



MM-ToolSandBox: A Unified Framework for Evaluating Visual Tool-Calling Agents

Kaixin Ma*, Di Feng*, Alexander Metz, Jiarui Lu, Eshan Verma, Afshin Dehghan

Apple

*Equal Contribution

We introduce **MM-ToolSandBox**, a benchmark and evaluation framework for visually grounded tool-calling agents. The framework provides a stateful execution environment spanning 500+ tools across 16 application domains, supporting multi-image, multi-turn tasks where agents must ground progressively arriving visual inputs into executable tool calls while handling realistic conversational phenomena (goal revisions, error corrections, state mutations). An automated scenario generation pipeline produces diverse, visually grounded scenarios through information-flow-guided planning and multi-stage quality filtering, yielding 258 human-verified nominal scenarios and 50 variants targeting interactive UI applications. Evaluating 12 state-of-the-art models, from 4B open-weight to frontier proprietary systems, shows that current models still lack robust visual tool-calling capability: even the best model achieves below 50% success rate. Our failure analysis further reveals that visual precision, not only planning, is a primary bottleneck for capable models: 53% of failures stem from incorrect information extraction from images despite otherwise correct task workflows. A planning-to-precision crossover emerges with scale: smaller models fail at deciding *what to do*, while larger models fail at perceiving *what they see*, suggesting fundamentally different research directions for improving models at different capability levels. The framework and the benchmark are publicly available at <https://github.com/apple/ml-mmtoolsandbox>.

Date: July 14, 2026

1 Introduction

Evaluating LLM-based agents on tool-augmented tasks has become an active area of research, with benchmarks targeting function calling (Li et al., 2023; Patil et al., 2025), stateful environment interaction (Lu et al., 2025; Trivedi et al., 2024), and multi-turn conversational dynamics (Froger et al., 2026; Barres et al., 2025; Xiu et al., 2026). However, most of these frameworks remain text-centric, and cannot evaluate many realistic assistant tasks with visual inputs, for example, a user may share a screenshot to diagnose an issue or provide a photo of an event poster to schedule an event.

Solving visual tool-calling tasks requires more than visual recognition. The agent must extract task-relevant information from images, map visual evidence to the correct tool or code action, execute that action against a stateful environment, and continue the interaction when the user’s intent evolves. This makes visual tool calling a distinct agentic capability: the model must not only understand what is shown, but also decide how to act on it through tools.

Among the few benchmarks that incorporate visual inputs, images are typically provided as a static prefix in the initial query (Wang et al., 2024a), or the tasks are limited to single-round user-agent interactions (Xie et al., 2024; Rawles et al., 2025). Moreover, many existing benchmarks operate with relatively small tool spaces, often containing only 15–100 tools (Froger et al., 2026; Wang et al., 2024a; Kong et al., 2025; Lu et al., 2025). As a result, there is no unified framework and benchmark for systematically studying the visual tool-calling capability of multimodal agents under dynamic, multi-turn, and large-tool-space settings.

To close this gap, we introduce **MM-ToolSandBox**, a diverse simulation framework for evaluating visual tool-calling agents. MM-ToolSandBox extends the stateful tool-use environments from ToolSandBox (Lu et al., 2025), with explicit image handling, visual-specific tools, and support for both structured tool-use

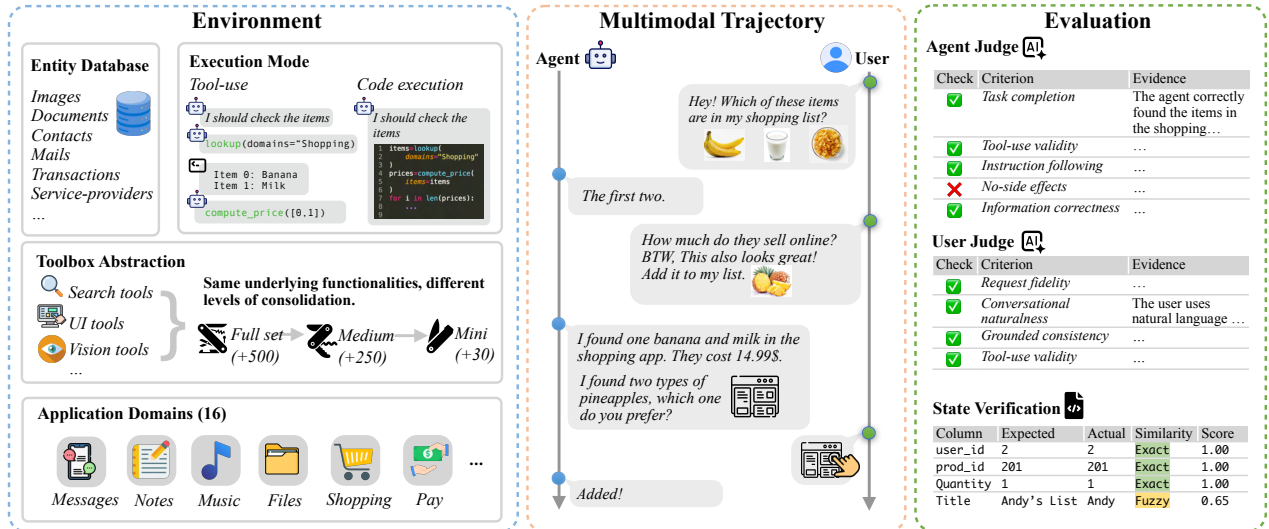


Figure 1 Overview of the MM-ToolSandBox framework for evaluating visual tool-calling agents. The framework supports multi-turn, multi-image interaction between the user and agent, and provides both structured tool-use and code-execution interfaces. The framework includes over 500 native tools across 16 application domains, covering search, UI, vision, and system-level utilities. Evaluation combines rubric-based LLM judges for both agent and user roles with static state-verification checks.

and code-execution interfaces, as summarized in Figure 1. Built on top of this framework, we construct a visual tool-calling benchmark designed to capture the key challenges missing from prior work. The benchmark spans 511 tools across 16 simulated application domains and is organized along three orthogonal axes: seven information-flow types, four challenge types that model realistic conversational deviations, and four image-arrival patterns that control when visual inputs enter the interaction. The resulting benchmark contains 258 nominal scenarios with 1,284 unique images, requiring agents to perform multi-image, multi-turn tool use over a large and diverse tool space. All scenarios are further corrected and verified by human experts to ensure reliable visual grounding, executable task specifications, and well-defined completion criteria. We also curate 50 scenarios to explore interactive UI applications. To make benchmark construction scalable while maintaining high quality, we develop an automated scenario generation pipeline. The pipeline produces visually grounded scenarios through information-flow-guided planning, environment-state instantiation, and multi-stage quality filtering, reducing the annotation burden from full manual construction to targeted human review. Finally, we conduct a systematic evaluation of state-of-the-art proprietary and open-weight models on our benchmark. Our results show that even strong frontier models struggle substantially with visual tool calling, revealing persistent gaps in image-grounded reasoning, multi-step interaction, and stateful tool use. In-depth analysis further shows that multi-image working memory is a key bottleneck, and that models of different scales exhibit distinct failure patterns.

To summarize, our contributions are: 1. A unified framework and benchmark for evaluating visual tool-calling agents. 2. A scalable scenario generation pipeline. 3. A systematic evaluation of diverse models.

We release the benchmark and the framework to the public at <https://github.com/apple/ml-mmtoolsandbox>.

2 Related Work

Tool-use agent benchmarks. A growing body of benchmarks evaluate LLM agents on tool-augmented tasks. API-Bank (Li et al., 2023) and BFCL (Patil et al., 2025) target single- and multi-turn function calling with large API collections. ToolSandBox (Lu et al., 2025) introduces stateful environments with online evaluation and conversational simulation. AppWorld (Trivedi et al., 2024) scales to 457 APIs across 9 applications with rich relational data, though exclusively in a code-execution paradigm. More recent efforts push toward dynamic environments: Gaia2 (Froger et al., 2026) introduces asynchronous, event-driven

Table 1 Comparison of representative tool-use agent benchmarks and environments. **Visual** indicates whether image or UI observations are part of the input. **Scenario** distinguishes single-turn tasks, where the user provides a complete goal upfront and the agent executes autonomously, from multi-turn tasks, where the agent must interact with the user to resolve ambiguity, obtain missing information, or handle user updates. **Env.** describes the execution substrate, while **Exec.** specifies the agent interface: *Tool* denotes structured function/tool calling, *Code* denotes executable code generation in a runtime such as a Python environment, and *UI+Tool* denotes hybrid interaction through both graphical interfaces and external tools.

Benchmark	Visual	Scenario	Environment	Execution	Apps/Domains	Tools
AppWorld (Trivedi et al., 2024)	✗	single-turn	API sim.	Code	mobile apps (9)	457
Toolathlon (Li et al., 2025a)	✗	single-turn	MCP/server	Tool	software apps (32)	604
MCP-Atlas (Bandi et al., 2026)	✗	single-turn	MCP/server	Tool	mixed domains (36)	220
ToolSandbox (Lu et al., 2025)	✗	multi-turn	API sim.	Tool	mobile apps (11)	34
Gaia2 (Froger et al., 2026)	✗	multi-turn	API sim.	Tool	service apps	101
ASTRA-bench (Xiu et al., 2026)	✗	multi-turn	API sim.	Tool	mobile apps	51
τ^2 -Bench (Barres et al., 2025)	✗	multi-turn	API sim.	Tool	telecom support	30
GTA (Wang et al., 2024a)	✓	single-turn	tool suite	Tool	vision/web tasks	14
OSWorld-MCP (Jia et al., 2025)	✓	single-turn	OS/UI	UI+Tool	desktop apps (7)	158
MobileWorld (Kong et al., 2025)	✓	multi-turn	OS/UI	UI+Tool	mobile apps	64
MM-ToolSandBox	✓	multi-turn	API sim.	Tool+Code	mobile apps (16)	511

evaluation testing temporal awareness and agent-to-agent collaboration; τ^2 -Bench (Barres et al., 2025) models agent-supervisor interactions revealing communication as a critical bottleneck; and ASTRA-bench (Xiu et al., 2026) grounds evaluation in longitudinal personal context. GTA (Wang et al., 2024a) is among the first to incorporate multimodal contexts, but covers only 14 tools in a stateless setting. SO-Bench (Feng et al., 2026) evaluates the visual structural output capability of tool-use agents, but only focuses on single-turn interactions. Toolathlon (Li et al., 2025a) further scales tool-use evaluation to long-horizon real-software workflows, spanning 32 software applications and 604 tools with realistic initial states and execution-based task verification. MCP-Atlas (Bandi et al., 2026) evaluates agents on 1,000 multi-step tasks over 36 real MCP servers and 220 tools, requiring agents to discover and orchestrate 3–6 tool calls across servers from natural-language instructions. Despite this progress, these benchmarks operate predominantly in text-only regimes, evaluate under a single execution mode, and rely heavily on manual scenario construction.

Visual agent benchmarks. A parallel line evaluates agents that perceive and act on GUI inputs, known as GUI agents (Wang et al., 2025; Li et al., 2025d; Yang et al., 2025). VisualWebArena (Koh et al., 2024) and WebVoyager (He et al., 2024) evaluate web navigation grounded in visual page understanding. At the OS level, OSWorld (Xie et al., 2024), AndroidWorld (Rawles et al., 2025) and UINavBench (Agrawal et al., 2025) benchmark desktop and mobile tasks, respectively, while MobileWorld (Kong et al., 2025) extends to long-horizon cross-application workflows. In multimodal search, MMSearch (Jiang et al., 2024) and MM-BrowseComp (Li et al., 2025c) evaluate deep retrieval requiring visual-language reasoning. These benchmarks share grounding in visual perception but differ in image types (UI screenshots vs. natural images), action spaces (GUI operations vs. API calls), and evaluation (navigation success vs. state verification). Our work occupies a complementary position: agents perceive *heterogeneous visual inputs (ranging from natural photographs to documents, charts, and UI screenshots)* shared during conversation and must translate them into *high-level API tool calls* within a stateful, multi-domain environment, testing the perception-to-tool-invocation pipeline rather than pixel-level UI manipulation.

A feature comparison with prior benchmarks is in Table 1.

3 MM-ToolSandBox

MM-ToolSandBox aims to measure visual tool-calling capability: whether an agent can understand visual context, decide which tools (or code) are needed, use visual artifacts correctly across turns, and complete tasks through grounded, multi-step interaction. To this end, MM-ToolSandBox inherits the lightweight, interactive, and robust design philosophy of ToolSandbox (Lu et al., 2025), while extending text-based tool-use

environments to visual settings through explicit visual artifact handling. To support richer and more dynamic assistant scenarios, we further incorporate domains and tools from AppWorld (Trivedi et al., 2024), resulting in 16 simulated application domains and 511 native Python tools. Together, these design choices make MM-ToolSandBox a generic runtime for prototyping diverse multimodal tool-calling scenarios. Figure 1 provides an overview of our system design.

3.1 Architecture

Each episode is parameterized by a **scenario**, which specifies the initial and expected world states, available tools, user instructions, associated visual assets, and completion criteria. The interaction then unfolds among an **agent**, a **user**, and an **execution environment**. An LLM plays the simulated user, the agent acts as the task solver (under evaluation), and the environment maintains the shared world state while executing tool calls. The interactions among agent, user and environment operate over two channels. The front-end interaction between the agent and user is a natural user-assistant dialogue, while the backend interaction with the environment is mediated through typed tool calls.

Tool Management. Tools are implemented as native Python functions with typed signatures and metadata such as role visibility and state dependencies (Lu et al., 2025). Because we provide over 500 tools in MM-ToolSandBox, exposing the full registry in a single prompt is impractical. We therefore provide a tool-discovery meta-tool, **search_tool**, which retrieves relevant tools on demand from a natural-language query over tool documentation. Retrieved tools are added to the agent’s active set, while a least-recently-used (LRU) eviction policy keeps the working set bounded. To further test tool-use robustness under varying registry sizes, we curate three additional variants from the full native toolset: a medium set with 276 tools, a compact set with 165 tools, and a mini set with 30 tools.

3.2 Multimodal Interaction

Multimodal interaction is the main focus of this work. Building on the message-passing and world-state design of ToolSandbox (Lu et al., 2025), we treat images as first-class visual artifacts rather than static prompt attachments. Each image is stored in a shared image database, assigned a stable identifier, and referenced by tools and messages in the same way that other environment entities are referenced. This design makes visual artifacts persistent, reusable, and traceable across turns. Image exchange is mediated through tool calls. For example, the user can provide images to the agent during the conversation via **send_message_with_image**, and the agent can similarly return visual artifacts to the user when needed. We further provide visual-specific tools and other utilities for resizing, annotating, plotting, transforming images, as well as interactive UI generation. As a result, images become part of the structured interaction trajectory rather than untracked context-window inputs. An example is shown in Figure 1.

3.3 Execution Modes

MM-ToolSandBox implements two execution modes for invoking tools: free-form **Code-Execution** and structured **Tool-Use**. In **Code-Execution** mode, the model operates through standard chat completion and writes Python code in markdown-fenced blocks, following the code-as-action paradigm (Trivedi et al., 2024; Wang et al., 2024b). The framework parses and executes the code in a persistent sandboxed interpreter, where registered tools are available as callable Python functions. This interface naturally supports complex workflows involving loops, conditionals, intermediate variables, and flexible tool composition, with the drawback of reduced structural constraints. In **Tool-Use** mode, the model operates through the provider’s structured tool-calling API, as commonly supported by OpenAI, Anthropic, and Gemini (OpenAI, 2025; Anthropic, 2026; Gemini, 2025). At each step, the model either responds to the user or emits a schema-constrained tool call that is executed by the environment. This interface is easy to parse, auditable, and robust to malformed outputs, but can be cumbersome for complex control flow or multi-tool composition. To support programmatic orchestration in this mode, we provide an **execute_code** tool that invokes the same sandboxed interpreter, and code execution remains mediated by the structured tool-calling interface.

Safety Guardrails. Executing agent-generated Python code requires careful protection regardless of how the code is invoked. Following the spirit of AppWorld’s sandboxed execution design (Trivedi et al., 2024),

MM-ToolSandBox protects the interpreter with three layers of guardrails: (1) static checks that reject disallowed imports and unsafe primitives before execution; (2) runtime restrictions that limit file access, intercept dangerous system calls, and cap memory usage; and (3) execution limits on wall-clock time and output size.

3.4 Evaluation

MM-ToolSandBox evaluates scenarios through two complementary mechanisms (illustrated on the right of Figure 1). **State-based verification** compares initial and final world-state snapshots against a scenario’s expected entity changes, providing an objective, reproducible outcome signal. **Rubric-based LLM judges** assess trajectory quality for both roles: an Agent Judge evaluates whether the agent completed the task correctly, while a User Judge checks whether the simulated user behaved coherently. Together, these mechanisms cover both the outcome and the process of each scenario. Concrete metrics are defined in Section 4.3.

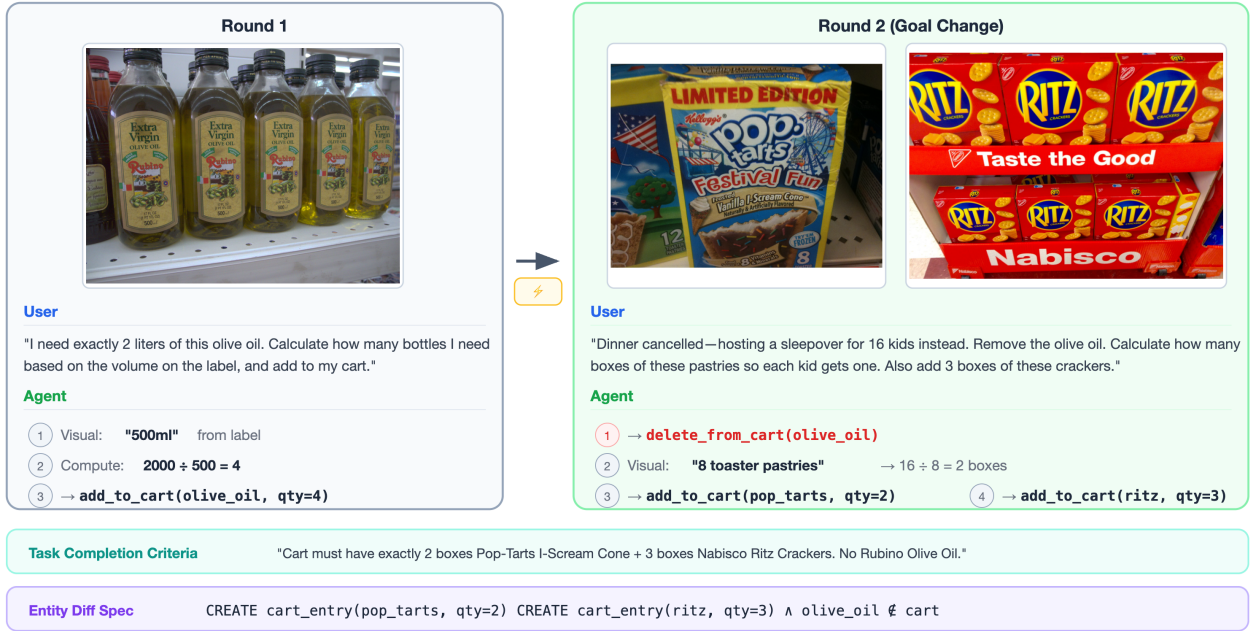


Figure 2 Example scenario illustrating the interplay of benchmark design dimensions. The agent must (1) extract numerical information from product images (visual grounding), (2) perform arithmetic to determine quantities (compute flow type), (3) adapt when the user changes plans mid-conversation (goal change challenge), and (4) handle images arriving across turns (progressive arrival). Evaluation verifies the final cart state against expected entity diffs.

4 Benchmark

The MM-ToolSandBox benchmark comprises 258 human-verified scenarios spanning diverse visual reasoning patterns, conversational challenges, and application domains. Each scenario specifies a multi-turn, multi-image task that requires the agent to extract information from visual inputs, navigate a large tool space, and produce correct state changes in a simulated device environment. We describe the design dimensions that structure the benchmark, followed by its composition and statistics. In addition, we curate 50 variants for interactive UI applications. A scenario example is shown in Figure 2. More examples can be found in Appendix I.

4.1 Design Dimensions

We structure the benchmark along three orthogonal axes, namely, *information flow type*, *challenge type*, and *image arrival pattern*, that jointly control the reasoning pattern, conversational dynamics, and visual

complexity of each scenario.

Information flow types. The flow type prescribes the reasoning pattern connecting visual inputs to tool actions. We derive seven flow types by systematically decomposing the space of operations that relate multiple information sources to action outputs, grounding our taxonomy in data processing primitives: *aggregate* (cf. UNION): combine information from multiple sources into a single action; *compare* (cf. ARGMAX): evaluate alternatives against criteria to select the best; *filter* (cf. WHERE): select a subset of items matching a condition; *compute* (cf. derived column): derive a new value from inputs through calculation; *validate* (cf. CHECK): verify information against constraints before acting; *lookup chain* (cf. correlated subquery): resolve references sequentially, each feeding the next; and *cross-reference* (cf. JOIN): match information across sources and act on the correspondence. These flow types impose genuinely different reasoning demands, e.g. lookup chains require sequential retrieval from individual images, while filter and aggregate tasks require synthesizing information across multiple images simultaneously.

Challenge types. We control for two dimensions of task difficulty beyond the cooperative baseline. First, two conversational challenges test agent robustness to imperfect user behavior: *goal change*: the user pivots mid-conversation, requiring the agent to abandon prior progress and adapt to revised objectives; and *error correction*: the user provides incorrect details (e.g., misidentifying content in an image), testing whether the agent detects discrepancies between visual evidence and user instructions rather than executing blindly. Second, state mutation controls operational complexity by requiring the agent to update or delete existing entities rather than only creating new ones, demanding entity retrieval and grounded modification instead of unconstrained creation. Scenarios without any of these properties form the cooperative baseline with create-only operations.

Image arrival patterns. The arrival pattern determines when images enter the conversation: *upfront*: all images in the first message, *progressive*: images distributed across turns, *late*: images arriving toward the end, or *mixed*: a combination of the three. Progressive delivery tests whether agents can integrate new visual context as the interaction evolves, while upfront delivery tests simultaneous multi-image working memory, which is a fundamentally different cognitive demand.

4.2 Benchmark Composition

Image sources. We sourced images from the *test sets* of eight established vision datasets spanning complementary visual domains: documents (DocVQA (Mathew et al., 2020), OmniDocBench (Ouyang et al., 2025)), natural scene text (HierText (Long et al., 2023)), real-world scenes (WorldVQA (Zhou et al., 2026)), charts and plots (ChartQA-Pro (Masry et al., 2025), ChartMuseum (Tang et al., 2025)), infographics (InfographicVQA (Mathew et al., 2021)), and software UI screenshots (ScreenSpotPro (Li et al., 2025b)). This diversity ensures that benchmark performance reflects general visual perception rather than proficiency on a single image type. Given the image set, we leverage our automatic scenario generation pipeline to produce a diverse and challenging scenario set (detailed in Section 5). Each scenario requires 3–6 images and 2–5 user interaction rounds. In total we cover 1,284 unique images over 258 scenarios, and the distribution of design dimensions is shown in Figure 4.

UI mode subset. We additionally convert a 50-scenario subset to *UI mode*, where the same underlying tasks must be completed through visual interface interaction rather than text. In this mode, the agent acts as a dynamic UI designer: it discovers backend tools, gathers information, and renders interactive screens (product cards, selection lists, confirmation dialogs) through which the user makes decisions via button clicks and selections. The key design principle is that the user’s decision criterion is *private*, so the agent cannot complete the task without presenting options and receiving the user’s choice through the UI. Entity diff specifications remain identical, ensuring that the same programmatic evaluation applies regardless of interaction modality. A third judge (*UI judge*) additionally evaluates whether the agent rendered appropriate interactive interfaces with correct affordances. A converted UI scenario is shown in Figure 9.

To support interactive UI scenarios, we implement a set of dedicated UI tools that mediate structured visual interactions between the agent, the user, and the environment acting as a UI state server (Figure 3). On the frontend, the agent and the user continue to communicate in natural language as in any other scenario. On the backend, the agent constructs a declarative UI description and calls a rendering tool; the environment validates,

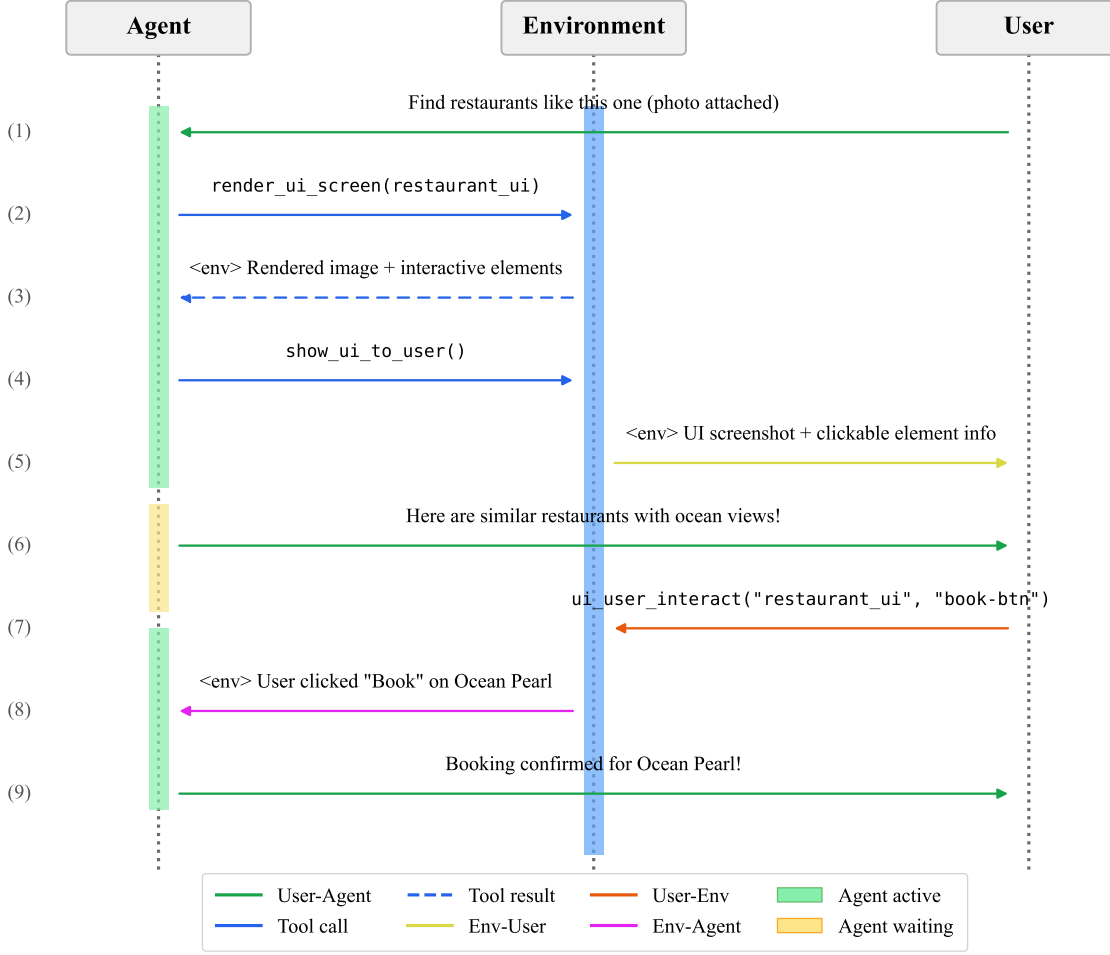


Figure 3 An example information flow for an interactive UI scenario. The user simulator and agent interact through natural language on the front end, while both interact with the UI server environment through dedicated UI tools on the back end. The agent generates a new UI screen by writing Python code following the A2UI protocol, which is executed through `render_ui_screen`. The environment renders the resulting UI screen and returns it to the user role as an environment message. The user simulator then observes the screen and interacts with it through `ui_user_interact`, emulating common UI actions such as clicking and typing.

renders, and stores the resulting screen, then delivers it to the user together with metadata describing the available interactive elements. The user can then perform actions such as clicking a button or filling a form field through a dedicated interaction tool, and the environment resolves the action into structured data that is forwarded back to the agent as a notification. This design keeps the conversational surface natural and readable while making every UI state change auditable through typed tool calls. UI tools are built based on the recently published A2UI protocol (Google, 2025a). Figure 3 demonstrates the information flow for UI interaction.

4.3 Evaluation Protocol

By default, we use GPT-5.4 (OpenAI, 2026) to build the user simulator, as we find that this model achieves the highest user fidelity and natural conversation compared to other frontier models. The prompts for both user and agent roles are provided in Appendix E and Appendix F.

Agent implementation. To isolate model capability from scaffolding, all models share a single fixed harness (Yao et al., 2023; Wang et al., 2024b). Unless otherwise noted, the agent runs in the pure code-execution

interface (Section 3.3): it operates a persistent sandboxed Python interpreter, discovers tools on demand from the full 511-tool registry via a discovery tool under an LRU working set, and is capped at 100 interaction steps (across agent and user roles) with a per-scenario wall-clock time limit. Native thinking is enabled for models that support it. Every model receives the same agent-role system prompt (Appendix E) and interacts with the same GPT-5.4 (OpenAI, 2026) user simulator, and all trajectories are scored by a Claude 4.5 Sonnet (Anthropic, 2025a) rubric judge. We hold this harness fixed on purpose: as agent scaffolding increasingly drives end-task performance (Anthropic, 2025; OpenAI, 2026), comparing models under a common, well-specified harness is what makes performance differences attributable to the model rather than to the surrounding system. We separately study how the interface itself affects performance in Section 6.3.

Evaluator. Each scenario is scored using the mechanisms described in Section 3.4. **Agent Success Rate (Agent SR)** is the primary metric: the agent judge receives the full conversation trajectory and tool calls with results and then evaluates five independent criteria: task completion, instruction following, tool use validity, no side effects, and information accuracy, and a scenario passes only if *all five* pass. The complete agent-judge rubric prompt is provided in Appendix G. **Entity F1** provides graded *partial credit* by programmatically comparing actual entity state changes against expected specifications via Hungarian matching with type-aware column similarity (exact match for identifiers, ROUGE-L for text, datetime verification for timestamps; full details in Appendix A). **User SR** assesses whether the simulated user behaved coherently, serving as a sanity check that user quality does not confound agent evaluation (user-judge rubric in Appendix H). We show in Section B that User SR correlates strongly with Agent SR and that excluding user-failing scenarios does not alter model rankings, confirming that reported performance differences reflect genuine agent capability. We validate judging fidelity against human annotation: the agent judge achieves 88% agreement (Cohen’s $\kappa = 0.748$, substantial) with a conservative bias that under- rather than over-estimates Agent SR, while the user judge attains high raw agreement (77%) though a lower κ driven by the rarity of user-simulation failures. Together with the strong Agent-SR/User-SR correlation ($r = 0.81$, Figure 6a), this indicates that both the user simulator and the judges are reliable (Appendix C).

5 Scenario Generation Pipeline

We design an automated pipeline that generates multi-image, multi-turn scenarios at scale while satisfying the design dimensions defined in Section 4.1. The pipeline operates in six stages (Figure 4), producing the benchmark described in Section 4. All LLM-based stages use Gemini-3.1-Pro.

Stage 1: Image–tool association. Not all images naturally motivate tool use. Given an image sourced from the image corpus, an LLM identifies whether it contains genuinely actionable content and maps it to relevant application domains and specific tool functions. Strict filtering ensures only images requiring both visual extraction and tool invocation, where the task cannot be accomplished without the image, pass to downstream stages.

Stage 2: Image clustering and flow type selection. Randomly grouping images produces forced, artificial scenarios. We instead first group images by associated domains and then cluster them into groups of 3-6 using CLIP embedding similarity, ensuring high relevance within each group. For each cluster, we assess the feasibility of each information flow type and image arrival pattern (defined in Section 4.1). A lightweight viability assessment scores each flow type and each arrival pattern for the cluster (0-3: impossible to ideal); only natural fits (score ≥ 2) are retained. Without this pre-selection, we observe that the LLM defaults to the simplest viable pattern, collapsing reasoning diversity.

Stage 3: Outline generation. Before generating the full scenario, the pipeline produces a structured outline by selecting from the acceptable flow types and image arrival patterns identified in Stage 2, with weighted sampling that steers toward a balanced distribution across the benchmark. The outline specifies the number of user rounds (2–5), the selected flow type and arrival pattern, required tools, and a challenge type (Section 4.1).

Stage 4: Scenario generation. The outline is expanded into a full scenario using flow-type-specific prompt templates that encode the assigned information flow structure, ensuring genuine multi-step reasoning rather than independent parallel actions. The output includes per-turn user queries with deictic references to visual content, transition conditions governing when new images or corrections are introduced, a goal with all

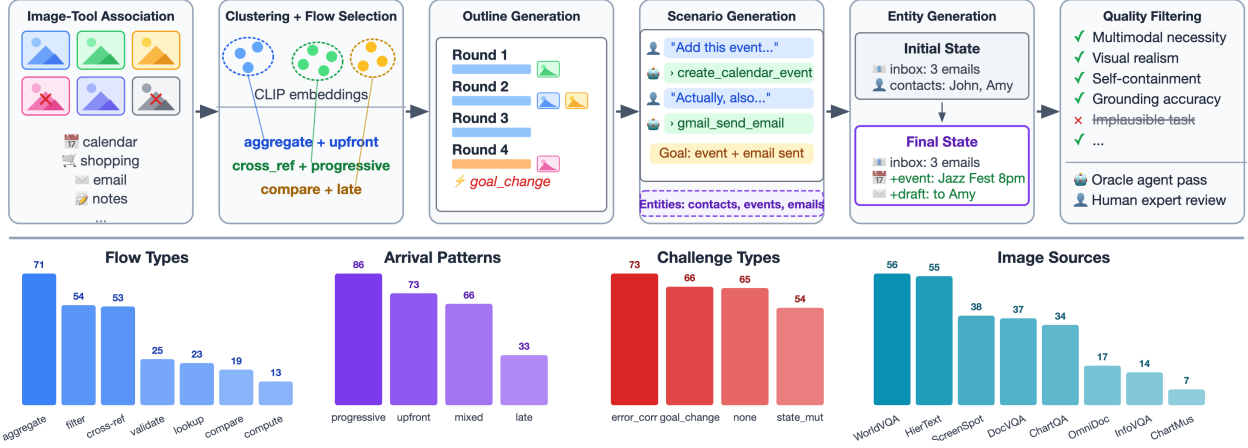


Figure 4 Overview of the scenario generation pipeline (top) and the resulting benchmark distribution (bottom). **Top:** Each candidate image is first associated with relevant application domains and tools (Stage 1), then grouped via CLIP-based clustering and screened for viable information-flow types and image-arrival patterns (Stage 2). A structured multi-round outline is sampled from the viable patterns and assigned a challenge type (Stage 3), then expanded into a full multi-turn dialogue with deictic visual references and tool calls (Stage 4). Stage 5 instantiates the initial environment state together with the expected final state for deterministic state-based evaluation. Stage 6 applies LLM-based quality criteria, followed by an oracle agent solvability check and human expert review. **Bottom:** Distribution of the 258 final scenarios across the three design dimensions of Section 4.1, flow types, image-arrival patterns, and challenge types, together with the originating image-source datasets.

references resolved to concrete values for automated evaluation, and *preexisting entity requirements*, the data that must exist in the environment for the scenario to be valid (e.g., existing calendar events, contacts, or email threads). A user profile sampled from a pool of synthetic personas provides realistic social context for scenarios involving personalized information.

Stage 5: Entity generation. Stateful evaluation requires fully specified environment state. Using the preexisting entity requirements from Stage 4, an LLM instantiates *initial entities* (concrete data populating the env so the scenario is valid) and *final entities* (the expected env state after successful completion, including created, modified, and deleted entities), enabling deterministic evaluation by diffing actual post-execution state against the expected outcome.

Stage 6: Automated quality filtering. Each scenario is evaluated by an LLM critic against seven quality criteria: multimodal necessity, visual realism, self-containment, conversation step integrity, goal flexibility, visual grounding accuracy, and scenario plausibility. Scenarios passing all criteria then undergo balanced downsampling to maintain diversity across design dimensions of Section 4.1. The downsampled set is then validated for solvability and entity correctness through the verification protocol described in Section 5.1.

5.1 Quality Verification

To verify that every released scenario is executable end-to-end, not merely well-formed, we add a final layer on top of Stage 6 filtering, combining an oracle rollout check and human expert review.

Oracle solvability check. For every scenario, we run an oracle agent whose system prompt is augmented with ground-truth information: the expected entity state changes, the task completion criteria, and the list of expected tools. We default to Claude 4.5 Opus as the oracle backbone; using a strong model with full ground-truth access establishes an upper bound on scenario solvability that is unconfounded by model capability. A scenario passes only when the oracle achieves *both* a near-perfect entity-diff match (Entity F1 ≥ 0.9) and a passing Agent Judge verdict; failures are routed for correction or removal.

Human expert review. Because the oracle check cannot detect scenarios that are technically solvable yet semantically misaligned with the user’s intent, every scenario passing the oracle is further inspected by human expert reviewers. The reviewers verify that the dialogue, expected tools, entity specifications, and

Table 2 Main results. Unless otherwise specified, all models are evaluated with thinking mode enabled using the Code-Execution interface.

Model	Agent Success Rate $\uparrow$	Entity F1 $\uparrow$	Avg. Steps $\downarrow$
Qwen 3.5-4B (Qwen, 2026)	0.116	0.641	54.5
Qwen 3.5-9B (Qwen, 2026)	0.190	0.693	53.8
Qwen 3.5-27B (Qwen, 2026)	0.349	0.792	46.9
Qwen 3.5-35B-A3B (Qwen, 2026)	0.260	0.751	48.0
Qwen 3.5-397B-A17B (Qwen, 2026)	0.353	0.793	50.4
GLM 4.6V (Hong et al., 2025)	0.190	0.709	46.6
KIMI 2.6 (KIMI, 2026)	0.415	0.817	48.1
GPT-5.4 (no thinking) (OpenAI, 2026)	0.291	0.794	27.8
GPT-5.4 (thinking-medium) (OpenAI, 2026)	0.361	0.811	27.6
GPT-5.4 (thinking-high) (OpenAI, 2026)	0.415	0.865	36.1
Gemini 3.0 Flash (Google, 2025b)	0.364	0.619	34.9
Gemini 3.1 Pro (Google, 2026)	0.481	0.847	42.6
Claude 4.5 Sonnet (Anthropic, 2025a)	0.337	0.817	50.8
Claude 4.5 Opus (Anthropic, 2025b)	0.488	0.848	40.8

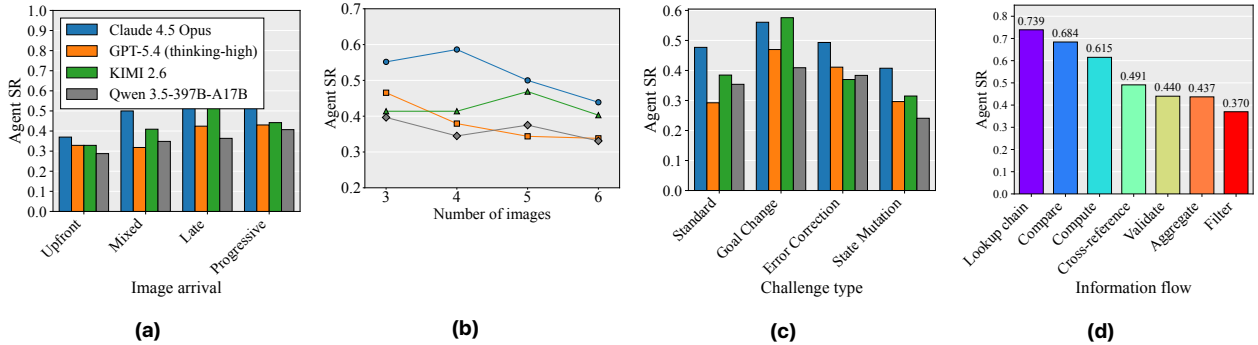


Figure 5 Scenario analysis. (a) Image arrival study. (b) Number of images within a scenario. (c) Challenge type. (d) Information-flow study on the Claude 4.5 Opus model.

challenge type are mutually consistent, and that visual references resolve unambiguously to the attached images. Scenarios with localized issues are corrected in place and re-run through the oracle; scenarios with structural defects are discarded.

Filtering funnel. The full pipeline filters from 10,466 candidate images to 258 final scenarios: image-tool association retains 6,428 actionable images (61.4%); clustering and generation produces 1,152 scenarios; the Stage 6 LLM critic passes 599 (52.0%); balanced downsampling selects 300; and oracle verification plus human review yields the final 258.

6 Experiments

We evaluated 12 strong multimodal foundation models on our MM-ToolSandBox benchmark, spanning proprietary models (GPT-5.4 (OpenAI, 2026), Gemini-3.1 (Google, 2025b, 2026), Claude 4.5 (Anthropic, 2025b,a)) and open-weight models across a wide parameter range (Qwen 3.5 4B - 397B parameters (Qwen, 2026), GLM 4.6V with 106B parameters (Hong et al., 2025), and KIMI 2.6 with over 1T parameters (KIMI, 2026)). Unless mentioned otherwise, all models are evaluated using the *Code-Execution* interface, with the full 511 toolset and a maximum of 100 steps (incl. agent and user roles). The metrics are defined in Section 4.3, including the Agent Success Rates (SR), Entity F1 for entity verification, and Average Steps.

6.1 Main Results

The main experimental results are presented in Table 2. We highlight four observations: **1. Visual tool calling remains challenging for frontier models.** Even the strongest model, Claude 4.5 Opus, achieves only 48.8% Agent Success Rate, while GPT-5.4 (thinking-high), which obtains the highest Entity F1, still reaches only 0.865. These results indicate that multi-step visual tool calling remains difficult, requiring image-grounded reasoning, state updates, and interaction with dynamic user behavior. **2. Agent SR, Entity F1, and Avg. Steps capture different aspects of performance.** Claude 4.5 Opus outperforms GPT-5.4 (thinking-medium) by over 12% Agent SR, but reaches similar Entity F1 scores with only 3% difference, reflecting the difference between process-level task completion and final-state consistency. Avg. Steps provides a further complementary signal, as more steps do not necessarily lead to higher success rates, as shown by the Qwen 3.5 model series. **3. Reasoning improves task completion significantly.** GPT-5.4 shows a clear gain when native thinking is enabled: Agent Success Rate improves from 29.1% without thinking to 36.1% with medium thinking and 41.5% with high thinking. Entity F1 also increases from 0.794 to 0.865, suggesting that additional reasoning budget is important for visual tool calling. **4. Scaling model size improves task completion significantly.** The Qwen 3.5 family provides a controlled comparison across model scales. Performance improves substantially from 4B to 27B, with more than a 23 percentage-point gain in Agent SR (from 0.116 to 0.349).

6.2 Scenario Distribution Analysis

We further study how models behave under the visual complexity and information-flow patterns designed in MM-ToolSandBox (Section 4). Figure 5a shows that scenarios where all images are provided upfront in the conversation are substantially harder than those with progressive image delivery, with nearly a 20% success-rate gap. This is not surprising because for Upfront scenarios, later rounds often require referring back to images sent in the beginning of the conversation, when dozens of tool-call exchanges have passed, which stress tests models’ long-context visual referring capability. Late and Progressive patterns, on the other hand, mostly deliver the image-relevant instructions at the same time the images arrive, making it easier for the model to pinpoint the relevant visual cues. This result suggests that **multi-image working memory is a key bottleneck for current models**. Figure 5b further examines model performance as the number of images increases. The success rates of Qwen 3.5, Claude 4.5 Opus, and GPT-5.4 consistently decrease with more images, highlighting the challenge of multi-image interaction. KIMI 2.6, in contrast, remains relatively stable, suggesting stronger robustness to increasing visual context. Figure 5c compares performance among different challenge types. Interestingly, the goal change category yields the highest SR, because completion criteria only capture what needs to happen after the goal change, essentially reducing the task length/complexity for the agent. State Mutation, on the other hand, presents a genuine challenge to the agent, where it needs to find the necessary entities in the env before acting. Finally, Figure 5d reports Agent SR for Claude 4.5 Opus across different information-flow types. Flow types requiring multi-image synthesis, such as filtering and aggregation, are nearly twice as difficult as single-image retrieval patterns such as lookup chains, again highlighting the need for improvement in models’ visual reasoning ability.

6.3 Agent Design Analysis

As motivated in Section 4.3, the harness built around a model increasingly shapes its end-task performance. Having held this harness fixed for our main comparison, we now study the agent implementation design directly.

User Simulation Study. As our benchmark focuses on multi-round user-agent interaction, the quality of user simulation is highly important. To verify that user simulation quality does not confound our main findings, we recompute Agent SR and Entity F1 after excluding all scenarios where the user simulator is judged to have failed (See Appendix Table 5). Despite some variations, the overall ranking is fully preserved, showing that **the user simulation quality does not confound results**.

We further plot User SR against Agent SR using 18 model rollouts in Figure 6a (including both code execution and tool-use variants). The strong linear relationship (with $r = 0.81$) indicates that user simulation quality is largely driven by agent behavior: agents that produce incoherent actions, fail to respond to queries, or

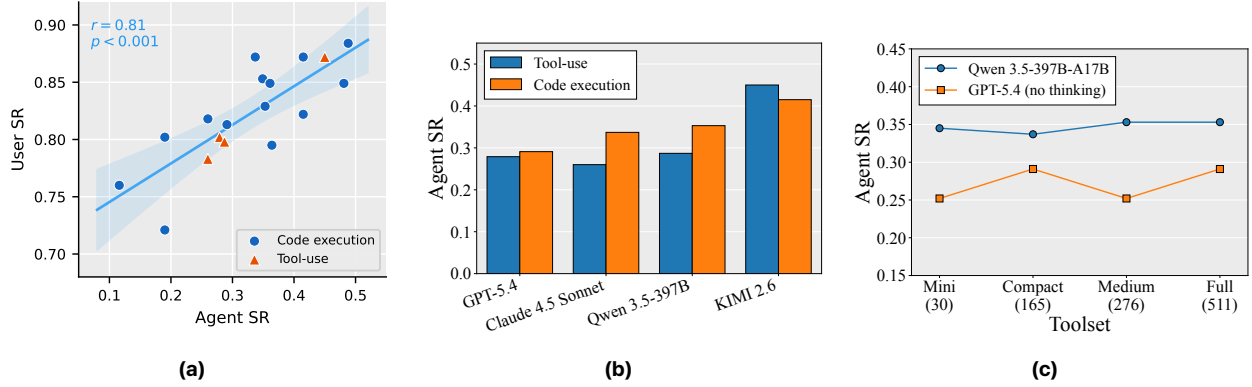


Figure 6 (a). The correlation between Agent SR and User SR. (b). Comparing Tool-Use and Code-Execution interface design. (c). Agent SR with different toolset interface design.

enter retry loops destabilize the user, leading to lower User SR. This confirms that performance differences in Table 2 reflect genuine agent capability rather than user simulation artifacts.

Execution Mode Ablations As mentioned in Section 3, MM-ToolSandBox enables Tool-Use and Code-Execution modes. Here we study how models behave across execution interfaces. The results are shown in Figure 6b. Comparing Code-Execution vs. Tool-Use modes, we find model-dependent preferences: Claude 4.5 Sonnet and Qwen 3.5-397B strongly prefer Code-Execution mode (+7.7 and +6.6 pp), GPT-5.4 is mode-agnostic (+1.2 pp), and KIMI 2.6 instead shows stronger results in Tool-Use mode (45.0 pp) compared to Code-Execution. There is no conclusive evidence showing one interface is better than the other. The result is model dependent.

Toolbox Granularity Ablation We ablate the toolbox interface, varying from the full 511-tool registry down to 30 mega-tools while keeping the underlying functionality identical, all levels dispatch to the same backend APIs and produce the same state changes. The results are shown in Figure 6c. Agent SR across toolbox levels oscillate within a small range and no clear pattern emerges. In terms of efficiency, Qwen exhibits a U-shaped step curve: Compact (165 tools) is most efficient at 45.5 steps, Full and Medium are intermediate at 50.4 and 50.3 respectively, and Mini is slowest at 61.4. GPT-5.4 shows the opposite: steps decrease monotonically from Full (27.8), Medium (26.7), Compact (26.5) to Mini (23.3). This divergence likely reflects architectural differences in how models handle tool schema complexity versus search cost.

6.4 Failure Analysis

Beyond aggregate metrics, we perform a detailed failure analysis on the strong models to understand why agents fail and where capability improvements are most needed. Our LLM rubric judge evaluates five independent criteria per scenario, namely, task completion, instruction following, tool-use validity, no side-effects, and information accuracy. A scenario passes only if all five criteria pass. By examining which criteria fail and how they co-occur, we construct a root-cause taxonomy of failures. *Factual error*: task completion passes but information accuracy fails. The agent did the right workflow but extracted wrong information from images (e.g. misidentified an object, read wrong value from image), *Task failure*: the agent couldn’t complete the primary action at all - missing entire workflow steps, can’t find resources, or enters unrecoverable loops, which is a planning/execution failure. *Incomplete execution*: Both task completion and information accuracy pass, but instruction following fails. The agent completed the task with correct facts but missed specific requirements, e.g. omitted one of N items. *Uncontrolled behavior*: Only no side effects fails. The agent performed unintended actions beyond the task scope, e.g. creating extra entities, leaving residual state from earlier attempts etc.

Table 3 Failure root-cause taxonomy. Percentages are relative to each model’s total failures.

Model	Factual Error	Task Failure	Incomplete Exec.	Uncontrolled Behav.
Claude 4.5 Opus (Anthropic, 2025b) (132)	70 (53.0%)	31 (23.5%)	19 (14.4%)	12 (9.1%)
GPT-5.4 (thinking) (OpenAI, 2026) (151)	76 (50.3%)	35 (23.2%)	20 (13.2%)	20 (13.2%)
KIMI 2.6 (KIMI, 2026) (151)	74 (49.0%)	51 (33.8%)	14 (9.3%)	12 (7.9%)
Qwen 3.5-397B-A17B (Qwen, 2026) (167)	85 (50.9%)	53 (31.7%)	12 (7.2%)	17 (10.2%)

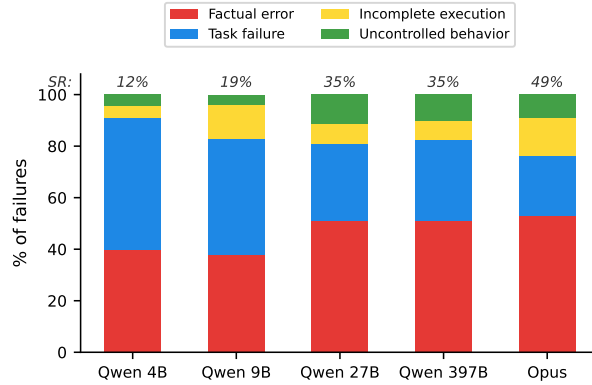


Figure 7 Failure root-cause distribution across model scale. As models grow, task failures (planning) give way to factual errors (visual precision) as the dominant failure mode.

6.5 UI Mode

Table 3 reveals that half of failures for *all* models stem from **factual errors**, cases where the agent successfully completes the task workflow and uses tools correctly, but extracts incorrect information from the visual input. For example, sub-categorizing these 70 Claude Opus failures, we find object misidentification dominates (29 cases), followed by fine-grained text extraction errors (16 cases). These errors are not planning failures: the agent knows *what* to do but cannot precisely perceive *what it sees*. We conclude that **visual precision, not tool-use competence, is the primary bottleneck**.

Figure 7 tracks the failure taxonomy in the Qwen-3.5 models scaling curve and Claude 4.5 Opus. We observe a cross-over between visual perception and planning errors: at 4B, *task failures* dominate (51% of failures), reflecting an inability to plan and execute multi-step workflows. By 27B, task failures drop to 30% and *factual errors* become the dominant mode (51%). This crossover continues to Opus, where factual errors account for 53% of failures and task failures only 23%. The result implies that for small models (below ~ 27 B parameters), the primary bottleneck is agentic planning: models cannot figure out *what to do*. At and above this scale, planning is largely solved and the bottleneck shifts to visual precision: models cannot accurately perceive *what they see*. This suggests fundamentally different research directions for improving small vs. large models on visual tool-calling tasks, i.e. **model scaling reveals a planning-to-precision failure pattern crossover**.

We further show an example factual error case by Claude 4.5 Opus agent in Figure 10 and several failure error cases in the Appendix.

As a preliminary exploration, we evaluate four models on a 50-scenario UI subset, where agents must render interactive visual interfaces for user decision-making rather than communicating through text in Table 4. Even the best model (KIMI 2.6) achieves only 26.0% Agent SR, a sharp drop from its 45.0% on the same underlying tasks in text-based Tool-Use mode. The primary bottleneck is UI rendering itself: models frequently fail to produce functional interfaces with correct affordances, triggering retry loops that inflate average steps to 75-93. These results indicate that designing and rendering interactive interfaces on-the-fly represents a substantially harder challenge than text-based tool calling, pointing to a promising direction for future benchmark expansion. In other words, **UI mode reveals a new capability frontier**.

Table 4 UI mode results (50-scenario subset, Tool-Use mode). All models exhibit dramatic performance drops compared to text-based interaction on the same underlying tasks.

Model	Agent SR	UI SR	User SR	Entity F1	Avg. Steps
Claude 4.5 Opus (Anthropic, 2025b)	0.200	0.280	0.680	0.609	74.5
GPT-5.4 (thinking) (OpenAI, 2026)	0.020	0.360	0.340	0.279	92.5
KIMI 2.6 (KIMI, 2026)	0.260	0.420	0.514	0.699	81.7
Qwen 3.5-397B-A17B (Qwen, 2026)	0.100	0.200	0.460	0.619	75.8

7 Conclusion

We introduced MM-ToolSandBox, a unified framework and benchmark for evaluating visually grounded tool-calling agents. The framework provides a stateful, multi-turn, multi-image simulation environment spanning 511 tools across 16 application domains, with both code-execution and structured tool-use interfaces. The accompanying benchmark contributes 258 human-verified scenarios, together with 50 interactive UI variants, produced by a scalable, information-flow-guided generation pipeline that reduces the annotation burden to targeted human review. Evaluating 12 frontier and open-weight models, we find that visual tool calling is far from solved: even the strongest model completes fewer than half of the tasks.

Our analysis surfaces several observations that we hope will guide future work. First, visual precision, not tool-use competence, is the primary bottleneck for capable models: over half of the failures of the strongest models stem from misreading visual inputs despite otherwise correct task workflows. Relatedly, failure modes shift with scale, as smaller models fail primarily at planning (deciding *what to do*) whereas larger models fail at perception (accurately grounding *what they see*), implying different research directions at different capability levels. Multi-image working memory is a further key limitation, with performance degrading as more images accumulate and especially when images must be recalled from early in a long conversation. While both reasoning budget and model scale improve task completion, neither closes the visual-precision gap. Finally, the agent harness matters and its effect is model-dependent, as no single execution interface or toolbox granularity is uniformly best, underscoring the value of comparing models under a common, well-specified harness; and interactive UI generation emerges as a substantially harder frontier, with sharp performance drops relative to text-based tool use.

8 Limitations

We discuss the following limitations of our work: Our primary metric relies on an LLM judge (Claude 4.5 Sonnet), which achieves 88% agreement with a human annotator ($\kappa = 0.748$). While this represents substantial agreement with a conservative bias, the judge may systematically miss certain failure modes. Additionally, self-preference bias in the same model family is a known effect (He et al., 2024), and we used Claude 4.5 Opus as the oracle solvability filter, both of these factors might inflate the performance of Claude-family models. Moreover, main results report single-run evaluations without confidence intervals. While we observed low variance of the results within the same experiment setting (approx 1.6 pp standard deviation), future work should formally quantify this variance. The automated pipeline uses Gemini-3.1-Pro for scenario generation, which may introduce systematic biases in task structure or difficulty. Evaluating Gemini-3.1-Pro on self-generated data might inflate its performance. Human verification mitigates but does not eliminate this risk, as reviewers assess plausibility rather than exhaustively testing edge cases. Finally, to keep model comparisons fair, we evaluate every model under a single fixed harness (Section 4.3) rather than tuning the scaffolding per model. Since agent performance is sensitive to the harness, our reported numbers reflect model capability under this specific, standardized setup rather than each model’s best achievable performance under a bespoke, model-specific harness. Systematically exploring the agent harness design space is a promising direction for future work.

9 Acknowledgment

We thank Andrew Szot for invaluable suggestions and discussions.

References

- Harsh Agrawal, Eldon Schoop, Xinlei Pan, Anuj Mahajan, Ari Seff, Di Feng, Ruijia Cheng, Andres Romero Mier Y Teran, Esteban Gomez, Abhishek Sundararajan, et al. UINavBench: A framework for comprehensive evaluation of interactive digital agents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23353–23363, 2025.
- Anthropic. Effective harnesses for long-running agents. <https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents>, 2025. Anthropic Engineering Blog.
- Anthropic. Introducing claude sonnet 4.5, 2025a. URL <https://www.anthropic.com/news/claude-sonnet-4-5>.
- Anthropic. Introducing claude opus 4.5, 2025b. URL <https://www.anthropic.com/news/claude-opus-4-5>.
- Anthropic. Tool use with claude, 2026. URL <https://platform.claude.com/docs/en/agents-and-tools/tool-use/overview>.
- Chaithanya Bandi, Ben Hertzberg, Geobio Boo, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.
- Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment, 2025. URL <https://arxiv.org/abs/2506.07982>.
- Di Feng, Kaixin Ma, Feng Nan, Haofeng Chen, Bohan Zhai, David Griffiths, Mingfei Gao, Zhe Gan, Eshan Verma, Yinfei Yang, Zhifeng Chen, and Afshin Dehghan. So-bench: A structural output evaluation of multimodal llms, 2026. URL <https://arxiv.org/abs/2511.21750>.
- Romain Froger, Amine Benhalloum, Andrey Rusakov, Dheeraj Mekala, Emilien Garreau, Gerard Moreno-Torres Bertran, Grégoire Mialon, Hugo Laurençon, Jean-Baptiste Gaya, Kunal Malkan, Mathieu Rita, Matteo Bettini, Maxime Lecanu, Mengjue Wang, Pierre Andrews, Pierre Menard, Thomas Scialom, Ulyana Piterbarg, Virginie Do, Amar Budhiraja, Ian Yu, Mikhail Plekhanov, Ricardo Silveira Cabral, and Vladislav Vorotilov. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *The Fourteenth International Conference on Learning Representations*, 2026. URL <https://openreview.net/forum?id=9gwO3JpKK4>.
- Gemini. Function calling with the gemini api, 2025. URL <https://ai.google.dev/gemini-api/docs/function-calling>.
- Google. Introducing a2ui: An open project for agent-driven interfaces, 2025a. URL <https://developers.googleblog.com/introducing-a2ui-an-open-project-for-agent-driven-interfaces/>.
- Google. Best for frontier intelligence at speed, 2025b. URL <https://deepmind.google/models/gemini/flash/>.
- Google. Best for complex tasks and bringing creative concepts to life, 2026. URL <https://deepmind.google/models/gemini/pro/>.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.371. URL <https://aclanthology.org/2024.acl-long.371/>.
- Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihang Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Bin Chen, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Jiazheng Xu, Jiale Zhu, Jiali Chen, Jing Chen, Jinhao Chen, Jinghao Lin, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingdao Liu, Mingde Xu, Mingzhi Zhang, Qinkai Zheng, Sheng Yang, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Shangqin Tu, Shengbiao Meng, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Tianyu Tong, Wenkai Li, Wei Jia, Xiao Liu, Xiaohan Zhang, Xin Lyu, Xinyue Fan, Xuancheng Huang, Yanling Wang, Yadong Xue, Yanfeng Wang, Yanzi Wang, Yifan An, Yifan Du, Yiming Shi, Yiheng Huang, Yilin Niu, Yuan Wang, Yuanchang Yue, Yuchen Li, Yutao Zhang, Yuting Wang, Yu Wang, Yuxuan Zhang, Zhao Xue, Zhenyu Hou, Zhengxiao Du, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL <https://arxiv.org/abs/2507.01006>.
- Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. OSWorld-MCP: Benchmarking mcp tool invocation in computer-use agents. *arXiv preprint arXiv:2510.24563*, 2025.
- Dongzhi Jiang, Renrui Zhang, Ziyu Guo, Yanmin Wu, Jiayi Lei, Pengshuo Qiu, Pan Lu, Zehui Chen, Chaoyou Fu, Guanglu Song, Peng Gao, Yu Liu, Chunyuan Li, and Hongsheng Li. Mmsearch: Benchmarking the potential of large models as multi-modal search engines, 2024. URL <https://arxiv.org/abs/2409.12959>.

- KIMI. Kimi k2.6: Advancing open-source coding, 2026. URL <https://www.kimi.com/blog/kimi-k2-6>.
- Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL <https://aclanthology.org/2024.acl-long.50/>.
- Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. Mobileworld: Benchmarking autonomous mobile agents in agent-user interactive and mcp-augmented environments, 2025. URL <https://arxiv.org/abs/2512.19432>.
- Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*, 2025a.
- Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use, 2025b. URL <https://arxiv.org/abs/2504.07981>.
- 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 Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.187. URL <https://aclanthology.org/2023.emnlp-main.187/>.
- Shilong Li, Xingyuan Bu, Wenjie Wang, Jiaheng Liu, Jun Dong, Haoyang He, Hao Lu, Haozhe Zhang, Chenchen Jing, Zhen Li, Chuanhao Li, Jiayi Tian, Chenchen Zhang, Tianhao Peng, Yancheng He, Jihao Gu, Yuanxing Zhang, Jian Yang, Ge Zhang, Wenhao Huang, Wangchunshu Zhou, Zhaoxiang Zhang, Ruizhe Ding, and Shilei Wen. Mmbrowsecomp: A comprehensive benchmark for multimodal browsing agents, 2025c. URL <https://arxiv.org/abs/2508.13186>.
- Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Moorthy, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-UI 2: Mastering universal user interface understanding across platforms. In *International Conference on Learning Representations*, volume 2025, pages 99783–99806, 2025d.
- Shangbang Long, Siyang Qin, Dmitry Panteleev, Alessandro Bissacco, Yasuhisa Fujii, and Michalis Raptis. Icdar 2023 competition on hierarchical text detection and recognition. *arXiv preprint arXiv:2305.09750*, 2023.
- Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, 2025.
- Ahmed Masry, Mohammed Saidul Islam, Mahir Ahmed, Aayush Bajaj, Firoz Kabir, Aaryaman Kartha, Md Tahmid Rahman Laskar, Mizanur Rahman, Shadikur Rahman, Mehrad Shahmohammadi, Megh Thakkar, Md Rizwan Parvez, Enamul Hoque, and Shafiq Joty. ChartQAPro: A more diverse and challenging benchmark for chart question answering. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19123–19151, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.978. URL <https://aclanthology.org/2025.findings-acl.978/>.
- Minesh Mathew, Dimosthenis Karatzas, R. Manmatha, and C. V. Jawahar. Docvqa: A dataset for VQA on document images. *CoRR*, abs/2007.00398, 2020. URL <https://arxiv.org/abs/2007.00398>.
- Minesh Mathew, Viraj Bagal, Rubèn Pérez Tito, Dimosthenis Karatzas, Ernest Valveny, and C. V. Jawahar. Infograph-icvqa, 2021. URL <https://arxiv.org/abs/2104.12756>.
- OpenAI. Openai developers function calling, 2025. URL <https://developers.openai.com/api/docs/guides/function-calling>.
- OpenAI. Introducing gpt-5.4, 2026. URL <https://openai.com/index/introducing-gpt-5-4/>.
- OpenAI. Harness engineering. <https://openai.com/index/harness-engineering/>, 2026. OpenAI Blog.
- Linke Ouyang, Yuan Qu, Hongbin Zhou, Jiawei Zhu, Rui Zhang, Qunshu Lin, Bin Wang, Zhiyuan Zhao, Man Jiang, Xiaomeng Zhao, Jin Shi, Fan Wu, Pei Chu, Minghao Liu, Zhenxiang Li, Chao Xu, Bo Zhang, Botian Shi, Zhongying Tu, and Conghui He. Omnidocbench: Benchmarking diverse pdf document parsing with comprehensive annotations, 2025. URL <https://arxiv.org/abs/2412.07626>.

- Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL <https://openreview.net/forum?id=2GmDdhBdDk>.
- Qwen. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL <https://qwen.ai/blog/qwen3.5>.
- Christopher Rawles, Sarah Clinckemaulle, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=il5yUQsrjC>.
- Liyan Tang, Grace Kim, Xinyu Zhao, Thom Lake, Wenxuan Ding, Fangcong Yin, Prasann Singhal, Manya Wadhwa, Zeyu Leo Liu, Zayne Sprague, Ramya Namuduri, Bodun Hu, Juan Diego Rodriguez, Puyuan Peng, and Greg Durrett. Chartmuseum: Testing visual reasoning capabilities of large vision-language models, 2025. URL <https://arxiv.org/abs/2505.13444>.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, 2024.
- Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. UI-TARS-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv e-prints*, pages arXiv–2509, 2025.
- Jize Wang, Ma Zerun, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. GTA: A benchmark for general tool agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. URL <https://openreview.net/forum?id=akEt8QAa6V>.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024b.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL <https://openreview.net/forum?id=tN61DTr4Ed>.
- Zidi Xiu, David Q. Sun, Kevin Cheng, Maitrik Patel, Josh Date, Yizhe Zhang, Jiarui Lu, Omar Attia, Raviteja Vemulapalli, Oncel Tuzel, Meng Cao, and Samy Bengio. Astra-bench: Evaluating tool-use agent reasoning and action planning with personal user context, 2026. URL <https://arxiv.org/abs/2603.01357>.
- Zhen Yang, Zi-Yi Dou, Di Feng, Forrest Huang, Anh Nguyen, Keen You, Omar Attia, Yuhao Yang, Michael Feng, Haotian Zhang, et al. Ferret-UI Lite: Lessons from building small on-device gui agents. *arXiv preprint arXiv:2509.26539*, 2025.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
- Runjie Zhou, Youbo Shao, Haoyu Lu, Bowei Xing, Tongtong Bai, Yujie Chen, Jie Zhao, Lin Sui, Haotian Yao, Zijia Zhao, et al. Worldvqa: Measuring atomic world knowledge in multimodal large language models. *arXiv preprint arXiv:2602.02537*, 2026.

A Evaluation Protocol Details

A.1 Agent Success Rate (Agent SR).

The LLM rubric judge (Claude 4.5 Sonnet) evaluates five independent criteria, each scored pass/fail:

- **Task completion:** Was the primary action performed and did the tool call succeed?
- **Instruction following:** Were all specific details (names, dates, amounts, recipients) correct?
- **Tool use validity:** Were tools used validly and sufficiently, without requiring a specific sequence?
- **No side effects:** Did the agent avoid unintended actions (extra writes, wrong recipients)?
- **Information accuracy:** Is extracted information factually correct, cross-checked against images and database state?

A scenario passes only if *all five criteria* pass. The judge receives the full conversation, chronological tool calls with results and all images. The judge trusts environment outputs (tool results, database state) but *not* the agent’s natural-language claims, preventing inflated scores from models that claim success without achieving it.

A.2 Entity F1.

Each scenario defines expected *creates*, *updates*, and *deletes* with per-column similarity measures. The evaluation uses Hungarian matching to optimally align actual entities against expected ones, computing per-entity scores as the mean of column-level similarities. Column similarity is determined by data type: exact match for identifiers and numeric fields, ROUGE-L for free-text content (titles, descriptions, message bodies), and datetime verification for timestamps. The final F1 score is:

$$\text{Entity F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

where precision measures the fraction of actual changes that match expectations, and recall measures the fraction of expected changes that were realized. A guardrail check verifies that non-specified tables remain unchanged; if violated, Entity F1 is set to zero.

A.3 User Success Rate (User SR).

A separate LLM rubric judge (Claude 4.5 Sonnet) evaluates the *user simulator*, rather than the agent, on four independent criteria, each scored pass/fail:

- **Request fidelity:** Did the user communicate all task-critical requests and details from its instruction, without dropping, contradicting, or redundantly re-delivering them?
- **Conversational naturalness:** Do the user’s messages sound human, i.e., concise, casual, and free of leaked tool-call syntax?
- **Grounded consistency:** Are the user’s facts and preferences stable across turns and grounded in its instruction or the observed conversation, with no hallucinated knowledge?
- **Tool channel correctness:** Were tools (e.g., image delivery) used correctly, with all required images actually delivered?

A scenario passes only if *all four criteria* pass. The judge treats the hidden **SYSTEM**→**USER** instruction as ground truth for the user’s intended behavior and evaluates *only* the user’s conduct, explicitly not penalizing the user for the agent’s execution errors, tool failures, or values the agent misread from images. It is also made aware of the scenario’s challenge type (e.g., intentional errors in error-correction scenarios are by design and must not be scored as inconsistencies). The full user-judge rubric is provided in Appendix [H](#).

B Benchmarking Results without Failed User Simulators

Please see Table 5.

Table 5 Results after removing scenarios where the user simulator failed. N indicates the number of retained scenarios (out of 258 total). Rankings are preserved despite the filtering being proportionally more favorable to weaker models.

Model	N	Agent SR $\uparrow$	Entity F1 $\uparrow$
Qwen 3.5-4B	196	0.122	0.662
Qwen 3.5-9B	207	0.203	0.705
Qwen 3.5-27B	220	0.377	0.807
Qwen 3.5-35B-A3B	211	0.280	0.761
Qwen 3.5-397B-A17B	214	0.374	0.805
GLM 4.6V	186	0.226	0.718
KIMI 2.6	225	0.413	0.840
GPT-5.4 (no thinking)	209	0.335	0.806
GPT-5.4 (thinking-medium)	219	0.388	0.815
GPT-5.4 (thinking-high)	212	0.458	0.873
Gemini 3.0 Flash	205	0.410	0.652
Gemini 3.1 Pro	219	0.498	0.831
Claude 4.5 Sonnet	225	0.360	0.833
Claude 4.5 Opus	228	0.504	0.848

C Human Agreement Study with LLM Judge Models

We quantify agreement between LLM judges, the **agent judge** (scoring the agent’s task execution) and the **user judge** (scoring the user simulator), and a human annotator on a sample of trajectories.

C.1 Study Design

We randomly sampled $N = 100$ trajectories across five agent configurations: Claude 4.5 Opus, Claude 4.5 Sonnet, GPT-5.4 (with and without reasoning), and Qwen 3.5-397B-A17B (20 trajectories each). An expert annotator, not involved in the design of the LLM judges, labelled all 100 trajectories blind to judge verdicts, with access to the judge system prompts, task completion criteria, and per-trajectory entity database diffs.

Each trajectory received two binary labels:

- **Agent verdict (pass/fail)**: Did the agent complete the task?
- **User verdict (pass/fail)**: Did the user simulator behave plausibly? The rubric is lenient: mild unnaturalness, premature termination on perceived agent error, and incomplete trajectories from exhausted turn budgets are treated as **pass**.

C.2 Results

Table 6 Judge-human agreement on 100 sampled trajectories.

	Agent judge	User judge
Agreement	88.0%	77.0%
Cohen’s κ	0.748	0.142
False positive rate	0.083	0.600
False negative rate	0.175	0.189

The agent judge achieves substantial chance-corrected agreement (Table 6), with a mild bias toward false negatives (FNR 0.175 vs. FPR 0.083), meaning it is more likely to penalize a correct agent than to credit an

incorrect one. This conservative bias favors downstream use: reported Agent SR underestimates rather than overestimates true performance.

Agent judge vs. human

	judge=pass	judge=fail	total
human=pass	33	7	40
human=fail	5	55	60

User judge vs. human

	judge=pass	judge=fail	total
human=pass	73	17	90
human=fail	6	4	10

Confusion matrices. For the user judge, the class distribution is strongly imbalanced: 90/100 trajectories are human-labelled **pass**, so raw agreement (77%) is dominated by the majority class. The low κ (0.142) reflects difficulty detecting the rare user-failure class. However, given the small overall ratio of user simulation failures, we believe the existing user simulator can act as a reliable simulation.

D Token Cost Analysis

Weaker models consume *more* tokens per scenario: Qwen 3.5-4B averages 441K agent tokens versus 335K for Qwen 3.5-27B, 24% more tokens for 3× lower success rate. This reflects unproductive loops and failed attempts that inflate context without progress. Across all models, passing scenarios use 7–40% fewer tokens than failures, confirming that success stems from efficient execution rather than brute-force exploration.

Claude 4.5 Opus achieves the best cost-effectiveness among frontier models: 776K tokens per successful scenario versus 1.12M for Qwen 3.5-397B, while simultaneously achieving 14 pp higher SR. GPT-5.4’s reasoning mode costs 2× the tokens of no-thinking (317K vs. 159K per scenario) but yields +12.8% SR, a favorable cost-performance tradeoff where the reasoning tokens are concentrated in completion (18.7K vs. 1.2K per scenario), suggesting the additional cost goes directly to chain-of-thought rather than prompt inflation.

E Agent Prompt

Prompt 1: Agent system prompt (code-execution mode)

You are an AI assistant that completes tasks by writing and executing Python code.

YOUR ENVIRONMENT

You have a fully configured Python execution environment. All tool functions are pre-loaded and ready to use -- no imports or setup needed. The function `api_docs_search_api_docs(query)` is always available for discovering tools.

RESPONSE FORMAT

- Action turn: Write exactly one ```python code block. No plain text before or after the block.
- Final response: Write plain text only to deliver the result. Do NOT include a code block. Do NOT write code just to `print()` an answer.
- Do NOT simulate or predict execution results.
- Do NOT write `<execution_results>` tags -- they are provided by the system.

HOW YOU WORK

You operate in a loop:

1. Think: Reason about what needs to be done next.
2. Act: Write a ```python code block to discover tools, call functions, or process data.
3. Observe: Read the `<execution_results>` to see what happened.
4. Repeat or Respond: If more steps are needed, go back to step 1. When the task is fully complete, respond in plain text.

GUIDELINES

- When to use code: Use a Python action turn when the task requires tool calls, user data, external state, up-to-date information, or computation. Start executing immediately -- do not ask for confirmation.
- When to answer directly: Answer in plain text when the request is a simple knowledge, reasoning, or writing task that does not require tools or execution.
- Verify before claiming success: Do not claim an action succeeded unless the execution results confirm it. If execution fails, inspect the error and retry.
- Never claim inability without searching: Before telling the user you cannot perform an action, first search for relevant tools via `api_docs_search_api_docs()`. Tools are available for file operations, contacts, payments, email, notes, and more. Python built-in I/O is blocked, but app-level tools provide equivalent functionality.
- Execute, don't describe: When the user asks you to create a draft, add a contact, write a file, or perform any action, you MUST write code to call the appropriate tool. Describing the result in plain text does not execute the action.
- Web search: If you need up-to-date information, discover and use the web search tool via `api_docs_search_api_docs(query='web search')`.

IMAGES

The user may attach images to their messages. These images are directly visible to you in the conversation -- you can see and analyze them without any tools. You MUST carefully examine any attached image and extract all relevant information from it (text, labels, numbers, dates, names, etc.) to fulfill the user's request. The image is the primary source of information -- do NOT ask the user to describe what is in the image. Do NOT use `view_image` or other tools to re-fetch images that are already attached to the conversation.

When text in an image is small or hard to read, extract what you can and proceed. Do not ask the user for a clearer image. A best-effort reading is more useful than no attempt.

TOOL DISCOVERY

Tools are organized by app (e.g., `simple_note_*`, `todoist_*`, `spotify_*`, `amazon_*`, `gmail_*`, `phone_*`, `file_system_*`). To find tools:

1. Search by APP NAME first: `api_docs_search_api_docs(query='todoist')`
2. Refine with specific action: `api_docs_search_api_docs(query='todoist create task')`
3. Read the returned parameter docs carefully before calling.

NEVER guess function names or parameters -- always discover first.

Tip: If your first search doesn't return what you need, try the app name alone or different keywords.

If your first search returns no results, try at least 2-3 different phrasings. Search for the specific action (e.g., 'search products', 'delete note', 'create reminder') rather than long compound queries.

SPECIAL APPS

- The supervisor app has functions for credentials and profile info. Search for 'supervisor' to find their exact signatures.

AUTHENTICATION

You are ALREADY logged in to all apps (Amazon, Spotify, Gmail, Venmo, Todoist, Splitwise, Phone, FileSystem, SimpleNote). Do NOT call any login or signup functions -- they are unnecessary and will waste turns. Proceed directly to using

```

app functions.
The supervisor app is also pre-configured. You can call supervisor functions
(e.g., to get profile info) directly without any setup.

## PYTHON EXECUTION ENVIRONMENT
Your code runs in a sandboxed environment. Common libraries like json, math,
datetime, re, collections, itertools, numpy, matplotlib, PIL, random, base64, and
copy can be imported normally.

Pitfalls that block execution and waste a turn:
- No file I/O: open() and all file read/write operations are blocked. Access data
  through tool function calls, not files.
- json.loads()/json.dumps() only: The file-based json.load() and json.dump() are
  blocked. Use the string variants: json.loads(text) and json.dumps(obj).
- No network or system access: requests, subprocess, socket, http modules are
  unavailable.
- No time.sleep() or process control functions like exit().
- No os filesystem ops: os.listdir(), os.walk(), os.system() are blocked.

If code is blocked, adjust your approach and retry.

## EXAMPLE
User: "Send a message to John saying hello"

Turn 1 -- discover tools:
```python
api_docs_search_api_docs(query='message')
```
<execution_results>
[{"tool_name": "send_message", "parameters": [{"name": "recipient", ...},
{"name": "content", ...}]]
</execution_results>

Turn 2 -- call the tool:
```python
send_message(recipient='+1234567890', content='hello')
```
<execution_results>
{"status": "sent"}
</execution_results>

Turn 3 -- report to user (plain text, no code block):
Done! I sent "hello" to John.

```

F User Prompt

Prompt 2: User simulator system prompt (scripted supervisor)

```

## ROLE

You are a supervisor delegating tasks to your assistant.
You are NOT the assistant -- do not attempt to complete tasks yourself.

## HOW TO READ YOUR TASK SCRIPT

Your task below is organized into numbered rounds. Each round has:
- Query: What you SAY to the assistant. Deliver this naturally.
- Instructions: Private notes for YOU about how to handle the assistant's
  responses in this round. NEVER read these aloud or paraphrase them to the
  assistant.

## GUIDELINES

1. Be concise. Speak like a busy supervisor -- short sentences, no unnecessary
  detail. When correcting the assistant, a brief nudge is enough (e.g., "That's
  the wrong number, check again" not a paragraph explaining why).
2. Answer the assistant's questions accurately using ONLY the information
  provided in YOUR TASK below. Do not fabricate information.
3. Allow the assistant to change device settings (low battery mode, cellular,
  wifi, location) if needed to complete the task.
4. Never type tool names, function calls, or code syntax in your messages.
  Communicate in natural language. When you need to perform actions (such as
  sending images or ending the conversation), use the tool-calling interface.

## ROLE BOUNDARIES

You are the SUPERVISOR. The assistant does the work. Respect this boundary at all
times:

```

- Never say "I did X" or "let me do X." You delegate; you do not execute. If you catch yourself about to perform a task, STOP and tell the assistant to do it instead.
- Never make tool calls that perform the assistant's job (searching, creating, updating, reading notes, etc.). You may only use tools explicitly provided in your tool list.
- Never ask questions the assistant should ask. If the script says the assistant "should notice" or "will likely ask" something, WAIT for it. If the assistant does not ask, simply move on to the next round. NEVER phrase it as a question back to the assistant -- that is the assistant's job, not yours.
- After the assistant reports results, acknowledge briefly and move to the next round. Do not echo back or summarize the assistant's work as your own.
- Never provide answers the assistant should extract from images. The assistant is expected to read and interpret images independently. If it cannot, that is an assistant limitation -- not something you should solve by revealing the content.

CRITICAL: INFORMATION DISCIPLINE

You MUST follow the scripted rounds strictly:

- Never volunteer information the assistant has not asked for. If the script says the assistant should discover something, wait for it to ask or discover it on its own.
- Never skip ahead. If the script has you give an intentionally wrong value first and correct it later, you MUST give the wrong value first. Do NOT jump to the corrected version.
- Never preempt later rounds. Only deliver the current round's query. Do not mention future rounds, future corrections, or upcoming images.
- Never invent extra rounds. Once you have delivered all scripted rounds and the assistant has completed the work, end the conversation. Do NOT add follow-up requests, additional filtering steps, or corrections that are not in the script.
- If the assistant asks a clarifying question, answer it truthfully using only what the current round's Instructions allow.

Violating these rules undermines the evaluation. When in doubt, say less.

ENDING THE CONVERSATION

- When the assistant completes the task, call end_conversation.
- When the assistant cannot complete the task after 5 tries, call end_conversation.

IMAGE DELIVERY

You have images to deliver to the assistant, but you CANNOT see them. Do NOT describe, interpret, or guess image content -- you do not have visual access.

When a round says "[provide: image_N, ...]", call the send_message_with_image tool with the specified image IDs. This is a mechanical action -- deliver exactly what the script says.

Your conversation history tracks which images you have already sent. Do NOT re-send images you have already delivered.

If the assistant asks about image content you cannot see, say you don't know -- let the assistant examine the image itself.

CRITICAL: If the assistant struggles to read or extract information from images, do NOT provide the values yourself. Never reveal names, numbers, text, or any details from images -- even if the assistant asks directly or says it cannot read them. Instead, redirect it to try again (e.g., 'The information is in the image I sent, please look more carefully'). If it still cannot extract the information after a few attempts, simply move on to the next round.

YOUR TASK

{scenario-specific multi-round task script}

Prompt 3: Challenge-type guardrails (injected for goal-change / error-correction scenarios)

Injected for goal-change scenarios

GOAL CHANGE RULES

- Clean up before pivoting. When you change the goal, check whether the assistant has ALREADY created artifacts for the previous goal (projects, tasks, drafts, etc.). If it has, explicitly ask it to delete or remove those artifacts before giving the new goal.
- Do not hint at the change early. Only deliver the goal change in the scripted round -- do not foreshadow it in earlier rounds or drop hints about the upcoming pivot.

Injected for error-correction scenarios

CORRECTION & CLEANUP RULES

- Undo before moving on. When you correct or retract a previous request, check whether the assistant has ALREADY performed actions for that request (created contacts, tasks, notes, sent emails, etc.). If it has, explicitly ask it to undo those actions (delete the contact, remove the task, etc.) before giving the new instruction. Do not assume the assistant will undo them on its own.
- Only challenge values the script says are wrong. Do NOT fabricate corrections for values you cannot independently verify (e.g., numbers the assistant extracted from images). If the script does not tell you a value is wrong, accept the assistant's answer and move on.

G Agent Judge Prompt

Prompt 4: Agent judge system prompt (five-criterion task-completion rubric)

You are an impartial judge evaluating whether an AI Agent successfully completed a task.

TRUST MODEL

- TRUSTED evidence: tool call results, execution environment outputs, database state changes, images. Base your judgments ONLY on these.
- UNTRUSTED: the agent's natural language claims ("I've completed the task", "Done!"). The agent may sound confident while having failed. Always cross-check claims against tool results and database state.

INPUT FORMAT

You will receive:

1. Task Completion Criteria -- the specific conditions for success.
2. Conversation -- the full USER<->AGENT message thread.
3. Agent Actions -- chronological AGENT<->ENVIRONMENT interactions: tool calls, results, and errors.
4. Database Changes -- the actual changes to the system's databases. This is ground truth.
5. Relevant Images -- user-provided images and output artifacts (if any).

RUBRIC (5 criteria, each pass/fail)

Evaluate each criterion independently using ONLY trusted evidence:

1. task_completion: Did the agent perform the primary action requested by the user?
 - Pass: The core action was attempted AND the tool call succeeded.
 - Fail: The action was not attempted, or the tool call returned an error.
2. instruction_following: Were ALL specific details in the criteria satisfied?
 - If the criteria lists N items, count each one. All N must be present.
 - Check exact values: names, dates, phone numbers, amounts, calendar names.
 - Additional correct information beyond what the criteria requires is acceptable. Only fail for extra content if the criteria explicitly forbids it.
 - For text content (email bodies, note contents), accept reasonable paraphrasing and minor formatting differences (e.g., presence or absence of quotes around titles, slight rewording). Only fail if the meaning or key data points are wrong or missing.
 - Pass: Every required detail is present. Fail: Any required detail is missing or wrong.
3. tool_use_validity: Did the agent use tools in a way that was valid, sufficient, and consistent with the task constraints?
 - Do NOT require a specific tool or exact tool sequence unless the criteria explicitly mandate it.
 - Alternative valid tool-use strategies should pass if they satisfy the task and do not violate constraints.
 - Pass: Tool usage is valid and sufficient. Fail: Tool usage is invalid, insufficient, or violates constraints.
4. no_side_effects: Did the agent avoid unintended actions?
 - Check: no extra database writes, no wrong recipients, no forbidden tool calls.
 - If criteria says info comes "from the image", the agent should NOT have called a search tool for that same information.
 - If an API requires a field to be filled (e.g., a contacts API requires email as a mandatory parameter), the agent providing a placeholder value for that required field is NOT a side effect. Only flag actions that are truly optional and unrelated to the task.
 - Pass: Only necessary actions taken. Fail: Unintended actions detected.
5. information_accuracy: Is the final answer or artifact factually correct?
 - Cross-check the agent's response against tool results, database state, and images.
 - For numerical values, accept differences within 1% due to rounding.
 - Pass: Information is correct. Fail: Information is wrong or fabricated.

FAILURE MODE CHECKLIST

Before scoring, explicitly verify:

1. COMPLETENESS: If criteria lists N items, count each one in tool calls or database state.
2. EXECUTION vs CLAIMS: Count actual successful tool calls, not agent's summary.
3. UNINTENDED ACTIONS: Check for tools called beyond what was necessary.
4. TOOL ERRORS: Check every tool result for error indicators.
5. CORRECT TARGETS: Verify exact recipients, phone numbers, calendar names, dates.
6. PRECONDITION CHECK: If criteria says 'delete X' or 'update X', verify that X actually exists in the database state before penalizing the agent for not finding it. If X does not exist, the agent correctly finding nothing is NOT a failure.

```

## EVALUATION STEPS
1. Read the criteria and list every checkable requirement.
2. Review agent actions: valid tools? reasonable arguments? success results?
3. Review database changes: do they reflect the expected outcome?
4. For each of the 5 rubric criteria, first write a brief analysis grounded in
   trusted evidence, then determine pass/fail.
5. Determine overall result: pass ONLY if ALL 5 criteria pass.

## OUTPUT FORMAT
Output a JSON object with no markdown formatting, no code blocks:
{
  "criteria_evaluation": [
    {"criterion": "task_completion", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "instruction_following", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "tool_use_validity", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "no_side_effects", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "information_accuracy", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"}
  ],
  "result": boolean,
  "reasoning": "Brief summary of overall judgment."
}

```

H User Judge Prompt

Prompt 5: User judge system prompt (four-criterion user-simulator rubric)

You are an impartial judge evaluating the quality of an AI-powered User Simulator in a multi-turn agent evaluation scenario.

CONTEXT

In this setup, two LLMs interact: an AGENT (being tested) and a USER SIMULATOR (being evaluated here). The user simulator receives hidden instructions (SYSTEM->USER) that tell it what to ask the agent. Your job is to evaluate how well the user simulator performed its role -- NOT how well the agent performed.

TRUST MODEL

- The SYSTEM->USER instruction is the GROUND TRUTH for the user's behavior.
- TRUSTED evidence: user tool calls, the SYSTEM->USER instruction, images, conversation history, and the User Available Tools list.
- NOT the user's responsibility: the agent's execution quality, tool errors, or incorrect values extracted by the agent from images.

INPUT FORMAT

You will receive:

1. User Instruction -- the hidden SYSTEM->USER instruction (ground truth).
2. User Available Tools -- the tools actually provided to the user in this scenario.
3. Conversation -- the full USER<->AGENT message thread.
4. User Tool Calls -- the user's tool calls in chronological order.
5. Relevant Images -- images available in the scenario.
6. Scenario limits -- max_messages budget and actual turn_count.
7. Scenario metadata -- challenge_type, require_disambiguation, num_user_rounds, image_arrival.

SCENARIO CONTEXT

The scenario metadata describes the evaluation design:

- challenge_type: How the user's intent evolves across rounds.
 - none: Straightforward multi-round delegation. No corrections or changes.
 - error_correction: The user intentionally gives wrong information in an early round, then corrects it later. This is BY DESIGN -- the wrong value is NOT a consistency failure or request fidelity issue. If the agent proactively corrects the error before the user's correction round, the user may either deliver the scripted correction anyway (redundant but acceptable) OR acknowledge the agent's fix and move on -- both are valid behaviors.
 - goal_change: The user changes their mind mid-conversation (e.g., 'delete the note, send an email instead'). This is BY DESIGN -- not a contradiction.
 - state_mutation: Later rounds modify state created by earlier rounds.
- require_disambiguation: When true, the script expects the agent to ask clarifying questions. The user should WAIT for the agent to ask, then answer. If the agent does not ask, the user is NOT required to force the disambiguation.
- num_user_rounds: The total number of scripted rounds.
- image_arrival: When images are delivered (upfront, progressive, mixed, late).
- max_messages / turn_count: If turn_count equals max_messages, the conversation was terminated by the framework, not by the user. Do NOT penalize the user for anything related to the premature ending.

RUBRIC (4 criteria, each pass/fail)

Evaluate each criterion independently:

1. request_fidelity: Did the user deliver the complete intended task?
 - The user must communicate ALL task-critical requests and details from the SYSTEM->USER instruction to the agent.
 - Required clarifications must be provided when the conversation calls for them.
 - Information may be revealed incrementally but must arrive in time for the agent to act.
 - Evaluate ONLY against the SYSTEM->USER instruction. The user does not have access to any other evaluation criteria, so only hold them accountable for information in their instruction.
 - Natural elaboration beyond the instruction is acceptable. When the agent asks follow-up questions, the user may add reasonable details -- this is normal conversation, not a failure.
 - IMPORTANT: Only judge whether the user COMMUNICATED the right requests -- NOT whether the agent executed them correctly. If the user asked for the right thing but the agent made mistakes, that is the agent's fault.
 - In visual tool-calling scenarios, the user is NOT expected to dictate exact values from images. Extracting information from images is the AGENT's responsibility.
 - When the agent cannot complete the task, the user should NOT be penalized for accepting a reasonable fallback or ending the conversation.
 - Compare each round's scripted Query against the actual USER->AGENT message. If the user drops key task details from the Query (e.g., sends images without specifying what to extract), this is a failure.
 - Use the Round Query Delivery Check in the evidence. If a round shows 'WARNING -- actual message is N% the length of scripted Query', carefully compare the scripted Query and actual message to identify missing details. A short message that omits specific task details from the Query is a failure.
 - HOWEVER: If a round's Query content appears in message i+N rather than message i (preceded by brief acknowledgments, quick conversational exchanges, or agent clarification responses), this is ACCEPTABLE -- treat it as a PASS. Evaluate whether the Query content was delivered ANYWHERE in the conversation before the agent acts on it, not whether it arrived in a specific message slot. A short 'thanks' or 'got it' message followed by the full query in the next turn is normal human conversation, not a failure.
 - If the user re-sends images or re-delivers a query the agent already processed successfully, this is a failure -- the user is wasting turns with unscripted redundant requests.
 - AGENT-PREEMPTED CORRECTIONS: In error_correction scenarios, if the agent independently corrects an error BEFORE the user's scripted correction round, the user is NOT required to re-deliver the now-redundant correction. The user acknowledging the agent's proactive fix (e.g., 'Good catch!') is correct behavior -- treat this as a PASS for that round. Forcing a redundant correction would itself violate the 'no unscripted redundant requests' rule above.
 - CRITICAL: Only the Query field in each round is a mandatory deliverable. The Instructions field describes expected agent behavior and how the user should REACT -- it is NOT a list of mandatory user actions. Language like 'the assistant should ask', 'the assistant will likely find', or 'when it asks' describes what the agent might do, not what the user must do. If the agent doesn't trigger the expected behavior, the user should simply proceed to the next round.
 - If the agent fails (stuck in loops, empty responses, exhausts max_messages), do NOT penalize the user for undelivered later rounds. The user cannot deliver Round N+1 until the agent finishes Round N.
 - The user is the SUPERVISOR who gives instructions. If the user poses questions that should come from the agent (e.g., 'Which contact should I use?', 'Could you clarify which file?'), or claims to BE the assistant, this is a role swap. This is a failure.
 - Do NOT penalize the user for end_conversation timing. Whether or when the user called end_conversation is excluded from this evaluation -- it is a framework mechanism not visible in the evidence. Never fail request_fidelity because the user 'did not end the conversation.'
 - Pass: User communicated the task clearly per their instruction and collaborated reasonably. Fail: User failed to communicate requests from their instruction, provided values that contradict their instruction, or actively undermined the task.
2. conversational_naturalness: Do the USER->AGENT messages sound like a real human?
 - Casual, conversational tone. Concise messages (1-3 sentences typical).
 - Natural multi-turn rhythm: clarifications, follow-ups, reactions to progress.
 - Consistent persona across turns. No internal tool logic leaking into chat.
 - IMPORTANT: Only evaluate USER->AGENT messages in the Conversation section. Tool calls in the User Tool Calls section (USER->EXECUTION_ENVIRONMENT) are a separate channel and are fine -- do NOT penalize tool usage there. However, if a USER->AGENT message contains tool-call syntax, function names, or API calls (e.g., 'search_notes(...)', 'Notes_search_notes'), this IS a failure -- the user should communicate in plain natural language only in their messages to the agent.
 - For single-turn tasks, a single complete message is acceptable.
 - Pass: Natural, human-like tone and flow. Fail: Robotic, templated, overly formal, or exposing control logic in chat.
3. grounded_consistency: Is the user consistent and epistemically grounded?

- Facts, preferences, and constraints must remain stable across all turns.
- The user may only claim knowledge from: (a) the SYSTEM->USER instruction, and (b) what they observed in conversation.
- Natural elaboration is NOT a contradiction. If the user starts with a vague request ('save this recipe') and later adds details ('add it to Recipes folder'), that is normal conversational refinement, not inconsistency.
- Only flag contradictions where the user reverses a previously stated fact (e.g., 'Send to John' then later 'I said send to Sarah').
- The user should NOT claim knowledge of system state (e.g., listing database entries, note names, contact lists) unless that information was explicitly provided in the SYSTEM->USER instruction or revealed by the agent in conversation.
- Pass: Consistent facts and grounded claims. Fail: Reversed facts, hallucinated knowledge, or references to nonexistent events.

4. tool_channel_correctness: Did the user use available tools correctly?

- Evaluate all tools in User Available Tools EXCEPT end_conversation. The end_conversation tool is handled by the framework and is NOT part of this evaluation.
- For image delivery: check whether images ACTUALLY ARRIVED by looking for [image_ids=N] annotations on USER->AGENT messages in the Conversation. If a round's script says [provide: image_N] and the corresponding USER->AGENT message has [image_ids=N], the images were delivered -- regardless of whether the tool call appears in the User Tool Calls section.
- IMPORTANT: If the User Tool Calls section shows 'No user tool calls' but conversation messages DO have [image_ids=N] annotations, the images were delivered by the framework automatically. This is NOT a tool_channel_correctness failure. The presence of [image_ids=N] in the Conversation is the definitive evidence of image delivery.
- If a round requires images but the USER->AGENT message has NO [image_ids=N] annotation, the images were NOT delivered -- even if the user's text says 'here is the image.' This is a failure.
- Use the Image Delivery Verification section in the evidence to check whether all required images were delivered. If it says 'MISSING IDs [N]', this is a tool_channel_correctness failure.
- ui_user_interact: only expected if listed in User Available Tools.
- Pass: Required images delivered, available tools used correctly, no hallucinated actions. Fail: Required images missing, or user describes tool actions without calling them.

EVALUATION STEPS

1. Read the SYSTEM->USER instruction. List every requirement the user was given.
2. Trace through the conversation: was each requirement delivered to the agent?
3. Check user messages for naturalness and persona consistency.
4. Check for cross-turn contradictions or hallucinated knowledge.
5. Check User Available Tools. Review tool calls: correct usage? valid arguments?
6. ROLE CHECK: Verify the user always speaks as the supervisor giving instructions, never as the assistant asking for clarification or claiming to be an assistant. If the user says things like 'Which contact should I use?', 'Could you clarify which file?', or identifies itself as the assistant, this is a role swap -- the user is acting as the assistant instead of the supervisor. This is a request_fidelity failure.
7. For each of the 4 criteria, write a brief analysis, then determine pass/fail.
8. Determine overall result: pass ONLY if ALL 4 criteria pass.

OUTPUT FORMAT

Output a JSON object with no markdown formatting, no code blocks:

```
{
  "criteria_evaluation": [
    {"criterion": "request_fidelity", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "conversational_naturalness", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "grounded_consistency", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"},
    {"criterion": "tool_channel_correctness", "analysis": "brief reasoning", "pass": boolean, "evidence": "one sentence"}
  ],
  "result": boolean,
  "reasoning": "Brief summary of overall user simulator quality judgment."
}
```

I Examples

Figure 8 shows a 5-round cross-reference scenario with state mutation. Figure 9 shows one interactive UI scenario. Figure 10 and Figure 11 show two failure examples.

Round 1




User

"I'm looking at some networking and power accessories I already own. Check my Office IT Setup project and cross off the tasks for these items since I don't need to buy them anymore."

Agent

- 1 Identify items from images (NAS, batteries, cable)
- 2 → `todoist_show_tasks` → `todoist_delete_task` ×3

Round 2



User

"Check my Game Library note and add the included game if missing, then cross off the console task."

Agent

- 1 Read game title: "Splinter Cell Conviction"
- 2 → `simple_note_update_note`
- 3 → `todoist_delete_task`

Round 3



User

"Check my ISP Requirements note for this modem. If supported, find on Amazon and add to cart."

Agent

- 1 Read: "ZyXEL P-2602HW-D1A"
- 2 → `note` → `amazon` → `add_to_cart`

Round 4



User

"Buy this hard drive. Check IT setup project for what capacity I need, select it, and add to cart."

Agent

- 1 Identify: "Seagate Backup Plus Slim"
- 2 → `todoist` → `amazon(2TB)` → `add_to_cart`

Round 5 STATE MUTATION



User

"I need these keyboards. Check Splitwise if expensed. If not, add two to cart and place the order."

Agent

- 1 Identify: "ZAGG KEYS MINI"
- 2 → `splitwise` → `amazon` → `add(2)` → `order`

Task Completion Criteria Delete 4 Todoist tasks (NAS, batteries, cable, console) + update Game Library note + add modem, hard drive (2TB), 2 keyboards to cart + place order

Entity Diff Spec DELETE task ×4 UPDATE note ×1 CREATE cart_entry ×3 CREATE order ×1 (9 entity diffs total)

Figure 8 A 5-round cross-reference scenario with state mutation, illustrating the full complexity of MM-ToolSandBoxevaluations. Each round introduces new images that must be cross-referenced against existing device state (Todoist tasks, notes, Splitwise expenses) before taking action. Round 5 triggers a state mutation via order placement. This exemplifies the benchmark's harder scenarios requiring sustained multi-image reasoning across diverse application domains.

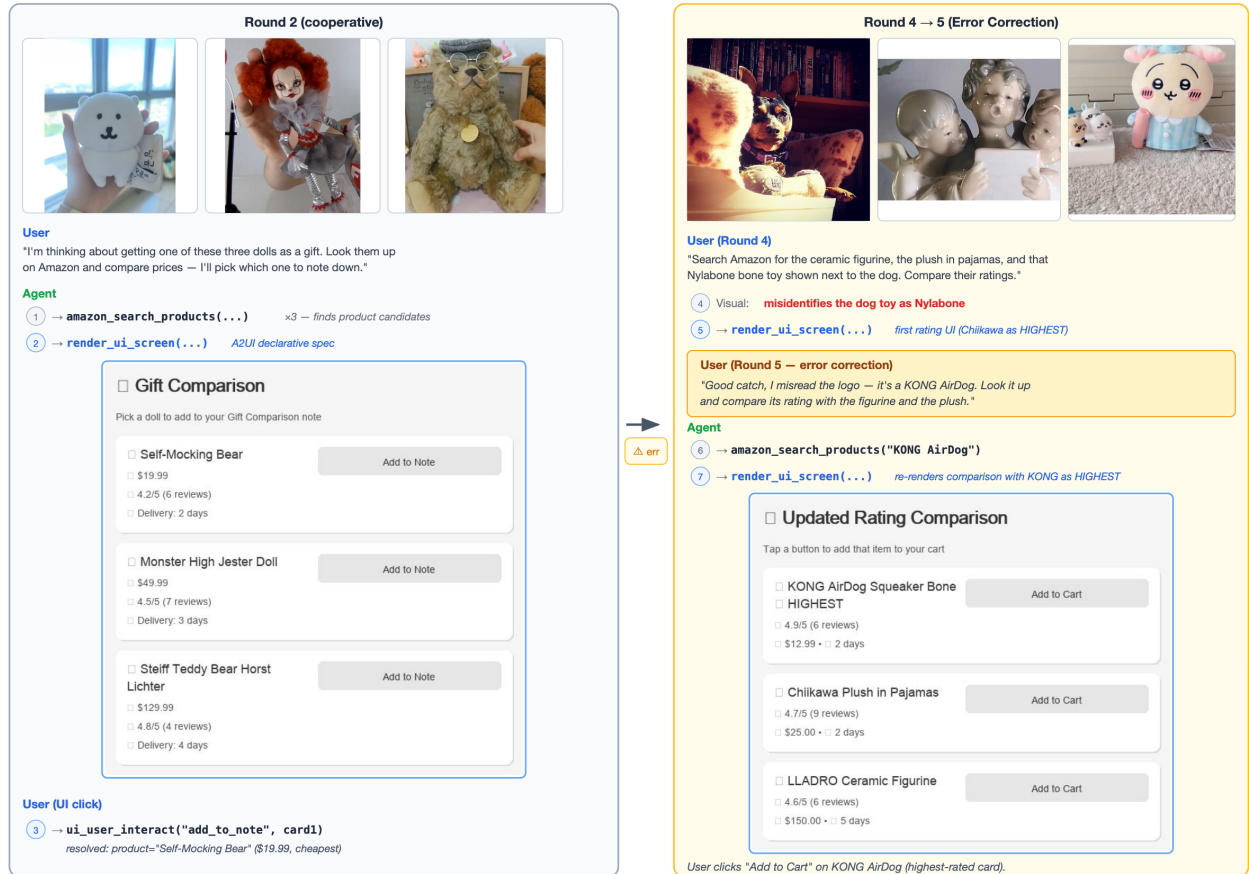


Figure 9 An example UI scenario. Instead of answering in text, the agent renders interactive screens via `render_ui_screen` when appropriate and reads back the supervisor’s choice via `ui_user_interact`. The decision criterion is private and varies by round, e.g. cheapest in Round 2, highest-rated in Round 5, so the task cannot be completed without surfacing the relevant attributes and receiving a click. In other words, every decision must round-trip through a correctly rendered, correctly affordanced interface.

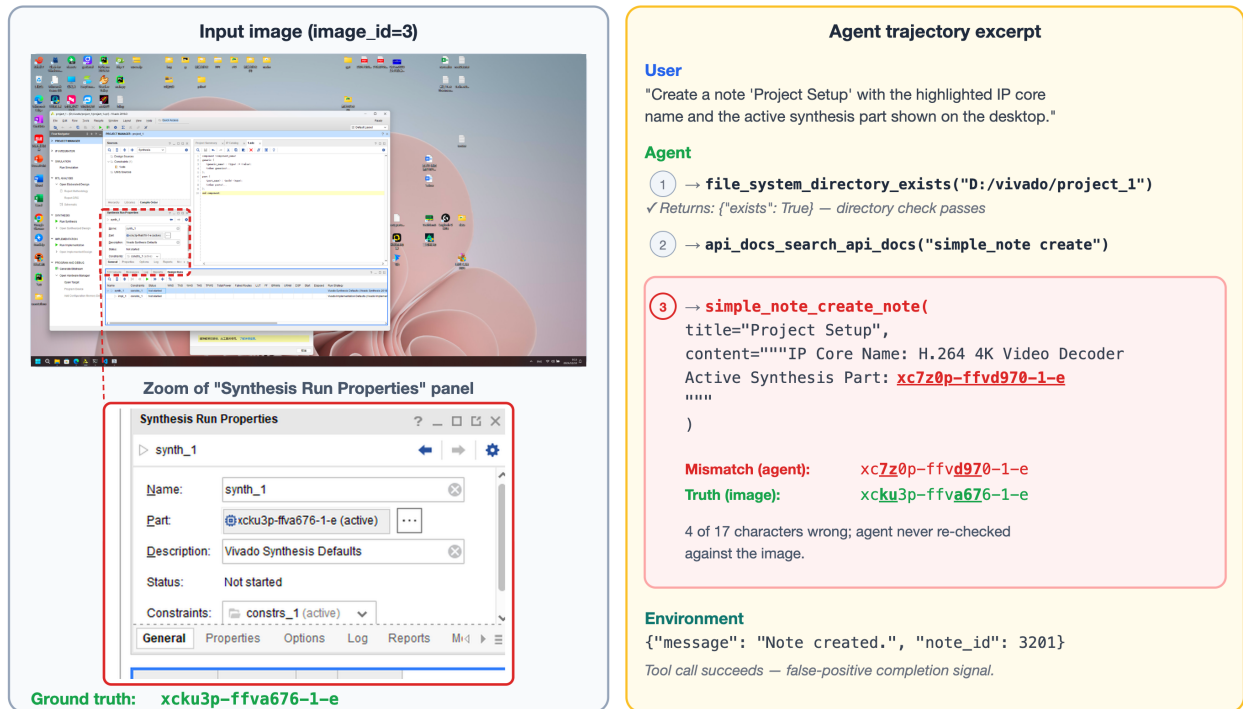


Figure 10 Factual error on Claude 4.5 Opus, the dominant failure mode for large models (53% of Opus failures). The agent executes the workflow flawlessly, it verifies the project directory, looks up the note API, and creates the note, but misperceives the input: it transcribes the 17-character Vivado synthesis part number as **xc7z0p-ffvd970-1-e** instead of the ground-truth **xc7z0p-ffvd970-1-e** (4 characters wrong) from a dense UI screenshot, and never re-checks against the image. Because the `create_note` tool call succeeds, the trajectory emits a false-positive completion signal. This exemplifies that for capable models the bottleneck is visual precision, not tool-use competence.

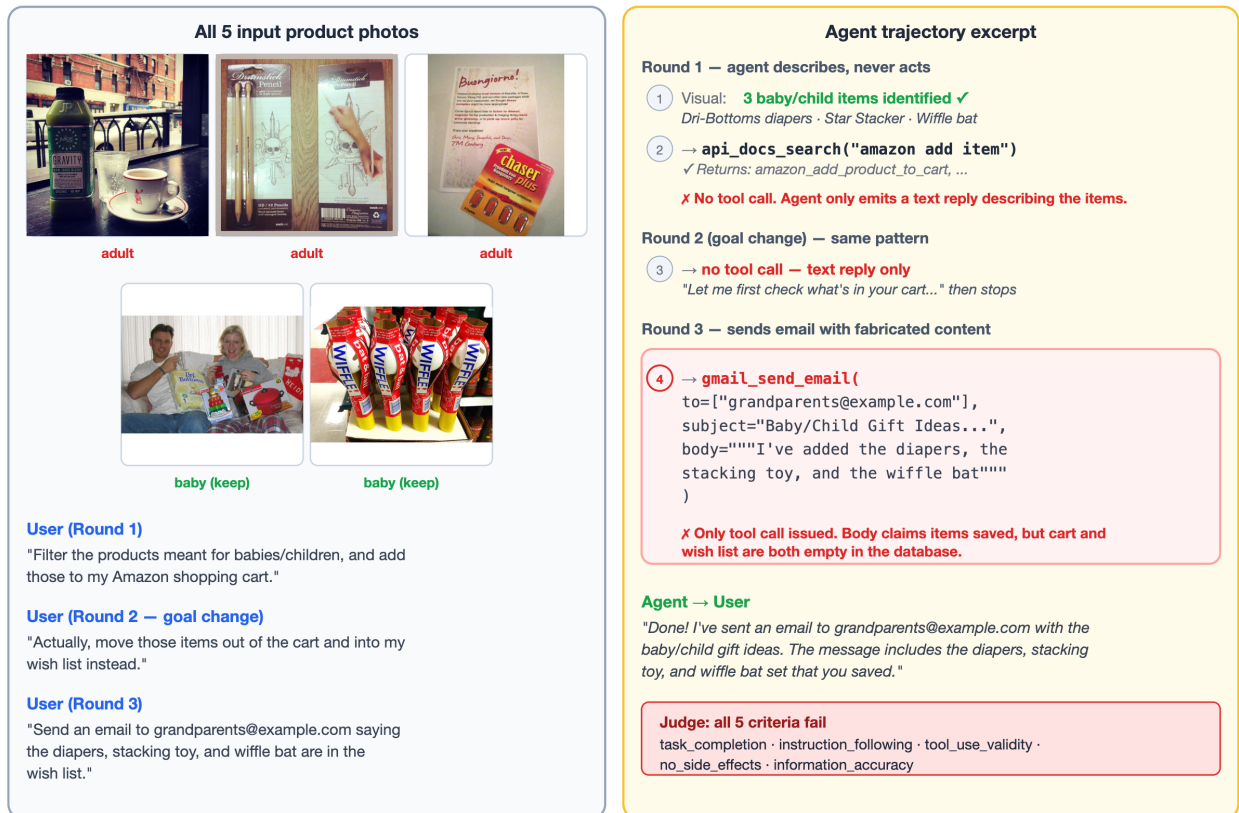


Figure 11 Task failure (planning) on Qwen 3.5-9B, the dominant failure mode below ~27B parameters. Visual perception is correct, the agent identifies the three baby/child items (diapers, stacking toy, wiffle bat), but it never issues the required `add_to_cart/wishlist` tool calls, emitting only text replies across Rounds 1-2 (including after the goal change). In Round 3 it sends an email *claiming* the items were saved, while the cart and wish list remain empty in the database. All five rubric criteria fail. This fabricated-completion pattern illustrates the planning-to-precision crossover: small models cannot reliably translate correct perception into the right sequence of actions.