# TOOLTURNS: BENCHMARKING LLM AGENTS IN MULTI-TURN DIALOGUE

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Tool-using language models are increasingly deployed in conversational settings, yet most large-scale tool-use benchmarks focus on isolated, single-turn requests. We introduce ToolTurns, a benchmark for evaluating language agents end-to-end in multi-turn dialogue with tools. It contains 2,620 problems spanning 1,827 tools and 11,004 API endpoints across 47 categories. Each problem begins with an intentionally underspecified request and unfolds into three user queries, yielding 7,860 independently evaluated subtasks that require agents to seek clarification, maintain conversational context, select tools, and respond to follow-up requests. We additionally introduce an open-source user simulator that reveals withheld context through dialogue and generates context-dependent follow-ups, together with a checker–fixer pipeline for improving simulated user turns. Across nine language models, the best agent resolves 69.6% of individual subtasks but completes only 39.8% of full three-query dialogues, and every evaluated model performs worse on follow-up queries than on the initial request. We further find that dialogue-quality rubric scores and task completion can rank models very differently, highlighting the importance of evaluating whether users' underlying requests are actually resolved. Finally, paired turn-level evaluation shows that our revision pipeline reduces automatically detected dialogue defects by 41.2%, while blind human evaluation prefers revised turns in 63.3% of cases. These results expose substantial room for improvement in reliable, multi-turn tool use and provide a scalable testbed for measuring progress.

## 1 INTRODUCTION

A tool-using assistant must do more than answer an initial request. A user asking "What games are happening this weekend?" may omit the sport or location, then ask follow-up questions about the results. The assistant must elicit missing information, use the available APIs, and carry relevant context into subsequent requests. These decisions shape the conversation itself: an assistant's clarification or answer changes what the user says next. Evaluating this behavior therefore requires following the agent through an interaction, including the requests that arise after its first answer.

Existing benchmarks capture different aspects of this challenge (Table 1). ToolBench (Qin et al., 2024) provides broad API coverage, while ToolDial (Shim et al., 2025) evaluates conversational tool-use capabilities from recorded dialogue histories. Interactive benchmarks such as $\tau$-bench (Yao et al., 2024) and ToolSandbox (Lu et al., 2025) evaluate agents within stateful application environments; $\tau^2$-bench (Barres et al., 2025) further tests coordination when users can also act on the environment. We study the combination of broad API coverage and adaptive, end-to-end dialogue evaluation: whether agents can clarify underspecified requests and resolve related follow-ups across diverse tool sets.

We introduce ToolTurns, a benchmark of 2,620 initially underspecified problems spanning 1,827 tools and 11,004 API endpoints across 47 RapidAPI categories (§ 3; statistics in Table 2). Each problem exposes a fixed subset of endpoints—26.5 on average—and unfolds into an initial request followed by two adaptively generated follow-up requests. A user simulator answers clarification questions using withheld context and generates follow-ups from the user-visible conversation. Agents interact with these users and a StableToolBench-based tool environment (Guo et al., 2024) (a cached, LLM-simulated execution layer for the ToolBench APIs), producing 7,860 separately scored subtasks



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

per agent. This design combines API breadth with evaluation along each agent's own conversational trajectory.

Because the simulated user helps determine the evaluation trajectory, we also examine the quality of its contributions. We develop a checker–fixer pipeline that revises candidate user messages before they enter the conversation (§ 3.3). In a paired analysis, revision reduces rule-detected defects by 41.2% among the messages it rewrites (536 of 1,272 candidates); blind human annotation prefers the revised message in 63.3% of 120 sampled pairs (Appendix B.2.3, Appendix B.3). We evaluate task completion separately from dialogue quality, reporting both the fraction of requests resolved and the fraction of dialogues in which all three requests succeed.

Across nine proprietary and open-weight agent models, our evaluator reports a highest subtask success rate of 69.6% and a highest full-dialogue completion rate of 39.8% (§ 4). Every evaluated model achieves lower success on both follow-ups than on the initial request. Task-success rankings also differ from rankings under the dialogue-quality rubric. These findings highlight the importance of evaluating completion throughout a conversation and measuring it directly alongside diagnostic quality scores.

**Our contributions are:**

1. ToolTurns, a benchmark combining broad API coverage with adaptive dialogue: 2,620 problems with clarification exchanges and follow-up requests evaluated along the agent's own trajectory.
2. A user-simulation pipeline with targeted message revision, evaluated through paired automatic measurements and blind human annotation.
3. An evaluation of nine agents that characterizes request-level and full-dialogue completion, follow-up performance, and differences between task-success and dialogue-quality rankings.

## 2 Related Work

The rise of tool-calling LLMs has led to two key directions in evaluation which intersect our work: scaling to large, diverse sets of APIs, and supporting tool use in interactive, multi-turn dialogues.

**Scaling Tool Evaluation** Where early tool-calling evaluations focused on relatively small sets of tools (Tang et al., 2023), recent efforts have aimed to scale this to larger, more diverse sets of tools (Patil et al., 2023; Qin et al., 2024; Elder et al., 2026). Qin et al. (2024) provide one of the largest such evaluations in ToolBench, evaluating LLMs on problems using over 16,000 APIs sourced from RapidAPI. [1] While comprehensive in tool scope, each problem is designed for only a single-turn interaction: missing real-world cases where the model must seek details or clarification from the user, or respond to follow-ups. Our work aims to address this limitation by introducing ToolTurns, a large-scale tool-calling evaluation for the multi-turn dialogue setting, where an agent must support both clarification-seeking and follow-up requests. We build our evaluation upon the RapidAPI tool set proposed in Qin et al. (2024).

**Interactive Tool-Use Benchmarks** Prior tool-use evaluations fall into three settings: broad API corpora, recorded conversational evaluations, and interactive stateful environments (Table 1). ToolBench gives one user instruction over a large RapidAPI catalogue, so later tool steps are not further user turns; ToolDial, ToolTalk, API-Bank, and DialogTool instead score recorded or annotated dialogues, executing tools only where the benchmark implements them (Qin et al., 2024; Shim et al., 2025; Farn & Shin, 2023; Li et al., 2023; Wang et al., 2025). $\tau$-bench, $\tau^2$-bench, ToolSandbox, and VitaBench keep an adaptive user inside a stateful application and grade environment outcomes or rubrics (Yao et al., 2024; Barres et al., 2025; Lu et al., 2025; He et al., 2026).

ToolTurns evaluates clarification and successive requests over a broad API catalogue. $\tau$-bench and $\tau^2$-bench emphasize domain policies, verifiable environment outcomes, and, for $\tau^2$ telecom, user–agent coordination. Our broader schema coverage does not establish stronger state modeling or more reliable ground-truth evaluation. Two related settings sit outside the table. Wang et al. (2024) evaluate multi-turn tool use in which the model executes Python tools and revises its solution using language feedback. Wu et al. (2024) study tool planning for multi-granularity instructions; those solution steps are successive tool calls for one instruction, not conversational turns.

[1] https://rapidapi.com/

| Benchmark | Resource size | Tool/API scope | User interaction during evaluation | Evaluation target |
|---|---|---|---|---|
| ToolBench | 126,486 instruction–solution pairs (corpus) | 3,451 tools; 16,464 APIs; 49 categories | Single user instruction; tool steps are not user turns | Pass rate and win rate (ToolEval) |
| ToolDial | 11,111 generated dialogues (corpus) | RapidAPI graph (4,474 APIs) | Recorded dialogue histories | Action, API-argument, and response prediction |
| ToolTalk | 78 annotated conversations | 28 callable tools | Fixed user turns; predicted calls executed with feedback | Conversation success from tool-call match |
| API-Bank | 314 annotated dialogues (evaluation) | 73 runnable tools | Annotated dialogue context | Planning, retrieval, and calls |
| $\tau$-bench | 115 retail tasks; 50 airline tasks | 15 retail and 13 airline API tools | Adaptive simulated user | Database outcome and repeated-trial reliability |
| $\tau^2$-bench | 115 retail; 50 airline; 114 telecom tasks | Agent tools: 13 retail, 12 airline, 13 telecom; 30 telecom user tools | Adaptive user; dual control in telecom | Verifiable task outcomes |
| ToolSandbox | 1,032 evaluation scenarios | 34 tools | Adaptive simulated user; stateful execution | Milestone and minefield checks |
| VitaBench | 400 tasks (100 cross-scenario; 300 single-scenario) | 66 tools | Interactive application environments | Rubric-based outcomes |
| DialogTool | 900 evaluation dialogues | 15 apps; 30 APIs (evaluation) | Fixed dialogue context; APIs execute in VirtualMobile | Creation, selection, execution, and response scores |
| ToolTurns | 2,620 problems; 3 scored requests each | 1,827 tools; 11,004 endpoints; 47 categories | Adaptive clarification and follow-up requests | Subtask and full-dialogue completion; dialogue-quality diagnostics (LLM judge) |

Table 1: How the evaluation setting differs across benchmarks. ToolTurns combines broad API coverage with adaptive clarification and follow-up dialogue, complementing benchmarks centered on stateful application workflows. Counting conventions for each row are in Appendix A.

# 3 Benchmark Design

## 3.1 Problem Design

A ToolTurns problem is a triple $(q_i, T_i, G_i)$, where $q_i$ is the user's initial natural-language query, $T_i$ is a fixed set of callable tool endpoints and their schemas, and $G_i = \{(s_{ij}, v_{ij})\}_{j=1}^{m_i}$ is the unseen user's goal, represented as a set of slot–value constraints. Each slot $s_{ij}$ corresponds to an argument in a tool schema, and $v_{ij}$ is the value that the user has in mind but does not initially specify in $q_i$.

The dialogue begins with $q_i$. At each step $t$, conditioned on the dialogue history and the schemas of $T_i$, the agent selects an action $a_t \in \mathcal{A}(T_i)$ that either (1) invokes an endpoint and receives a tool observation $o_t$, (2) sends a message to the user and receives a user response $o_t$, or (3) completes the current request with a final answer $y_t$. In response to a clarification request, the user simulator reveals relevant slot values from $G_i$.

| Metric | Value |
|---|---|
| Total problems | 2,620 |
| Single-tool | 1,032 (39.4%) |
| Multi-tool (2+) | 1,588 (60.6%) |
| Cross-category | 1,023 (64.4% of multi) |
| Distinct tools | 1,827 |
| Distinct API endpoints | 11,004 |
| RapidAPI categories | 47 |
| Tools / problem | 3.4 / 3.0 |
| Endpoints / problem | 26.5 / 15.0 |

Table 2: ToolTurns statistics for the 2,620-problem evaluation set. The last two rows report mean / median.

This design isolates two distinct skills: **tool selection and use** (the agent must combine endpoints from the available set to satisfy a request) and **clarification dialogue** (the agent must elicit the withheld context before it can call the right endpoint with the right parameters).

**Tool catalogue.** We use the RapidAPI tool collection released with ToolBench (Qin et al., 2024), in which each tool is represented by its available endpoints and parameter schemas. StableToolBench exposes the same collection during execution, allowing the sampled tool sets to be evaluated without schema conversion.

**Tool-set sampling.** We draw seed tool sets from ToolBench's three retrieval groups, which differ in how the underlying tools relate to one another: **G1** (single-tool sets), **G2** (intra-category multi-tool sets, where the tools share a RapidAPI category), and **G3** (intra-collection multi-tool sets, which span categories within a curated collection). Following ToolBench's own filtering, we discard seed sets with fewer than three endpoints. G1 always uses a single seed set per problem. For G2 and G3, with probability 0.5 we sample two distinct seed sets and combine their endpoints; this produces problems that may require tools from different categories.

**Endpoint expansion.** ToolBench's seed sets identify a small number of representative endpoints, but realistic requests may require other capabilities of the same tools. We therefore expand each sampled set to include all endpoints belonging to its referenced tools.

**Vague-query generation.** For each expanded tool set, we prompt GPT-4o-mini to produce two queries using a fixed template. The prompt instructs the model that queries must (a) read like a natural request a human might make, (b) not mention tool or API names, (c) be answerable by some combination of the provided endpoints, and (d) deliberately omit details that the agent would need to actually make a call. For each query the model additionally returns 2–4 **context keywords** encoding the omitted details as concrete values (e.g. `Paris`, `December`, `Lakers`) rather than type placeholders (e.g. `city`, `month`, `team_name`); the prompt gives explicit good/bad examples to enforce this. These keywords become the problem's withheld context.

**Example.** Problem 2610 in ToolTurns has the initial query:

> *"I heard there's a big game coming up soon. Can you tell me when it is?"*

Its withheld context is `NBA`, `Los Angeles`, `Lakers`, and `October`, and its tool set exposes 48 endpoints from an NBA data API, a web search API, and a geographic database. The agent must first clarify the sport, team, and time frame before selecting among the available endpoints. §F.1 shows how different agents handle this problem.

### 3.2 Dialogue Generation Protocol

A dialogue is generated by alternating between an **agent** (the assistant under evaluation), a **user simulator** (§3.3), and the **tool environment** provided by StableToolBench. For each problem, the environment exposes only the sampled set of endpoints. Each problem (§3.1) yields exactly one dialogue.

**Agent turn structure.** Within a single turn the agent runs a tool-using search loop. Two reserved control tools terminate the turn: `finish`, which the agent calls when it has a final answer for the user, and `message_user`, which the agent calls when it needs clarification or additional information before it can proceed. Between these, the agent may make an unbounded number of RapidAPI tool calls; tool responses come from StableToolBench's cached RapidAPI pool where available and from an LLM-simulated response otherwise, so the dialogue is deterministic with respect to cache content. The user simulator never sees tool calls or tool responses. Each dialogue has three requests, the initial query and two follow-ups; the message streams and the turn and search budgets are in Appendix C.

### 3.3 User Simulation

The user simulator answers the agent's clarification questions from the problem's withheld context and generates the follow-up requests that continue the dialogue. It is implemented as two separate LLM-prompted generators that share the same backbone model and the same view of the conversation; the active generator is chosen by whether the agent called `finish` or `message_user`.

**Follow-up query generator.** Active after the agent calls `finish`. The prompt instructs the simulator to extend (not rephrase) the conversation: ask a creative follow-up that builds on what the assistant already provided, explores a new facet of the problem, and is solvable by exactly one operation from `api_list`. A strict anti-hallucination rule forbids the simulator from attributing facts (IDs, names, numbers, dates) to the assistant or tools unless those facts appeared explicitly in the dialogue, while still permitting the user to introduce their own preferences and constraints.

**User-response generator.** Active after the agent calls `message_user`. The prompt is the same anti-hallucination contract as above, with one important addition: the generator receives the problem's withheld context (§3.1) and is instructed to weave relevant context keywords into the reply naturally when answering the assistant's clarification, rather than dumping all keywords up front. This is the mechanism by which the user simulator reveals the deliberately omitted details (location, dates, entity names, etc.) incrementally as the agent asks.

**Backbone model.** The simulator backbone is configurable to any OpenAI-compatible model; in our main experiments (§4) we use Qwen3-32B-AWQ served via vLLM, an open-weight model chosen for its balance of user-simulation quality and serving cost (§B.2). As a user simulator, GPT-4o-mini appears only as a closed-source comparison condition in the ablations of Appendix B.2 and Appendix B.4.

#### 3.3.1 Checker–Fixer Pipeline

The simulator generates each user turn (follow-up question or response to the assistant) with a single LLM call by default. When the optional checker–fixer pipeline is enabled, two further stages run before the turn is committed.

**Checker.** A separate LLM call evaluates the generated message against a fixed set of rules over the current dialogue history. The checker detects eleven violation types (Table 8). The checker returns a structured report with a boolean `has_issues` flag and a typed list of detected violations.

**Fixer.** If `has_issues` is true, a second LLM call revises the user message. The fixer receives the dialogue history, the original message, and the checker report, and is instructed to resolve all reported violations while preserving the communicative intent and without introducing new violations (e.g. no fabricating tool facts). The revised message replaces the original and is used as the next user turn; if no issues are reported the original message is used as-is. The same LLM is used for generation, checking, and fixing. Alongside the eleven LLM-judged violation types, a deterministic rule set flags near-duplicate sentence frames, verbatim repetition of prior turns, and empty or non-advancing messages; these rules require no model call and are reused as an independent measurement instrument in Appendix B.2.3. The pipeline's effect on turn quality is summarized in §4.2; the dialogue-level comparison and the paired turn-level analysis are in Appendix B.2.

### 3.4 Evaluation Design

We evaluate each generated dialogue along two axes: **dialogue quality**, measured by a five-aspect LLM-judged rubric, and **task success**, a per-subtask binary judgment of whether the user's informa-

tion need for that query was fully resolved. Both rely on LLM judges with structured function-calling outputs and a fixed prompt set across conditions.

#### 3.4.1 DIALOGUE QUALITY RUBRIC

The dialogue quality rubric scores each generated dialogue across five aspects, one LLM call per aspect: **consistency**, factual coherence between the assistant's claims and tool outputs; **relevance**, whether assistant turns address the user's request; **efficiency**, economy of tool calls and turns; **correctness**, whether the final answer is factually right with respect to tool evidence; and **adaptability**, handling of clarifications, corrections, and underspecified inputs. Each aspect is scored on a discrete $[0, 1]$ scale with five levels (0, 0.25, 0.5, 0.75, 1.0) and the judge additionally returns a brief justification. Aspect scores are combined into a single weighted aggregate (correctness 0.30; consistency, relevance, efficiency 0.20 each; adaptability 0.10). Because scores take only five discrete values, mean differences at the second decimal place correspond to a one-level shift on a small fraction of dialogues, so we do not read small second-decimal gaps as meaningful differences.

#### 3.4.2 TASK SUCCESS

Task success is determined by a **hybrid evaluator** that applies a deterministic hard-fail gate before invoking an LLM judge. It scores each **subtask** of the dialogue (the initial query and each follow-up query, §3.2) independently, rather than issuing a single judgment for the dialogue as a whole. The gate handles failure modes that can be identified cheaply and with certainty, so the LLM judge is reserved for cases where the outcome is uncertain.

**Subtask segmentation.** Each dialogue is split into its three main queries (one initial query and `num_follow_up_queries = 2` follow-ups), and the gate and LLM judge below are applied separately to each subtask's slice of the dialogue (its query, turns, tool calls, and tool responses).

**Deterministic gate.** A subtask fails without an LLM call when every substantive tool observation is StableToolBench documentation-placeholder prose, a description of the API rather than live data (for example, *"The API call to ... is designed to provide ..."*), and none of the observations yields data-shaped output such as structured fields, lists, or typed scalars beyond bookkeeping keys like `status`, `success`, or `code`. The gate is rarely decisive: it applies to only 20 of the 70,740 subtasks across the nine agents in §4.1 (0.03%; at most five per agent), placeholder prose appears at all in only 154 (0.2%), and no pooled subtask pass rate in Table 3 moves by more than 0.03 points with or without it, so cache coverage does not account for the failure rates we report.

**LLM judge.** If the gate passes, a single LLM call decides semantic task completion for that subtask via structured function-calling. The judge receives the subtask's query and its slice of the dialogue (turns, tool calls and their responses, and the assistant's answer) and must return a boolean `task_succeeded` with a brief justification. The system prompt instructs the judge to be strict on wrong or missing answers; a subtask passes only if the user's information need for that query was **fully** met by the final outcome.

**Dialogue-level score.** A dialogue's task-success score is the fraction of its subtasks that succeed, `success_count / total_subtasks` (e.g. 2/3 if the initial query and one of the two follow-ups succeed); a dialogue counts as fully successful only if every subtask succeeds. We report both quantities in §4.1: the pooled subtask pass rate as the headline figure, and the stricter all-subtasks-pass rate alongside it.

## 4 EXPERIMENTS

We report two experiments: a comparison of nine agent models on dialogue quality and task success (§4.1), and a validation of the simulated user's turn quality (§4.2).

| Metric | Proprietary | | | | Open-Weight | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 4o-mini | 4.1-mini | 5.4-mini | Opus 4.8 | Qwen3.5 397B | Qwen3.6 27B | Gemma 4 31B | GPT-OSS‡ 120B | GLM-5.2 744B |
| Overall (weighted) | 0.646 | 0.740 | **0.817** | 0.809 | 0.683 | 0.709 | 0.759 | 0.693 | 0.685 |
| Task success | 57.0% | 64.5% | 56.3% | **69.6%** | 60.9% | 61.0% | 67.5% | 55.2% | 67.9% |
| all three pass | 25.1% | 33.9% | 26.0% | **39.8%** | 29.3% | 29.0% | 36.4% | 24.9% | 37.7% |
| Consistency | 0.839 | 0.866 | **0.924** | 0.898 | 0.837 | 0.858 | 0.899 | 0.873 | 0.836 |
| Relevance | 0.581 | 0.713 | 0.771 | **0.788** | 0.646 | 0.717 | 0.696 | 0.695 | 0.662 |
| Efficiency | 0.433 | 0.573 | **0.696** | 0.604 | 0.444 | 0.520 | 0.529 | 0.494 | 0.439 |
| Correctness | 0.722 | 0.798 | 0.861 | **0.866** | 0.766 | 0.732 | 0.862 | 0.708 | 0.752 |
| Adaptability | 0.569 | 0.693 | 0.802 | **0.908** | 0.685 | 0.707 | 0.754 | 0.680 | 0.716 |

Table 3: Dialogue quality and task success per agent model, over a shared set of 2,620 problems. Bold denotes the best value per row. Open-weight models are labeled by family with total parameter count below; checkpoints and quantization are in Appendix E. The weighted aggregate, the pooled subtask pass rate, and the stricter all-three-pass rate are defined in § 3.4; dialogues and subtasks for which the judge returned an error are excluded. ‡GPT-OSS is also the judge model, so its scores are not judge-independent.

## 4.1 Agent Comparison

### 4.1.1 Experimental Setup

We evaluate how the choice of agent model affects dialogue quality and task success, holding every other component of the pipeline fixed so that observed variation is attributable to the agent. The user simulator is `Qwen/Qwen3-32B-AWQ` with the checker–fixer pipeline enabled (§ 3.3.1). This configuration is validated in § 4.2. The same model supplies API responses missing from the StableToolBench cache. All rubric and task-success judge calls use `gpt-oss` at temperature 0.2 with a fixed prompt set across all agents.

### 4.1.2 Agent Models Under Evaluation

We evaluate nine agent models drawn from two access tiers.

**Proprietary models.** We include four proprietary models: GPT-4o-mini, GPT-4.1-mini, GPT-5.4-mini, and Claude Opus 4.8. The three OpenAI models are a practical cost tier for running this evaluation at scale, and they offer strong instruction-following and function-calling performance relative to their cost. Opus 4.8 is a frontier model included as a reference for current proprietary capability.

**Open-Weight models.** We include five open-weight models: Qwen3.5-397B-A17B, Qwen3.6-27B, Gemma 4-31B-IT, GPT-OSS-120B, and GLM-5.2-744B (Appendix E). Together with the OpenAI tier this spans two orders of magnitude in parameter count (27B to 744B), both dense and MoE architectures, and four quantization schemes. All five have reasoning enabled by default, and we did not disable it, so these agents plan with an internal thinking pass before emitting tool calls.

### 4.1.3 Results

Table 3 reports mean scores across 2,620 dialogues per agent model, broken down by the five quality aspects, the weighted aggregate, and task success. Table 4 breaks task success down by the position of the main query within the dialogue.

**ToolTurns is hard, especially for completing all three task queries.** Pooled subtask pass rates fall in a 55–70% band and no model reaches 70%, so even for the best agent the judge finds the user's information need unmet on more than three subtasks in ten. The stricter criterion is far more discriminating: requiring all three subtasks of a dialogue to succeed drops every model below 40%, and the best model (Opus 4.8, 39.8%) completes a full three-query dialogue in fewer than two cases out of five. The gap between task success and passing all three highlights the difficulty of ToolTurns. If subtask outcomes within a dialogue were independent, a 69.6% subtask rate would imply a $0.696^3 \approx 34\%$ all-pass rate; the observed 39.8% is higher, so successes and failures are positively correlated within a dialogue. Either some problems are simply harder, or an agent that establishes the right tool and context early tends to keep succeeding. These two task-success rates

| Model | Initial | FU 1 | FU 2 | Δ |
|---|---|---|---|---|
| GPT-4o-mini | 65.2 | 55.7 | 50.3 | −14.9 |
| GPT-4.1-mini | 67.7 | 63.9 | 62.0 | −5.7 |
| GPT-5.4-mini | 57.6 | 55.9 | 55.5 | −2.1 |
| Opus 4.8 | 74.7 | 65.4 | 68.6 | −6.1 |
| Qwen3.5-397B | 67.6 | 57.8 | 57.5 | −10.1 |
| Qwen3.6-27B | 65.8 | 57.2 | 60.1 | −5.7 |
| Gemma 4-31B | 73.5 | 65.5 | 63.5 | −10.0 |
| GPT-OSS-120B | 65.2 | 51.5 | 49.0 | −16.2 |
| GLM-5.2-744B | 74.1 | 64.8 | 64.8 | −9.3 |

Table 4: Subtask success rate (%) by position of the main query within the dialogue: the initial query and the two follow-ups (FU 1, FU 2). Δ is the change from the initial query to the second follow-up. Every model degrades as the dialogue lengthens.

nonetheless rank models almost identically (Spearman $\rho = 0.97$ across the nine models, differing only in two swaps of adjacent pairs), so the choice between them affects the reported difficulty of the benchmark far more than the ordering of the models.

**Rubric quality and task success measure different things.** GPT-5.4-mini leads the weighted aggregate (0.817) and is best on consistency and efficiency, yet its task success (56.3%) is second-worst of the nine, below GPT-4o-mini's 57.0%. GLM-5.2 is the opposite: it is second only to Opus 4.8 on task success by both definitions (67.9% / 37.7%) while placing seventh on the weighted aggregate (0.685). The rubric and the task-success judge are therefore measuring different things. Rubric and task-success rankings correlate at only $\rho = 0.23$, while the two task-success definitions correlate at $\rho = 0.97$. The two can coincide, with Opus 4.8 best on task success and a close second on the aggregate (0.809), but the rubric does not reliably identify such models: it places GPT-5.4-mini ahead of Opus 4.8 despite a 13.3-point deficit in task success. Consistency and efficiency both score locally well-formed, economical tool use, which is not the same as resolving the request. An agent that issues few, coherent calls and writes a fluent answer scores well on both whether or not the answer is what the user asked for. GPT-5.4-mini makes 3.6 API calls per dialogue, the fewest of any agent, and 6.9% of its `finish` calls give up on the request; §F.2 shows a problem on which it scores 0.925 on the rubric and fails all three subtasks. The two also contribute very unequally to the aggregate. Consistency is uniformly high and spans only 0.09 (0.836–0.924), so once a tool result is in context, maintaining factual coherence with it barely separates these models. Efficiency spreads 0.26 (from 0.433 to 0.696) at a 0.20 weight, making it the single largest source of differences in the aggregate. It is also the aspect on which GPT-5.4-mini's lead is largest (0.092 over the runner-up, Opus 4.8). Adaptability spreads wider still (0.34, from 0.569 to 0.908) but carries the smallest weight (0.10).

**Open-weight models are competitive, and parameter count predicts little.** The two best open-weight models trail the frontier proprietary model by 1.7 and 2.1 points of task success (GLM-5.2 67.9%, Gemma 4 67.5%, against 69.6% for Opus 4.8) and beat every OpenAI model (the best, GPT-4.1-mini, reaches 64.5%); four of the five beat two of the three OpenAI models. Gemma 4 also achieves the second-best correctness score of any model (0.862, behind only Opus 4.8 at 0.866), despite being the second-smallest open-weight model and served at 4-bit. Across the 27B–744B range spanned by the open-weight models, size is a weak predictor of either metric. Gemma 4 at 31B is within 0.4 points of task success of GLM-5.2 at 744B, roughly twenty-four times its size, and ahead of the 397B Qwen3.5 by 6.6 points. The sharpest comparison is Qwen3.6-27B against Qwen3.5-397B: the newer, smaller, dense model matches the older sparse flagship on task success (61.0% vs. 60.9%) and is slightly ahead on the weighted aggregate (0.709 vs. 0.683), at a small fraction of the serving cost. What separates models appears to be the quality of their dialogue and tool-selection behavior rather than raw capacity, which is what we would want from a benchmark aimed at that skill. A locally served open-weight model is competitive on task success and on the dialogue-quality rubric.

**Success degrades with dialogue depth.** Because task success is scored per subtask, we can ask where in a dialogue agents fail. Table 4 shows that all nine models score highest on the initial query and lower on both follow-ups. The magnitude varies substantially: GPT-OSS and GPT-4o-mini lose 16.2 and 14.9 points from the initial query to the second follow-up, whereas GPT-5.4-mini loses

only 2.1. Opus 4.8 is the strongest model at both ends of the dialogue (74.7% initial, 68.6% on the second follow-up) and loses 6.1 points, less than GLM-5.2 or Gemma 4. For every model the larger share of the loss occurs at the first follow-up, and the second follow-up is then roughly flat relative to it (Qwen3.6-27B and Opus 4.8 even recover 2.9 and 3.2 points). That shape points to a one-time cost of switching from a fresh request to one that must build on prior dialogue, rather than a steady accumulation of context-length pressure. This is evidence that the multi-turn setting is what makes the benchmark difficult: the same agents on the same tools resolve the opening, context-free request 2–16 points more often than a request that depends on the dialogue so far. It also explains the GPT-5.4-mini result above: its weak overall task success comes from a consistently low level rather than a multi-turn weakness, since it is the flattest model here while being worst on the initial query (57.6%, against 73.5–74.7% for Gemma 4, GLM-5.2, and Opus 4.8). Part of GPT-OSS's decline reflects protocol non-compliance rather than dialogue ability: in 56.1% of its dialogues it returns an answer as plain text instead of calling `finish`, which leaves the subtask open (§F.4).

### 4.2 Validating Simulated User Turns

Because the simulated user shapes the evaluation trajectory, we validate its turn quality. On the turns it rewrites, the revision pipeline removes 41.2% of automatically detected defects, and blind human annotators prefer the revised turn 63.3% of the time. These measurements use the 1,272 user turns of a separate 322-dialogue single-call condition, not the 2,620-problem ToolTurns set: a checker fires on 42.7% of turns and a fixer rewrites 42.1%, the paired within-turn reduction is $-0.257$ defects per turn (95% bootstrap CI $[-0.308, -0.207]$, $n = 536$), and the human preference is 76 of 120 sampled pairs ($p = 4.5 \times 10^{-3}$). The gains concentrate on contextual unawareness, repetition, and failure to advance the task rather than on templated phrasing, which is the most frequent trigger but is indistinguishable from chance under human preference. Dialogue-level rubric scores are insensitive to a turn-local intervention. The scores, violation-type breakdown, comparison against human-written API-Bank dialogues, and detector calibration are reported together in Appendix B.

#### 4.2.1 Human Validation of the Agent Evaluators

We validate the LLM Judge used in Table 3 through human annotation. For binary task success, human and LLM judgments achieved substantial agreement (Cohen's $\kappa = 0.68$), validating the use of an LLM judge for evaluating agents for this task. Across the five dialogue-quality aspects, we see moderate agreement, with a quadratic weighted Cohen's $\kappa = 0.53$. While the agreement is lesser due to the challenges of evaluating each quality dimension, we find agreement is still significant.

## 5 Conclusion

We evaluate tool-using agents on ToolTurns: multi-turn dialogue over a broad API catalogue, with clarification and follow-up requests scored along each agent's own trajectory. The difficulty lies in the gap between resolving one request and completing a dialogue: no agent finishes even two dialogues in five, and every agent is weaker once a request depends on what came before. Because rubric scores rank models differently from completion ($\rho = 0.23$), progress in this setting has to be measured as completion rather than inferred from dialogue quality.

## 6 Limitations

ToolTurns scores agents on trajectories that a simulated user helps determine, so measured difficulty mixes the agent with that user's choices. Repairing individual user turns need not move dialogue-level scores; that comparison is in Appendix B. Tool observations come from StableToolBench's cache or from an LLM simulation on a cache miss, not from live provider calls, and task success is defined against that record. Initial queries and withheld context are generated rather than written by people, and every dialogue is scored on a fixed three-request horizon. The dialogue-quality rubric and the task-success judge are themselves LLM judges. The task-success judge checks answers against the tool observations of the current subtask only, so it can penalize an agent that corrects a wrong observation or reuses evidence from an earlier subtask (§F.3).

### REPRODUCIBILITY STATEMENT

Construction of ToolTurns is specified in §3: tool-set sampling and vague-query generation in §3.1, the dialogue protocol in §3.2, and both evaluators in §3.4. Appendix C gives the message streams, the turn and search budgets, and the simulator decoding temperatures; Appendix D lists the checker's violation types and Appendix E the open-weight checkpoints and quantization schemes. Open-weight agents and the user simulator are served with vLLM behind an OpenAI-compatible endpoint, so swapping an agent requires no change to the generation code; judge calls use a fixed prompt set at temperature 0.2 across all agents. Appendix F gives worked transcripts, with the tool calls and judge justifications behind our qualitative claims. The ToolTurns problems, the user simulator with its checker–fixer pipeline, the evaluators, and the scripts that produce the reported tables are provided as supplementary material.

### AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic data, to support qualitative analysis of simulated dialogues, and to interpret experimental results, and to improve the writing of the paper. GPT-4o-mini generates the initial queries and withheld context (§3.1). The user simulator, checker, and fixer are language models (§3.3). Dialogue quality and task success are scored by LLM judges (§3.4), and tool observations on a cache miss are simulated by StableToolBench (Guo et al., 2024). Formulating mathematical claims, writing proofs, developing theoretical models, and translation are not applicable to this work. We have reviewed all AI-assisted work: prompts and judge instructions were held fixed across the agents being compared, and the measurements reported in the paper were checked against those outputs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### ETHICS STATEMENT

The human study is a blind preference annotation of 120 pairs of model-generated user turns (§B.3). Each item showed the preceding dialogue and two unlabelled candidate messages, and the annotator chose the better turn under a rubric fixed in advance. Those turns were produced by the user simulator rather than collected from people, and the paper reports only aggregate preferences.

**Data terms.** Tool schemas are the RapidAPI collection released with ToolBench (Qin et al., 2024). ToolBench distributes that dataset for research and education under the Apache License 2.0. StableToolBench (Guo et al., 2024), which supplies cached responses and the fallback simulation, is released under the same license. Neither release transfers rights in the underlying RapidAPI listings or in content returned by third-party APIs. Live calls remain subject to RapidAPI's terms and to each provider's terms. Our experiments use the released schemas together with StableToolBench's cache or simulation, and do not make live provider calls or claim ownership of provider content.

## REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. $\tau^2$-Bench: Evaluating Conversational Agents in a Dual-Control Environment, 2025. URL https://arxiv.org/abs/2506.07982.

Benjamin Elder, Anupama Murthi, Jungkoo Kang, Ankita Rajaram Naik, Kiran Kate, Kinjal Basu, and Danish Contractor. Live API-Bench: 2500+ Live APIs for Testing Multi-Step Tool Calling, January 2026. URL http://arxiv.org/abs/2506.11266. arXiv:2506.11266 [cs.SE].

Nicholas Farn and Richard Shin. Tooltalk: Evaluating tool-usage in a conversational setting, 2023. URL https://arxiv.org/abs/2311.10775.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In Findings of the Association for Computational Linguistics: ACL 2024, pp. 11143–11156, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.664. URL https://aclanthology.org/2024.findings-acl.664/.

Wei He, Yueqing Sun, Hongyan Hao, Xueyuan Hao, Zhikang Xia, Qi Gu, Chengcheng Han, Dengchang Zhao, Hui Su, Kefeng Zhang, Man Gao, Xi Su, Xiaodong Cai, Xunliang Cai, Yu Yang, and Yunke Zhao. VitaBench: Benchmarking LLM agents with versatile interactive tasks in real-world applications. In The Fourteenth International Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2509.26490.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.emnlp-main.187/.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Findings of the Association for Computational Linguistics: NAACL 2025, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large Language Model Connected with Massive APIs, May 2023. URL http://arxiv.org/abs/2305.15334. arXiv:2305.15334 [cs].

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Jeonghoon Shim, Gyuhyeon Seo, Cheongsu Lim, and Yohan Jo. ToolDial: Multi-turn Dialogue Generation Method for Tool-Augmented Language Models. In The Thirteenth International Conference on Learning Representations, March 2025. URL https://openreview.net/forum?id=J1J5eGJsKZ. arXiv:2503.00564v1.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. ToolAlpaca: Generalized Tool Learning for Language Models with 3000 Simulated Cases, September 2023. URL http://arxiv.org/abs/2306.05301. arXiv:2306.05301 [cs.CL].

Hongru Wang, Wenyu Huang, Yufei Wang, Yuanhao Xi, Jianqiao Lu, Huan Zhang, Nan Hu, Zeming Liu, Jeff Z. Pan, and Kam-Fai Wong. Rethinking stateful tool use in multi-turn dialogues: Benchmarks and challenges. In Findings of the Association for Computational Linguistics: ACL 2025, pp. 5433–5453, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.284. URL https://aclanthology.org/2025.findings-acl.284/.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In The Twelfth International Conference on Learning Representations, 2024. URL https://arxiv.org/abs/2309.10691.

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. ToolPlanner: A tool augmented LLM for multi granularity instructions with path planning and feedback. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 18315–18339, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1018. URL https://aclanthology.org/2024.emnlp-main.1018/.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains, June 2024. URL http://arxiv.org/abs/2406.12045. arXiv:2406.12045 [cs.AI].

# A NOTES ON THE BENCHMARK COMPARISON

Sizes are corpus instances, annotated dialogues, or tasks, as labeled in Table 1; catalogue scale is not a difficulty ranking. ToolDial's 16 labels are dialogue action types, not callable tools, and its recorded-history test split has 1,166 dialogues. API-Bank's 1,888 training dialogues and 2,138 training APIs are excluded. $\tau$-bench and $\tau^2$-bench counts stay per domain; the generated $\tau^2$ telecom pool has 2,285 tasks, of which 114 are evaluated. DialogTool's training corpus (16,142 dialogues, 20 apps, 45 APIs) is separate from this evaluation row. The ToolBench repository samples 200 instructions from each of six subsets; those subsets are not summed, and that sampling note is in the repository rather than the paper body. ToolBench's 126,486 instruction–solution pairs are the count in the current paper and repository; an earlier preprint reports 12,657. DialogTool's row uses the 900-dialogue evaluation split in its Table 2. Its prose turn count disagrees with that table, so turn counts are omitted. ToolBench's Tool/API scope is that released catalogue: 3,451 tools, 16,464 APIs, and 49 categories. The ToolTurns row counts the distinct tools, endpoints, and categories that appear in the 2,620 problems: 1,827, 11,004, and 47 (Table 2), because seed sets with fewer than three endpoints are discarded and problems are sampled from G1–G3 (§ 3.1). ToolBench's evaluation split is likewise a subset of the same catalogue; that subset is not the figure in the table. Each of our problems exposes a subset of the catalogue (26.5 endpoints on average), and exposure does not mean every endpoint is used or required; tool responses are cached or simulated with StableToolBench (Guo et al., 2024).

# B USER-SIMULATOR EVALUATION

These analyses hold the agent and tool environment fixed and evaluate the simulated user: the dialogue-level judge, the checker–fixer comparisons, the blind human study of revised turns, and a comparison against human-written API-Bank dialogues.

## B.1 USER SIMULATOR EVALUATOR

The dialogue quality rubric and task-success evaluator both target the agent: they assume a reasonable user simulator and measure how well the assistant performs. To assess the simulator independently — for example, when comparing user-simulator configurations with the agent held fixed — we use a separate LLM judge that scores complete dialogues across seven dimensions on a continuous $[0, 1]$ scale. The judge receives the original query, the complete dialogue, and the final assistant answer; for each dimension it returns a score and a brief justification. The dimensions and their weights in the aggregate score are: **realism** (0.15) — whether the simulator's behaviour and language feel like a real user; **contextual awareness** (0.20) — whether messages are grounded in what the assistant actually said; **relevance** (0.15) — whether follow-ups are useful and on-topic; **natural language** (0.10) — whether phrasing is human-like and varied; **conversational flow** (0.15) — whether the dialogue progresses naturally; **clarity** (0.10) — whether queries are clear and interpretable; and **goal progression** (0.15) — whether user turns help resolve the original query. The overall score is the weighted sum of the seven dimension scores.

## B.2 IMPROVING USER SIMULATION WITH A CHECKER–FIXER PIPELINE

We evaluate the checker–fixer pipeline introduced in § 3.3.1 against single-call baselines, using the dedicated user-simulator evaluator from Appendix B.1. This isolates the effect of the revision pipeline (and, separately, the choice of simulator LLM) on user-simulator quality with the agent and tool environment held fixed.

### B.2.1 EXPERIMENTAL SETUP

We compare three user-simulator conditions on a shared set of 322 dialogues, holding the agent and tool environment fixed:

- **GPT-4o-mini (baseline).** The user simulator is GPT-4o-mini with a single call per turn; no checker or fixer. This serves as the closed-source baseline.

| Dimension | GPT-4o-mini | Qwen3-32B (single-call) | Qwen3-32B (checker–fixer) |
|---|---|---|---|
| Overall (weighted) | 0.858 | 0.862 | **0.871** |
| Realism | 0.807 | 0.805 | **0.812** |
| Contextual awareness | 0.863 | 0.869 | **0.880** |
| Relevance | 0.904 | 0.900 | **0.911** |
| Natural language | 0.874 | 0.877 | **0.880** |
| Conversational flow | 0.808 | 0.827 | **0.839** |
| Clarity | 0.905 | 0.902 | **0.907** |
| Goal progression | 0.864 | 0.867 | **0.877** |

Table 5: User-simulator quality scores under three conditions (322 dialogues each). Bold denotes the best value per row. Qwen3 refers to Qwen3-32B-AWQ; **checker–fixer** denotes the full checker–fixer pipeline (§3.3.1). Dimensions and weights follow Appendix B.1.

- **Qwen3-32B single-call.** The user simulator is Qwen3-32B-AWQ with one call per turn and no revision. This isolates the open-weight model's raw quality.
- **Qwen3-32B + checker–fixer.** The user simulator is Qwen3-32B-AWQ with the checker–fixer pipeline enabled.

The 322 initial queries were constructed by running the automatic vague-query procedure twice over ToolBench's three retrieval groups: 129 from G1, 129 from G2, and 64 from G3. As in §3.1, each query is paired with all endpoints of the tools in its sampled set—one tool for G1, and possibly several for G2 and G3— and GPT-4o-mini wrote the API-agnostic questions and their withheld context. The agent and tool environment are identical across conditions; only the user model and the optional revision step differ. All user-simulator evaluator calls use GPT-4o-mini as the judge with a fixed prompt set across conditions.

### B.2.2 Results

Table 5 reports mean scores across 322 dialogues for each condition.

All three conditions score in a narrow band (0.858–0.871 overall), and the differences between them are small relative to the resolution of the judge. The checker–fixer condition is nominally highest on all seven dimensions and on the weighted overall score, and the open-weight simulator with revision (0.871) is at least at parity with the GPT-4o-mini closed-source baseline (0.858). We deliberately do not read the $+0.009$ margin over the single-call open-weight baseline as evidence that the pipeline works: dialogue-level aggregates are structurally insensitive to this intervention, because the fixer rewrites only the minority of turns its checker flags and any effect is then averaged over the untouched turns of the dialogue. Appendix B.2.3 therefore measures the intervention where it actually operates.

**Aggregate parity conceals a difference in specificity.** The near-identical overall scores of GPT-4o-mini and the open-weight conditions do not imply equivalent behaviour. Counting **concrete references** per user turn — quoted names, identifiers, hashtags, CamelCase tokens, and proper-noun spans, extracted by a deterministic parser — GPT-4o-mini produces 0.98 per turn against 1.98 for Qwen3-32B, at essentially the same turn length (40.2 vs. 42.1 words) , and 43.1% of its turns contain no concrete reference at all against 19.3% for Qwen3-32B. The proprietary baseline attains comparable rubric scores in part by asking vaguer questions, which are harder for a rubric judge to penalise; a simulator that never commits to a specific entity cannot be marked down for misusing one. This is a limitation of aspect-style rubric judging that we return to in Appendix B.3.3.

### B.2.3 Turn-Level Intervention Analysis

The checker–fixer is a turn-local mechanism: it inspects one candidate user message and, if flagged, rewrites that message. Measuring it by comparing whole dialogues across two independently sampled conditions dilutes the effect twice over — across the untouched turns of each dialogue, and across the sampling variation between runs. We therefore evaluate the intervention directly, and pair at the level at which it acts.

**Protocol.** We take all 1,272 user turns produced by the Qwen3-32B single-call condition. These are unrevised drafts from the same backbone under the same prompts, so they are exactly the inputs the pipeline would have received. Holding the preceding dialogue history fixed, we run the production checker on each turn and, wherever it fires, the production fixer, using the same model and decoding settings as the deployed system. Only the single user message is rewritten; no dialogue is regenerated and no agent or tool call is re-executed, so the counterfactual differs from the observed draft in exactly one respect. We then score the original draft and the rewritten message against each other under identical history.

**Coverage.** The checker fires on 543 of 1,272 turns (42.7%), and the fixer alters the message on 536 of those (98.7% of firings; 42.1% of all turns). The most frequently reported violations in unrevised drafts are templating (75.9% of firings, 32.4% of all turns), lack of progress (33.7% of firings), and contextual unawareness (27.3%). Templating dominates, and 39.8% of firings report it as the sole violation, so the modal intervention is a rewording that preserves the request while breaking a reused sentence frame. Revised turns are correspondingly close to their drafts in content but not in form: mean word-level overlap between draft and revision is 0.37. We state this explicitly because it bounds what the pipeline should be credited with — it is predominantly a diversity and phrasing intervention, not a hallucination filter.

**Result.** On the 536 rewritten turns, we count violations using the deterministic rule set of § 3.3.1, which is independent of any LLM judge. Revision reduces the total from 335 to 197 detected defects, a 41.2% reduction. The paired within-turn change is $-0.257$ defects per turn (95% bootstrap CI $[-0.308, -0.207]$); 167 turns improve, 37 worsen, and 332 are unchanged (two-sided sign test, $p = 6.4 \times 10^{-21}$). Because the same detector is applied to the draft and its revision on identical history, this comparison is unaffected by the detector's absolute calibration: any constant bias cancels within the pair.

The result establishes that revision repairs the turns it targets; it does not establish a proportional improvement in end-to-end dialogue quality, which Table 5 shows to be small. An intervention that fires on 42% of turns and repairs a subset of defects in those turns should be expected to move a dialogue-level weighted average only slightly.

### B.3 User-Simulator Evaluation vs. Human Judgment

Appendix B.2.3 establishes the effect of the checker–fixer pipeline using a deterministic detector that shares its rule vocabulary with the checker itself. That circularity is worth taking seriously, so this section evaluates the same intervention against human judgment. Further analyses of the same labels are in Appendix B.3.3.

#### B.3.1 Experimental Setup

The checker–fixer acts on individual turns, so we annotate at that granularity. From the 536 turns revised in Appendix B.2.3 we sampled 120 uniformly at random. For each we presented the preceding dialogue history followed by the original draft and the revision as unlabelled candidates, in an order randomised per item. Annotation was blind: system identities were withheld, as was any indication of which candidate had been revised. The better user turn was selected under a rubric fixed in advance, applied in priority order: grounding in what the assistant actually provided, contextual awareness, task progression, avoidance of templated phrasing, and naturalness of user voice. Length was explicitly excluded as a criterion, as was surface politeness unless it accurately reflected what the assistant had said or done. No tie option was offered. These labels calibrate the automatic measure rather than serve as ground truth.

#### B.3.2 Results

**Human judgment confirms the intervention.** The revised turn was preferred in 76 of 120 cases (63.3%, 95% CI $[54.2, 71.7]$, two-sided sign test $p = 4.5 \times 10^{-3}$), with no position bias (side A chosen 45.0% of the time, and side A carried the revision in 48.3% of items). This is an independent confirmation of Appendix B.2.3: the two measurements share the sampled turns but nothing else, since the human rubric is applied to the turn as a whole rather than to the rule set the checker uses. The recurring failures identified in unrevised drafts were verbatim reuse of a prior turn's sentence

| Checker reported | Revision preferred | $p$ |
|---|---|---|
| Contextual unawareness | 26/34 (76.5%) | 0.003 |
| Repetition | 13/17 (76.5%) | 0.049 |
| Lack of progress | 34/45 (75.6%) | $8\times10^{-4}$ |
| Any non-templating | 49/73 (67.1%) | 0.005 |
| Templating alone | 27/47 (57.4%) | 0.38 |
| Verbosity | 5/19 (26.3%) | 0.064 |
| All revised turns | 76/120 (63.3%) | 0.005 |

Table 6: Blind human preference for the revised user turn over the original draft, by the violation type the checker reported. Categories overlap, since a turn may trigger several. Two-sided sign tests.

frame, requests for information the assistant had just declined or already supplied, and references to APIs or values appearing nowhere in the dialogue.

**The benefit is concentrated on substantive defects, not on the most frequent one.** Table 6 breaks the preference down by what the checker reported. Where the checker caught contextual unawareness, failure to advance the task, or repetition, the revision is preferred roughly three times in four. Where templated phrasing was the sole violation — 39.8% of all firings — the preference is indistinguishable from chance. Verbosity triggers are worse than neutral: the original draft was preferred in 14 of 19 such cases, suggesting the fixer's response to a verbosity flag is to compress away content that human judgment valued. Excluding verbosity triggers raises the overall preference to 70.3% ($n = 101$, $p = 5.5 \times 10^{-5}$). The pipeline's headline firing statistic and its human-visible benefit therefore come apart, and a checker that ignored surface phrasing would plausibly retain most of the gain at lower cost.

**Limitations.** The human annotation covers 120 turns drawn from a single intervention, and the rubric's templating criterion overlaps with a category the checker fires on, so annotation and detector are not wholly independent on that dimension. The turn-level design also cannot speak to whether repairing individual turns compounds into better dialogues; Table 5 suggests the aggregate effect is small. We did not evaluate a checker restricted to the violation types that human annotation validates, since doing so would require regenerating the full dialogue corpus.

### B.3.3 Further analyses of the human turn comparison

**A stylistic asymmetry that is a mechanism, not an artefact.** Revised turns open with an acknowledgment of the assistant's previous message far more often than drafts (43.3% vs. 20.0%), and the acknowledgment-bearing candidate was chosen in 41 of the 46 items where exactly one side had one. Read in isolation this looks like a learnable cue that could substitute for judgment. It is better explained as the repair itself. The 46 acknowledgment-asymmetric items are those where the checker flagged contextual unawareness (50.0% of them, vs. 14.9% of the remainder) or lack of progress (67.4% vs. 18.9%), whereas the 74 matched items are overwhelmingly templating-only (83.8% vs. 56.5%). Opening a turn by registering what the assistant just said is the natural way to repair a turn that ignored it, so conditioning the acknowledgment away removes the treatment rather than a confound, and the matched subset coincides almost exactly with the templating-only subset in which no effect exists to detect. We report the asymmetry for transparency but do not treat the matched-subset rate (56.8%, $p = 0.30$) as the corrected estimate.

**The deterministic detector is directionally sound but under-sensitive.** Human labels let us calibrate the instrument used in Appendix B.2.3. Across the 120 annotated pairs the deterministic count expresses a preference on only 47; on those it agrees with the human choice 68.1% of the time ($\kappa = 0.23$) and favours the revision 36 to 11, matching the human direction. On the remaining 73 it is silent, yet the revision is preferred on 45 of them (61.6%, $p = 0.06$). The detector thus misses roughly three fifths of the changes a human perceives while rarely pointing the wrong way.

**Implications for how these measures should be used.** Two conclusions follow, and together they justify the design of Appendix B.2.3. First, an absolute defect rate from this detector is not a trustworthy standalone quality measure, and we do not report one; it is silent on the majority of changes a human notices. Second, weak absolute calibration does not preclude paired use: applied

to a draft and its revision under identical history, a constant bias cancels within the pair, leaving the difference interpretable even though the levels are not. Since the detector is silent far more often than it errs, that difference understates rather than inflates the effect — which is why the automatic estimate ($41.2\%$ fewer defects) and the human estimate ($63.3\%$ preference) can both be reported without conflict. More broadly, the specificity gap in Appendix B.2 — GPT-4o-mini making half as many concrete references per turn while scoring comparably on the rubric — indicates that aspect-style judges reward safe, underspecified user behaviour, and should be complemented by measures sensitive to what a simulator actually commits to.

### B.4 Comparing the User Simulator to Human-Written Dialogues (API-Bank)

The experiments in Appendix B.2 compared user-simulator configurations against each other; the more fundamental question is whether the simulator behaves at all like a real user. We use API-Bank (Li et al., 2023) as a source of human-written, multi-turn user–assistant dialogues with explicit tool-use structure, and apply the same user-simulator evaluator (Appendix B.1) to both human-written and simulator-generated dialogues over a matched task distribution.

#### B.4.1 Experimental Setup

**Why API-Bank.** API-Bank's evaluation split provides annotated conversations that — unlike most ToolBench-style corpora, which centre on RapidAPI tool definitions and synthetic or retrieved queries — are closer to what we want for studying tool-using dialogue under a fixed task distribution. It is not, however, RapidAPI-isomorphic: API-Bank names capabilities with short symbolic API names (e.g. calendar or email operations), so it cannot be run directly against our StableToolBench environment without a bridging step to real RapidAPI tool schemas.

**Data preparation.** We normalize API-Bank test data into a single repository layout: 622 JSON files (one per dialogue) plus a global `initial_queries.json` index (622 rows; 522 multi-turn level-1/2 scenes, 50 level-3 holistic scenes, 50 level-3 batch scenes). Each dialogue record carries `dialogue_messages` (user / assistant roles), an `initial_query`, `metadata.tools` (distinct tool names in first-seen order), an optional `api_request_trace`, and gold fields where the source provides them (e.g. next API string vs. next utterance, depending on level). This format is stable for downstream scripting and reusable for the LLM-judge comparison below.

**Mapping API-Bank tasks to RapidAPI tools.** Because our agent environment executes RapidAPI-style tools indexed from a local ToolBench tree (category / tool / API JSON), we map each API-Bank dialogue's tool hints into a small set of concrete RapidAPI operations. We first collect the distinct API-Bank tool names appearing in `metadata.tools` across the slice we use (68 unique names in the full 622-dialogue set). For each name, we retrieve candidates from the local catalog via embedding similarity over serialized tool+API descriptions, then optionally apply a single lightweight LLM rerank that chooses a fixed number of candidates by index, so the model cannot invent tool names. For each dialogue, we merge per-name lists in hint order, deduplicate by `(category, tool, API)`, and cap the number of exposed APIs (top 15), producing a merged query file aligned with our existing dialogue runs (`original_query`, `query_str`, `context`, `api_list`).

**Simulated dialogues.** For each of the 622 API-Bank initial user goals we run our generative dialogue protocol: a tool-using agent and a user simulator alternate until dialogue limits or termination criteria are met. The agent may call RapidAPI tools through StableToolBench, restricted to the mapped `api_list` for that instance. We generate two simulator conditions — GPT-4o-mini and Qwen3-32B-AWQ with the checker–fixer pipeline enabled — holding the agent and tool environment fixed.

**Evaluation procedure.** We score the user side of each dialogue with the user-simulator evaluator from Appendix B.1: same judge model (GPT-4o-mini), same prompt, same seven dimensions, applied uniformly to all four dialogue sets. The two human reference slices are **Human (all 622)** and **Human (excl. 54)**, where the latter removes 54 purely one-line items that do not exercise multi-turn behaviour. The aim is to determine whether, under this single rubric, the simulator is not much worse than real users — justifying its use for larger-scale experiments — not to claim the judge is ground truth or that the two sets differ only in who wrote the user turns. In particular, the human references are tied to API-Bank's assistant and environment, while our simulated runs use our agent and the mapped

| Dimension | GPT-4o-mini user | Qwen3-32B user | Human (all 622) | Human (excl. 54) |
|---|---|---|---|---|
| Overall (weighted) | 0.845 | **0.851** | 0.737 | 0.772 |
| Realism | 0.762 | **0.766** | 0.576 | 0.610 |
| Contextual awareness | 0.900 | 0.888 | **0.915** | 0.909 |
| Relevance | 0.864 | **0.885** | 0.618 | 0.677 |
| Natural language | 0.861 | **0.867** | 0.776 | 0.823 |
| Conversational flow | 0.783 | **0.807** | 0.710 | 0.772 |
| Clarity | 0.915 | 0.903 | 0.914 | **0.916** |
| Goal progression | 0.841 | **0.850** | 0.663 | 0.715 |

Table 7: User-simulator quality scores for our simulator vs. human-written API-Bank dialogues (622 tasks per condition), all scored by the same LLM judge with the same prompt and dimensions (Appendix B.1). **Qwen3-32B user** uses the checker–fixer pipeline. **Human (excl. 54)** removes 54 purely one-line items from the reference set. Bold denotes the best value per row. Simulator and human dialogues differ in the assistant and tool environment as well as in the user, so observed gaps reflect protocol differences and not user behaviour alone.

RapidAPI tools, so part of any gap may reflect the assistant and environment rather than the user model alone.

### B.4.2 Results

Table 7 reports user-simulator quality scores for the four dialogue sets, each judged by the same LLM under identical instructions.

Simulated users score higher on average than the filtered reference set (0.851 vs. 0.772 overall). Part of this gap is depth: our generated dialogues typically include many user turns and follow-ups, whereas the reference bundle is mixed — shorter threads and single-turn-style items remain common even after the 54 purely one-line cases are removed. The rubric rewards coherent follow-up behaviour and sustained progress, so longer, follow-up-heavy runs are favoured even when both sides are judged by the same template. The remaining contrast reflects style and packaging: reference text often reads like benchmark snippets (economical wording, occasionally unnatural disclosure or formatting), while simulated users tend to be more cooperative and verbose in ways the automated judge reads as fluent and on-task. The gap should therefore be read as distributional and protocol mismatch — especially turn count and elaboration — not as proof that synthetic users are uniformly "more human" under a single gold standard. Notably, on the two dimensions least sensitive to dialogue length — contextual awareness and clarity — humans and the simulators score within 0.01–0.03 of each other across all four sets, suggesting parity on surface-level grounding once length effects are factored out.

## C Dialogue generation details

**Dual message streams.** The runtime maintains two parallel message lists: **`search_messages`** is the agent's full view, including system prompt, tool calls, tool responses, and intermediate reasoning, and is carried across turns so the agent has full context; **`dialogue_messages`** is the user-visible conversation, containing only the initial query, user-simulator turns, and the agent's terminating `finish`/`message_user` content. The user simulator sees only `dialogue_messages` — never tool calls or tool responses — which is what enforces realism: the simulator behaves like a human user who does not observe the assistant's internal tool operations.

**Termination and message budgets.** Each dialogue consists of exactly one initial query followed by a fixed `num_follow_up_queries` = 2 follow-up queries, giving three main queries (**subtasks**) per dialogue. Each main query's sub-dialogue is capped at `max_turns_per_main_query` = 10 user-visible messages; if the cap is reached before the agent calls `finish` or `message_user`, the agent is forced to conclude that subtask. The response generator may still emit the sentinel `<STOP>`, treated as the user abandoning the current subtask early. A separate budget, `max_search_messages` = 15, caps the agent's within-turn search depth; if exceeded, the agent is given one synthesis call that prompts it to call `finish` based on the tool responses collected so far, rather than aborting the turn outright.

**Follow-up generator output.** The follow-up generator returns a JSON object with the new `instruction` text plus a candidate `api_calls` hint (used only for inspection; the agent re-plans freely).

**Decoding temperatures.** The simulator generates each turn at temperature 0.7 for follow-up queries and 0.8 for clarification responses (slightly higher for variety).

## D CHECKER VIOLATION TYPES

| Type | What the checker flags |
|---|---|
| Repetition | Repeating prior requests or information already given |
| Hallucination | Fabricating facts attributed to the assistant or tools |
| Contextual unawareness | Ignoring what the assistant actually said or asking for already-provided information |
| Verbosity | Unnecessary or overly long text |
| Conversational flow | Unnatural or disconnected phrasing |
| False agreement | Claiming to have received information that was not provided |
| Excessive restatement | Echoing the assistant's reply instead of a brief acknowledgment |
| Lack of progress | Not answering the assistant's question or not advancing the task |
| Vagueness | Queries too generic to be actionable |
| Ignoring assistant questions | Not addressing a direct question |
| Templating | Reusing a near-verbatim sentence frame from an earlier user turn with only the key term swapped |

Table 8: The eleven violation types detected by the user-turn checker (§3.3.1).

## E OPEN-WEIGHT AGENT CHECKPOINTS

The five open-weight agents in § 4.1 are Qwen3.5-397B-A17B (sparse MoE, 397B total / 17B active, official FP8), Qwen3.6-27B (dense, bf16), Gemma 4-31B-IT (dense, 4-bit QAT `w4a16`), GPT-OSS-120B (sparse MoE, native MXFP4), and GLM-5.2-744B (NVFP4). Routing all models through the same OpenAI-compatible API means no changes to the dialogue generation code are required when swapping agents.

## F WORKED EXAMPLES

The transcripts below are taken from the nine-agent runs of §4.1; each is identified by its problem index in ToolTurns. Tool outputs are abridged, long agent messages are cut where marked [. . . ], and tool names are shortened by dropping the `_for_<tool>` suffix. User turns, agent messages, and judge justifications are otherwise verbatim. All tool observations come from StableToolBench's cache or its LLM simulator (§3.2), not from live APIs, and several of the examples turn on that fact. JUDGE lines give the task-success verdict for the subtask (§3.4.2).

### F.1 ONE PROBLEM, SEVERAL TRAJECTORIES

Problem 2610 is a real instance of the sketch in § 3.1. The initial query is *"I heard there's a big game coming up soon. Can you tell me when it is?"*, the withheld context is `NBA`, `Los Angeles`, `Lakers`, `October`, and the tool set exposes 48 endpoints from three tools (Free NBA, Web Search, GeoDB Cities). Table 9 gives every agent's outcome. The NBA tool is unreliable in this environment: depending on the call it returns an error, games from past seasons, or another team's games under the Lakers' ID. Agents therefore have to clarify the request and then decide what to do with bad tool data, and they differ on both.

**Gemma 4: clarifies, then answers only from tool evidence (3/3).**

USER "I heard there's a big game coming up soon. Can you tell me when it is?"
CALL `web_search(q="big game coming up soon sports")`

| | 4o-mini | 4.1-mini | 5.4-mini | Opus 4.8 | Qwen3.5 | Qwen3.6 | Gemma 4 | GPT-OSS | GLM-5.2 |
|---|---|---|---|---|---|---|---|---|---|
| Initial | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| FU 1 | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ |
| FU 2 | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✗ | ✓ |
| Rubric overall | 0.55 | 0.75 | 0.68 | 0.80 | 0.53 | 0.85 | 0.80 | 0.43 | 0.70 |

Table 9: Subtask outcomes and weighted rubric score for problem 2610. Follow-up queries differ across agents because each is generated from that agent's own dialogue.

AGENT Could you please let me know which sport or specific game you're referring to? There are many big games coming up across different leagues like the NBA, NFL, NHL, or college sports, and I'd love to give you the exact details!

USER Oh, sorry! I was thinking about the NBA season since I'm in Los Angeles and a huge Lakers fan. I was wondering if there's a big game coming up for them in October?

CALL `get_all_teams(page="1")`

CALL `get_all_games(seasons="2024", team_ids="14")`

TOOL Games for the Milwaukee Bucks. [. . . ]

CALL `web_search(q="Los Angeles Lakers schedule October 2024 big games")`

TOOL "The Los Angeles Lakers will open the 2024-25 season with a high-stakes road game against the Golden State Warriors on October 25, 2024. They will also host the Phoenix Suns on October 29, 2024 [. . . ] Their most anticipated game is against the Boston Celtics on October 31, 2024, at Crypto.com Arena."

AGENT The most anticipated matchup is on **October 31, 2024**, when the Lakers will host the **Boston Celtics** at Crypto.com Arena. Other notable games in October include: October 25, 2024: A high-stakes road game against the Golden State Warriors to open the season. October 29, 2024: A key home game against the Phoenix Suns. [. . . ]

JUDGE ✓ *The assistant provided specific dates for the Lakers' upcoming big games in October 2024, matching the information from websearch results and fully answering the clarified user request.*

Gemma disregards the schedule endpoint's wrong-team output, and every date in its answer appears in a tool observation. The judge checks consistency with those observations, not with the real 2024–25 schedule.

**GPT-4o-mini: skips clarification and answers a different question (1/3).**

USER "I heard there's a big game coming up soon. Can you tell me when it is?"

CALL `get_all_games(seasons=["2023"], per_page="5", page="1")`

TOOL `{"error": "Failed to generate fake response"}` (twice)

CALL `news_search(q="upcoming NBA games")`

TOOL "The Los Angeles Lakers and Golden State Warriors are set to face off in the final round of the NBA playoffs, with the first game scheduled for July 10, 2026."

AGENT The first game in the NBA playoffs between the Los Angeles Lakers and the Golden State Warriors is scheduled for July 10, 2026. You can find more details in the news articles regarding upcoming NBA games.

USER Actually, that's not quite right - the NBA regular season starts in October, not the playoffs. The Lakers have a few key games coming up in October to kick off the season. Do you know when their first home game will be in Los Angeles this year?

*[eleven further NBA-API calls: the team list, then ten requests for 2023-season games, each of which returns an error or another team's games]*

AGENT I currently cannot retrieve the specific date for the Los Angeles Lakers' first home game in October. You might want to check the official NBA website or sports news sites for the latest updates on their schedule.

USER Oh, right! I forgot you couldn't access the schedule directly. I actually just checked the Lakers' website and their first home game in Los Angeles this season is on October 28th against the Sacramento Kings. [. . . ]

*[the agent returns search-result links through* `message_user` *until the subtask reaches the 10-message cap]*

JUDGE ✗ *The assistant initially gave an incorrect date and never provided the correct date of the upcoming Lakers game; the final answer only offered streaming and ticket links without confirming the game's date, so the user's original request for when the big game is was not satisfied.*

The agent never asks which game the user means; it guesses a league, finds an unrelated event in a news snippet, and presents it as the answer. The user's correction supplies the withheld context, but the agent gets no further with it. The transcript also shows a simulator failure: once the agent gives up, the simulated user supplies a date and opponent that appear nowhere in the dialogue. The anti-hallucination rule in § 3.3 forbids attributing facts to the assistant, but it does not stop the user from asserting facts of its own, and the checker did not flag this turn. The judge still fails the subtask, since the agent itself never answered, but the remaining subtasks build on the user's invented game.

**GPT-OSS: clarifies correctly, then fabricates the schedule (0/3).**

USER "I heard there's a big game coming up soon. Can you tell me when it is?"

AGENT Sure! Could you let me know which sport or specific game you're interested in (e.g., NBA, football, a particular team, etc.)?

USER Oh, I'm talking about the NBA! Specifically, the Los Angeles Lakers. I know they have some home games in October, but I'm not exactly sure which dates. Do you know when their next game is this month?

*[six NBA-API calls, none of which returns a 2026 schedule (they return errors, the team list, or games from the 2018 season), and eight web searches whose snippets say the Lakers play the Warriors, Suns, and Jazz in October but give no dates]*

AGENT According to the official 2026-27 Los Angeles Lakers schedule posted on NBA.com, the Lakers' first home game in October 2026 is on **Saturday, October 25, 2026** when they host the Golden State Warriors at Crypto.com Arena. The next two home games that month are: **Monday, October 27, 2026** vs. the Phoenix Suns; **Thursday, October 30, 2026** vs. the Utah Jazz. These three games are the Lakers' only home appearances in October 2026. [. . . ]

JUDGE ✗ *The assistant fabricated specific October 2026 dates for Lakers home games without any supporting data from the tools; the available tool results did not contain the schedule, so the answer is unreliable and does not correctly answer the user's request for the next game.*

USER *(follow-up 1)* Can you provide more details about the October 25th game against the Golden State Warriors, such as the start time or any special events associated with it?

*[searches return three different start times; the agent reports one of them, but as plain text rather than through* `finish`*, so the subtask stays open for three more user turns about Fan Fest perks, merchandise, and food]*

USER *(follow-up 2)* Can you check if there are any vegetarian or gluten-free options available at the Fan Fest food court, since I have dietary restrictions and want to make sure I can enjoy the snacks without issues?

None of the three dates appears in any tool observation; the opponents do, and the attribution to NBA.com is invented. Because the simulator sees only the user-visible conversation, it treats the fabricated schedule as fact, and both follow-ups are about a game whose date the agent made up. The first follow-up's verdict is also affected by a protocol failure specific to this agent. The runtime treats a turn that ends without `finish` as a message to the user, so the simulator keeps replying, and the judge scores the last answer in the slice, which is about food, against a question about tip-off time. Table 11 shows how often this happens.

**Opus 4.8: clarifies, answers the date, cannot confirm the time (1/3).** Opus asks for the sport, team, and date range, finds a season opener against the Warriors on 5 October 2026 in a news snippet, and says that it could not confirm the tip-off time. The judge fails the initial subtask because the user's clarification asked for "the exact date and time." Under the strict "fully met" criterion, an honest partial answer receives the same score as a wrong one.

### F.2 High rubric score, no task success

Problem 140 isolates the disagreement between the dialogue-quality rubric and task success reported in § 4.1. The query is *"I'm looking for a new smartphone but I have a specific brand in mind."* with withheld context `Samsung`, `Galaxy S23`, `features`, and three endpoints (`GetBrands`, `GetPhonesByBrandId`, `GetPhoneDetails`). The brand list the environment returns is truncated alphabetically at Lenovo, so Samsung's identifier never appears in any observation and no agent can

complete the intended tool path. GPT-5.4-mini receives the highest rubric score of the nine agents on this problem (0.925) and fails every subtask; Opus 4.8 scores 0.60 on the rubric and passes every subtask (Table 10).

| | Consistency | Relevance | Efficiency | Correctness | Adaptability | Overall | Task success |
|---|---|---|---|---|---|---|---|
| GPT-5.4-mini | 1.00 | 1.00 | 0.75 | 1.00 | 0.75 | 0.925 | 0/3 |
| Opus 4.8 | 1.00 | 0.25 | 0.50 | 0.50 | 1.00 | 0.600 | 3/3 |

Table 10: Rubric aspects and task success for two agents on problem 140.

**GPT-5.4-mini.**

AGENT Which brand are you interested in?
USER I'm looking at Samsung phones, specifically the Galaxy S23. I heard it has some great features, but I'd like to know more about it. [...]
CALL `getbrands()`
TOOL `[{"id":237,"name":"Acer". . . },. . . ,{"id":288,"name":"Lenovo",. . .` (truncated)
AGENT I can look up the Galaxy S23 details, but I need the Samsung brand entry first. Would you like me to continue by finding Samsung in the brand list and then pulling the S23 specs?
USER Yes, please go ahead and find Samsung in the brand list. [...]
CALL `getbrands()`
TOOL (the same truncated list)
CALL `finish(return_type="give_up_and_restart", final_answer="I can look up Samsung and the Galaxy S23, but I only received a partial brand list response and couldn't reliably confirm Samsung's entry or retrieve the S23 details from it. Please try again, and I'll continue from the brand list to find Samsung and then the Galaxy S23 specs.")`
JUDGE ✗ *The assistant failed to provide any details about the Samsung Galaxy S23 and instead gave up, indicating it could not retrieve the necessary information.*
*[both follow-ups repeat the pattern: one more* `getbrands` *call and another* `give_up_and_restart`*]*

**Opus 4.8.**

*[asks for the brand; the user names Samsung and the Galaxy S23;* `getbrands` *returns the same truncated list, and the agent tries brand IDs that do not appear in it]*
CALL `getphonesbybrandid(brandid=323)`
TOOL `{"error": "Failed to generate fake response"}`
CALL `getphonesbybrandid(brandid=322)`
TOOL `[12345, 12346, . . . , 12400]`
CALL `getphonedetails(phoneid=12345)` `. . . (12346)` `. . . (12347)`
TOOL "Phone ID 12345 corresponds to a hypothetical device. [...]"; "Phone ID 12346 corresponds to the Apple iPhone 14 Pro. [...]"; the same for 12347
AGENT [...] I want to be upfront: our phone database wasn't returning reliable Samsung records just now (the entries came back either incomplete or mislabeled), so rather than give you inaccurate specs, here's a trustworthy rundown of the Galaxy S23 family: **Galaxy S23 (base model)** – Released Feb 2023; 6.1" Dynamic AMOLED 2X display, 120Hz [...]
JUDGE ✓ *The assistant provided a comprehensive overview of Samsung Galaxy S23 models, covering specifications, variants, and guidance, directly satisfying the user's request for feature details despite missing database records.*

The rubric credits GPT-5.4-mini for never saying anything the tools contradict, and it marks Opus down on relevance and correctness for answering beyond them. The task-success judge asks only whether the user's need was met, and it accepts an answer drawn from the model's own knowledge when the agent reports that the tool is broken. We report this as a property of the two metrics, not a verdict on which agent behaved correctly. The pattern is not specific to this problem. In 63 dialogues GPT-5.4-mini scores at least 0.9 on the rubric and fails all three subtasks, and 6.9% of its `finish` calls are `give_up_and_restart`, against 0.1% for Opus 4.8 (Table 11).

### F.3 When the evaluator is wrong

Task success is judged against the tool observations in each subtask's slice of the dialogue (§3.4.2). The two cases below show how that design produces false failures.

**A wrong tool observation (problem 589, Opus 4.8).** The user has chosen a meeting slot on Wednesday, 30 September 2026, 11:00–12:00 Central Time, and asks how it looks in Eastern and Pacific time.

CALL `time_zone_converter(datetime="2026-09-30T11:00:00", from_tzname="US/Central", to_tzname="US/Pacific")`

TOOL `"from": {"datetime": "2026-09-30T11:00:00-05:00"}, "to": {"datetime": "2026-09-30T08:00:00-08:00"}`

AGENT [...] (all on daylight saving time that date) [...] Pacific (PT / PDT): 9:00 AM – 10:00 AM [...]

JUDGE ✗ *The assistant gave the correct Eastern Time conversion but incorrectly listed Pacific Time as 9:00-10:00 AM instead of the correct 8:00-9:00 AM per the tool results, so the user's request was not fully met.*

The tool applies the Pacific standard-time offset in late September, when daylight saving time is in effect. The agent overrides it with the correct answer, and the judge, which is instructed to check the answer against tool results, fails the subtask. The rubric scores this dialogue 0.95.

**Evidence outside the judged slice (problem 1515, Gemma 4).** In the initial subtask, a balance endpoint reports that the user's wallet holds "2.50 ETH (≈ $5,000.00 USD)" and "3,000 USDC (≈ $3,000.00 USD)", and the agent reports this correctly (✓).

USER *(follow-up 1)* That's helpful! Could you also show me a breakdown of the specific assets in my Ethereum wallet?

CALL `wallets_assets_by_chain(address="0xabc123def456", chain="ethereum")`

TOOL `{"data": [], "projects": [], "total": 0}`

AGENT Based on your wallet (0xabc123def456) on the Ethereum blockchain, here is the breakdown of your assets: 2.50 ETH (approximately $5,000.00 USD); 3,000 USDC (approximately $3,000.00 USD)

JUDGE ✗ *The assistant fabricated asset balances (2.50 ETH, 3,000 USDC) even though the tool returned an empty asset list, resulting in a critical hallucination and failing to provide the correct breakdown.*

The figures come from a tool observation one subtask earlier, which the judge never sees because it receives only the current subtask's slice. One can argue that the agent should have pointed out the contradiction between the two endpoints, but "fabricated" is wrong. To estimate how often this happens, we took the 3,124 failed follow-ups whose justification cites fabricated or unsupported content and extracted the numbers in each final answer that do not appear in that subtask's tool observations. In 297 cases (9.5%), most of those numbers do appear in an earlier subtask's tool observations. This amounts to 0.2–1.4 points of follow-up success per agent, well below every agent's follow-up decline in Table 4.

### F.4 Behavioural statistics

Table 11 counts, over all 2,620 dialogues per agent, the behaviours that recur in the examples above.

| | 4o-mini | 4.1-mini | 5.4-mini | Opus 4.8 | Qwen3.5 | Qwen3.6 | Gemma 4 | GPT-OSS | GLM-5.2 |
|---|---|---|---|---|---|---|---|---|---|
| Task success (%) | 57.0 | 64.5 | 56.3 | 69.6 | 60.9 | 61.0 | 67.5 | 55.2 | 67.9 |
| `finish` calls that give up (%) | 7.1 | 1.6 | 6.9 | 0.1 | 7.9 | 6.1 | 1.1 | 12.4 | 0.6 |
| Initial queries with no API call (%) | 1.7 | 4.0 | 28.3 | 3.3 | 2.3 | 2.2 | 4.1 | 9.2 | 1.1 |
| API calls per dialogue | 15.6 | 8.3 | 3.6 | 7.2 | 14.0 | 10.3 | 10.6 | 10.7 | 17.4 |
| `message_user` calls per dialogue | 3.05 | 1.99 | 0.73 | 0.86 | 1.38 | 1.78 | 0.92 | 1.36 | 1.29 |
| Dialogues with an answer outside `finish` (%) | 0.3 | 0.0 | 0.5 | <0.1 | <0.1 | 0.3 | <0.1 | 56.1 | <0.1 |

Table 11: Agent behaviour over the 2,620 ToolTurns dialogues. **Give up**: `finish` called with `return_type` `give_up_and_restart`. **No API call**: the initial subtask, including any clarification turns, contains no call other than `finish` or `message_user`. **Answer outside `finish`**: at least one assistant turn that contains a `final_answer` payload as plain text instead of a `finish` call; the runtime treats such a turn as a message to the user.

Across the nine agents, the give-up rate ranks them almost exactly in reverse order of task success (Spearman $\rho = -0.93$). The three agents with the highest task success give up in 1.1% of `finish` calls or fewer, and every other agent gives up in at least 1.6%. GPT-5.4-mini makes the fewest API calls (3.6 per dialogue) and handles 28.3% of initial queries without any API call, compared with 1.1–9.2% for the other agents; it succeeds on only 44.3% of those. This combination explains how it leads on efficiency while trailing on task success (Table 3). GPT-OSS ends a turn with a plain-text answer instead of `finish` in 56.1% of its dialogues, so its subtasks often stay open and are scored on whatever answer ends the slice. Part of GPT-OSS's low task success, and of its 16.2-point follow-up decline, therefore reflects non-compliance with the tool-calling protocol rather than failures of reasoning or tool use. GPT-4o-mini calls `message_user` most often (3.05 per dialogue); as in § F.1, some of these calls carry answers rather than questions, which also keeps a subtask open. Finally, keyword matching on the judge's justifications (e.g. "fabricated", "hallucination", "not supported by") flags unsupported content in 7.3% of failed initial subtasks and 16.5% of failed follow-ups. The count is heuristic, but it agrees with the GPT-OSS example above: failed follow-ups more often contain claims that go beyond the available tool evidence.