.569
Qwen3-30B-A3B	BASE	95	0	64	26	8	153	81	18.97	0.300
Qwen3-30B-A3B	Gold-tool	40	0	89	53	14	147	84	19.67	0.326
Mistral Small 3.2	BASE	127	16	196	32	19	16	21	4.92	0.155
Mistral Small 3.2	Gold-tool	101	0	230	47	18	11	20	4.68	0.143

C.2. Full Guidance Ablation Results

Table 7 reports the full guidance ablation results for the three models used in the intervention study. Guidance effects vary substantially by model. Qwen and Mistral obtain their largest gains from Proactive Integrated guidance, whereas Gemma performs best with Proactive Evidence-Gap guidance. Forced Tool Control is not the best condition for any model, indicating that forcing a relevant next tool does not by itself solve downstream execution failures.

Table 7. Full guidance ablation results. Δ values are computed relative to BASE for each model.

Model	Condition	Pass	Δ pass	Mean cov.	Δ cov.
Qwen3-30B-A3B	BASE	18.97	0.00	0.300	0.000
Qwen3-30B-A3B	Sufficiency Signal	22.72	+3.75	0.325	+0.025
Qwen3-30B-A3B	Reactive Tool-Path	23.65	+4.68	0.338	+0.038
Qwen3-30B-A3B	Reactive Evidence-Gap	24.82	+5.85	0.360	+0.060
Qwen3-30B-A3B	Reactive Integrated	26.00	+7.03	0.355	+0.055
Qwen3-30B-A3B	Forced Tool Control	24.82	+5.85	0.357	+0.057
Qwen3-30B-A3B	Proactive Evidence-Gap	27.17	+8.20	0.371	+0.071
Qwen3-30B-A3B	Proactive Integrated	33.26	+14.29	0.443	+0.143
Gemma 4-31B	BASE	44.26	0.00	0.561	0.000
Gemma 4-31B	Sufficiency Signal	46.14	+1.87	0.543	-0.019
Gemma 4-31B	Reactive Tool-Path	44.73	+0.47	0.531	-0.031
Gemma 4-31B	Reactive Evidence-Gap	43.09	-1.17	0.535	-0.027
Gemma 4-31B	Reactive Integrated	43.09	-1.17	0.516	-0.045
Gemma 4-31B	Forced Tool Control	41.22	-3.04	0.501	-0.060
Gemma 4-31B	Proactive Evidence-Gap	52.69	+8.43	0.602	+0.041
Gemma 4-31B	Proactive Integrated	49.41	+5.15	0.586	+0.025
Mistral Small 3.2	BASE	4.92	0.00	0.155	0.000
Mistral Small 3.2	Sufficiency Signal	4.45	-0.47	0.127	-0.028
Mistral Small 3.2	Reactive Tool-Path	4.92	0.00	0.135	-0.020
Mistral Small 3.2	Reactive Evidence-Gap	7.26	+2.34	0.170	+0.015
Mistral Small 3.2	Reactive Integrated	7.03	+2.11	0.171	+0.016
Mistral Small 3.2	Forced Tool Control	7.49	+2.57	0.180	+0.025
Mistral Small 3.2	Proactive Evidence-Gap	4.45	-0.47	0.149	-0.006
Mistral Small 3.2	Proactive Integrated	11.24	+6.32	0.228	+0.073

D. Failure Sub-Patterns within Each Bucket

This appendix further breaks down each failure bucket into recurring trace-level sub-patterns. The main text assigns one primary label to each failed trajectory, but trajectories within the same bucket can fail in different ways. To clarify the behavioral meaning of each bucket, we inspected sampled BASE traces across models and grouped recurring patterns qualitatively. These sub-patterns are not used as an additional quantitative taxonomy; they are intended to make the stage-level buckets easier to interpret.

Table 8. Failure sub-patterns observed within each taxonomy bucket.

Bucket	Sub-pattern	Typical trace signature
T0 No-tool	**T0a** Refuses without trying	The agent never calls a tool and instead says that it cannot access the needed information.
	T0b Asks user for data	The agent asks the user to provide a list, address, file, or database before making any tool call.
	T0c Only writes a plan	The agent describes the steps it would take, but does not execute any tool call.
	T0d Empty completion	The run ends without a substantive answer or without entering the tool-use loop.
T1 Routing	**T1a** Wrong tool type	The agent uses a generic web search or unrelated tool when a task-specific data or API tool is required.
	T1b Lists options, then asks	The agent lists candidate databases, namespaces, or collections, but asks the user which one to use.
	T1c No fallback after error	After one failed call to an irrelevant tool family, the agent stops without trying another tool path.
T2 Workflow	**T2a** Stops mid-workflow	The agent resolves early hops, then stops before completing the remaining evidence chain.
	T2b Trusts empty result	A query returns no records, and the agent reports the empty result as the answer without reformulating.
	T2c Trusts first result	The agent accepts the top search hit without checking whether it satisfies the task constraints.
	T2d Gives up after tool error	A tool, quota, or fetch error occurs, and the agent stops instead of retrying or changing strategy.
	T2e Fills gap by guessing	The agent gathers partial evidence but invents the missing part instead of continuing the workflow.
T3 Req./Param.	**T3a** Wrong date or time	The agent reaches the right data source but applies an off-by-one date, wrong year, or wrong time window.
	T3b Wrong entity	The agent uses a near-miss entity from search results instead of the correct one.
	T3c Bad query meaning	The agent uses the right tool family but issues a semantically invalid filter or query.
	T3d Wrong aggregation	The agent retrieves relevant data but aggregates it over the wrong period or scope.
	T3e Bad calculation	The agent reaches the right tool for a required calculation, but the calculation request is mis-specified, so the required value is never correctly retrieved.
	T3f Guesses after empty search	After an empty or unhelpful result, the agent answers from unsupported prior knowledge instead of repairing the request.
T4 Evidence Use	**T4a** Omits evidence	The observations contain the required multi-claim evidence, but the final answer reports only part of it.
	T4b Stops before synthesis	The agent reaches the needed evidence but asks a clarification question or stops without completing the answer.
	T4c Uses wrong field	The agent retrieves a structured record but uses the wrong row, column, or field in the final answer.
	T4d Bad final formatting	The evidence is present, but the final answer miscomputes or misformats the retrieved values.
TX Runtime	**TXa** Repeated loop	The agent repeats the same or near-identical tool call and receives similar observations until the run budget is exhausted.
	TXb Schema–query loop	The agent alternates between schema inspection and querying without making task progress.
	TXc Repeated API errors	The agent repeatedly retries after server, quota, or API failures without changing strategy.
	TXd Empty or plan-only final	The agent reaches the turn limit and produces either no answer or only a plan.
	TXe Generic web-search loop	The agent keeps searching generic web content instead of switching to the task-specific data source.

T0 No-tool. T0 failures occur before tool selection becomes meaningful. In these traces, the agent never enters the tool-use loop: it refuses, asks the user for data, writes only a plan, or produces no substantive output. This separates no-tool failures from routing failures, where the agent at least attempts tool use.

T1 Routing. T1 failures are the closest category to the conventional wrong-tool-selection explanation. The agent calls a tool, but enters an irrelevant tool family or stops after an unhelpful first tool attempt. Gold-tool access can directly remove this bucket, but the remaining sections show that many failures occur after routing has already been resolved.

T2 Workflow. T2 failures capture incomplete execution after a relevant tool path has been entered. The common pattern is not total failure, but partial progress followed by premature stopping, over-trusting an empty or top-ranked result, giving up after an error, or filling missing evidence by guessing. This bucket shows why relevant tool access does not imply workflow completion.

T3 Request/Parameter. T3 failures occur when the right tool family is reached but the request itself is poorly grounded. These traces expose the gap between selecting the correct tool and formulating the correct query, filter, entity, date, aggregation scope, or calculation. This is a central post-selection bottleneck.

T4 Evidence Use. T4 failures occur after the required evidence is already present in tool observations. The problem is no longer retrieval but final-use: the agent omits evidence, stops before synthesis, selects the wrong field, or misformats retrieved values. Since our T4 attribution is conservative, this bucket should be read as a lower bound on evidence-use failures.

TX Runtime. TX failures reflect control-level breakdowns rather than ordinary semantic mistakes. Repeated tool loops, schema–query ping-pong, API-error retries, and empty or planning-only final answers indicate that agents need runtime recovery and strategy switching in addition to better reasoning.

Summary. The sub-pattern view clarifies what the aggregate taxonomy hides. T2 failures are often partial workflows that stop too early; T3 failures are request-grounding failures after the right tool path is reached; T4 failures are synthesis failures after evidence has been retrieved; and TX failures are runtime or control failures. Together, these patterns reinforce the main conclusion: reliable MCP agents require workflow continuation, request repair, evidence-state tracking, and runtime recovery beyond selecting the right tool.