STAGE-Claw: Automated State-based Agent Benchmarking for Realistic Scenarios

Anonymous ACL submission

Abstract

Large language models are increasingly used to power personal agents for everyday applications, but evaluating these agents remains a challenge. Existing benchmarks still rely on sandboxed artifacts, static task design, and coarse scoring, which hinder scalability and limit progress toward reliable personal-agent evaluation. This paper introduces STAGE-Claw, an automated framework for building and evaluating realistic personal-agent scenarios in state-based personal-computing environments. Given a task hint, STAGE-Claw automatically creates and validates a realistic benchmark task with its environment, task prompts, ground truth, and related verification programs. Agents are then evaluated in realistic operating environments, where performance is measured by the correctness of the final system state rather than only the textual response. Using STAGE-Claw, this paper creates a benchmark with 40 challenging real scenario agent tasks, evaluates 11 frontier models, and analyzes their task scores, costs, tool-call reliability, and common failure patterns. Overall, STAGE-Claw offers a scalable, state-based way to evaluate agents in realistic user scenarios.

1 Introduction

Large language models (LLMs) are increasingly being adopted as the reasoning backbones of autonomous agents (Wang et al., 2024; Xi et al., 2025; Yao et al., 2022), as exemplified by systems such as Claude Code (Anthropic, 2025) and OpenClaw (OpenClaw, 2026), which augment LLMs with tool interfaces, execution environments, memory mechanisms, and control logic. This shift changes evaluation (Liu et al., 2024; Ye et al., 2026; Li et al., 2026), agents must not only answer textual prompts but also plan over multiple steps, coordinate heterogeneous tools, and interact with environments. For example, agent integrated with email, calendars, files, browsers, and other everyday applications (OpenClaw, 2026), benchmarks must measure reliable action in persistent, cross-tool user scenarios (Li et al., 2026).

Recent agent benchmarks have moved beyond text-only evaluation, covering tool use and multi-step reasoning, web instruction following, desktop interaction, and visually grounded web tasks (Mialon et al., 2024; Deng et al., 2023; Xie et al., 2024; Zhou et al., 2024; Koh et al., 2024). However, they are still limited in three crucial aspects. **First, most existing evaluations replace real application state with sandboxed artifacts.** For example, PinchBench (Kilo AI, 2026) evaluated calendar scheduling ability through generated `.ics` files and email-related tasks through synthetic inboxes stored as workspace text files. This file-based formulation simplifies task completion evaluation, but it could ignore some operations in real scenarios, such as software-permission and tool-access errors. Consequently, it mainly evaluates artifact generation rather than an agent's ability to interact with the applications. **Second, existing evaluation tasks are typically constructed manually and are therefore difficult to scale.** GAIA (Mialon et al., 2024) and Claw-Eval (Ye et al., 2026) rely on fixed question-answer instances or human-verified tasks and rubrics. However, personal agents must adapt to diverse user preferences, goals, workflows, and evolving contexts, requiring evaluation tasks that scale across personalized and dynamic scenarios. Such coverage is difficult and costly to achieve with manually curated static benchmarks. **Third, existing evaluations often lack process-aware diagnostics.** Only final results scoring or checkers (Ma et al., 2024; Trivedi et al., 2024) may fail to diagnose where errors occur within the completion workflow. For instance, an incorrect conference-tracking related calendar operation produced by an agent may result from errors in intermediate steps, such as time-zone conversion, conflict resolution,

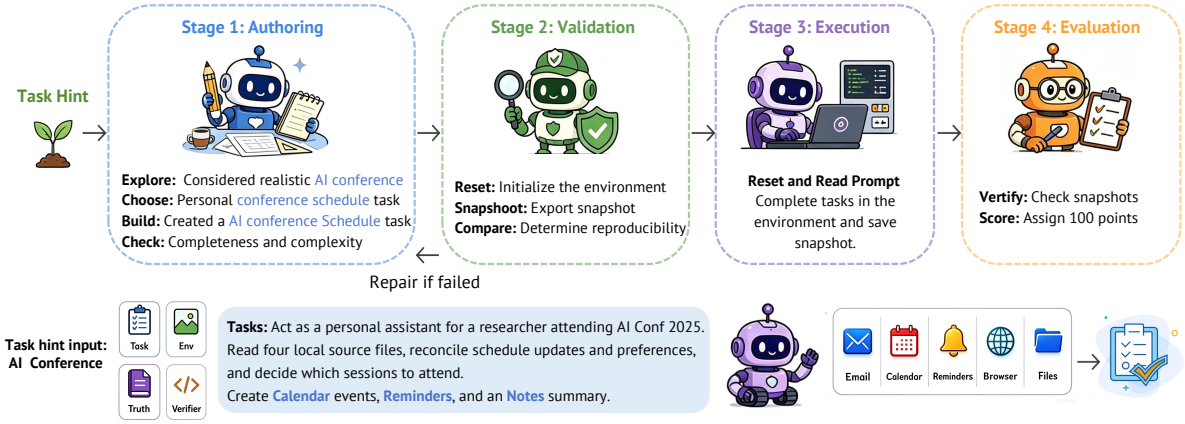

Figure 1: The workflow of STAGE-Claw. *1) Benchmark authoring*: Explore a task hint and generate the task. *2) Benchmark validation*: Check task correctness, difficulty, and reproducibility, revising if needed. *3) Agent execution*: Target agent attempts the task in the environment. *4) State-based evaluation*: Score results by verifying system state.

or reconciling inconsistent message sources, which final-result-only checkers cannot localize. Overall, these limitations motivate a state-based and scalable evaluation paradigm that assesses agents in realistic scenarios.

To address these limitations, this paper proposes ***STAGE-Claw*** (***S***tate-based, ***T***ool-integrated, ***A***gent task ***G***eneration and ***E***valuation), an automated framework for constructing and evaluating agent benchmarks in realistic environments. STAGE-Claw addresses these challenges from three aspects. **State-based evaluation** verifies whether an agent's actions produce the expected state changes in the environment, rather than merely checking outputs or artifacts. **Automated construction** and **Process-aware diagnosis** refer to evaluating agents by automatically generating and validating benchmark instances and analyzing fine-grained metrics to localize task failure reasons. As shown in Figure 1, STAGE-Claw automatically creates task instances from the task hint word, validates their verifiability, difficulty, and reproducibility, executes target agents in the generated environments, and evaluates their performance by verifying persistent system-state changes across tools. This design shifts the evaluation from final-artifact checking to state-based assessment of agent behavior in realistic scenarios. We build 40 challenging tasks and conduct a detailed analysis of 11 frontier models' test results. Overall, our contributions are summarized as follows:

- This paper proposes STAGE-Claw, a framework that systematically automates the construction and validation of state-based agent evaluation instances in realistic scenarios.
- Using STAGE-Claw, we build a state-based benchmark of 40 challenging tasks grounded in 5-group realistic scenarios, covering workflows that require cross-source reasoning, tool state updates, and cross-tool consistency.
- We evaluate 11 frontier models on STAGE-Claw and conduct a detailed analysis of the task trajectory and results, which contribute insights for developing reliable, state-based, and extensible agent evaluation systems.

2 STAGE-Claw

In this section, we introduce **STAGE-Claw**, a four-stage automated framework for constructing and evaluating state-based agent benchmarks.

Formalized Definition. Each benchmark instance is formulated as a state-transformation problem over a reconstructable real environment:

$$\mathcal{B} = (q, E_0, G, R, V),$$

where q is a task prompt, E_0 is an initial tool environment, G specifies a target final state, R is a scoring rubric, and V is an executable verifier. An agent observes only (q, E_0) at the beginning, while (G, R, V) is reserved for evaluation to prevent information leakage. Given an agent policy π, execution from the initial state $\boldsymbol{s}_0$ produces a trajectory

$$\tau_\pi = (\boldsymbol{s}_0, o_0, a_0, \ldots, a_{H-1}, o_H, \boldsymbol{s}_H),$$

where o_t is the environment observation, a_t is the agent's action for tool use, and $\boldsymbol{s}_H$ is the final environment state. The agent succeeds not by merely

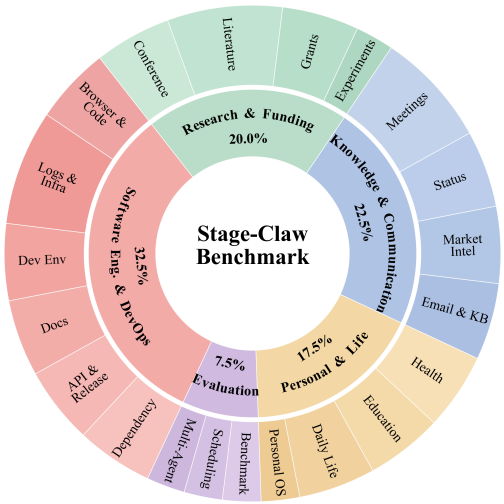

Figure 2: Statistics of task categories.

producing a final textual answer, but by transforming E_0 into a final state G where a V is utilized to evaluate its correctness.

Implementation Environment. STAGE-Claw evaluates agents in real computing environments, where user requests often span multiple applications. The environment can be regarded as a collection of all the states of the tools. Initial environment $E_0 = (\mathcal{T}, \boldsymbol{s}_0)$, where $\mathcal{T}$ denotes the set of available tools and $\boldsymbol{s}_0$ denotes their initial joint state. The global environment state at step t is represented as $\boldsymbol{s}_t = \{s_t^\tau\}_{\tau \in \mathcal{T}}$, where s_t^τ is the state of tool τ. Agents receive user-level access to realistic tools, including the file system, browser, terminal, calendar, email, reminders, and notes, which support read, write, and execution operations over persistent application states. Tasks are modeled as transitions from the initial state $\boldsymbol{s}_0$ to an acceptable final-state set $\mathcal{S}_G$ specified by a target G, and are considered successful when $\boldsymbol{s}_H \in \mathcal{S}_G$, indicating that the agent has produced the required persistent changes while preserving relevant existing state.

2.1 Stage 1: Benchmark Authoring

The first stage automatically constructs executable benchmark instances from task hint words. We manually curate 40 real assistant scenarios as task hint words. For each task hint, STAGE-Claw invokes a benchmark-authoring agent to instantiate an executable task. The authoring agent acts only as a benchmark designer. Explore realistic user needs and imagine several challenging scenarios, then select a scenario with sufficient complexity. Based on the selected scenario, the authoring agent constructs a task instance containing a task prompt, an environment-construction guide, the corresponding ground truth, and an executable verification program aligned with the ground truth.

To ensure sufficient complexity, each task is designed to be multi-step, involve multiple tools or information sources, be reconstructable from a clean state, and support objective state-based evaluation. We also deliberately add some operators with different difficulty types (illustrated in Table 5 in Appendix A.3) to model the complexity of real-world scenario tasks. In general, we build 40 tasks into 5 groups. Figure 2 shows the task distribution.

2.2 Stage 2: Benchmark Validation

Before admission into the evaluation set, each task instance is checked by an independent validation agent that does not solve or modify the task. The checker verifies four properties. *Structure*, ensuring that each task includes environment construct guidance, visible task prompt, hidden ground truth, and the corresponding executable verify program. *Reproducibility*, rebuilding the environment twice from a clean state and comparing snapshots across tool states such as files, calendars, reminders, notes, and email. Ensure that the task environment can be consistently reconfigured to the same initial state each time. *Verifiability*, assessing whether the scoring is objective and executable. *Difficulty calibration*, checking whether the difficulty level of the task includes the following types of difficulty. Such as cross-source conflicts, hidden dependencies, noisy data, entity alignment, tool state updates, and cross-tool consistency.

Each dimension receives a status of **PASS**, **FAIL**, **BLOCKED**, or **WARN**. These statuses are aggregated into a weighted 100-point checker score. Instances scoring above a threshold (80 in our experiments) are accepted. Failed instances are returned to *Stage 1* with diagnostics for targeted repair and re-submitted until they pass or reach the maximum number of repair attempts.

2.3 Stage 3: Agent Execution

This stage involves testing the qualified tasks. For each validated benchmark, STAGE-Claw resets the initial state, reconstructs the environment from the environment-construction guide, and creates an isolated execution workspace containing only the visible task prompt, related documents, and tools. The evaluated agent must complete the task within a predefined time budget using only the available

Benchmark	State-based	Multi-tool	Auditable	Auto Construction	Perturbation
AgentBench (Liu et al., 2024)	○	○	○	✗	✗
GAIA (Mialon et al., 2024)	✗	○	✗	✗	✗
τ-bench (Yao et al., 2024)	✓	✓	✓	✗	✗
WebArena (Zhou et al., 2024)	✓	○	✓	○	✗
OSWorld (Xie et al., 2024)	✓	✓	✓	○	✗
ToolBench (Qin et al., 2024)	✗	✓	○	✓	✗
Terminal-Bench (Merrill et al., 2026)	✓	○	✓	✗	✗
PinchBench (Kilo AI, 2026)	○	○	○	✗	✗
Claw-Eval (Ye et al., 2026)	✓	✓	✓	○	✓
STAGE-Claw (Ours)	✓	✓	✓	✓	✓

Table 1: Comparison of agent evaluation benchmarks along STAGE-Claw. **State-based** checks explicit environment states; **Multi-tool** requires coordinated tool use; **Auditable** supports traces, snapshots, rubrics checkers; **Auto Construction** enables automatic or programmatic task construction; and **Perturbation** introduces controlled noise, conflicts, or errors. Green checks, yellow circles, and red crosses denote full, partial, and no core support.

instructions and tools. During execution, it may interpret requirements, interact with tools and update tool states, inspect intermediate results, and produce final outputs. Once the agent finishes, STAGE-Claw records execution metadata, including run status, elapsed time, and tool-interaction traces. It also captures a snapshot of the final environment. These recorded states are sent to the evaluation stage, where the evaluator assesses whether the agent's outputs and final-state record satisfy the task requirements.

2.4 Stage 4: State-Based Evaluation

The final stage performs a state-based evaluation after the evaluated agent completes the task execution. After execution, STAGE-Claw runs the task-specific verifier to check whether the recorded states align with the hidden ground truth G. The evaluator checks file output, tool-state updates, and formatting constraints, then returns a structured evaluation report. LLM-assisted adjudication is used only as a guarded fallback. It is triggered when executable verification fails to execute, times out, or cannot access a required snapshot field because of evaluator-side tool instability. All evaluation paths enforce a unified total-score format for automatic parsing and aggregation. Finally, STAGE-Claw records the agent score, execution status, and evaluation metadata and links them with execution logs to support detailed analysis.

Overall, we constructed 40 challenging tasks using STAGE-Claw. As summarized in Table 1, STAGE-Claw combines state-based evaluation, multi-tool workflows, auditable verification, automated construction, and controlled perturbation in one workflow. This design makes STAGE-Claw a scalable state-based evaluation framework rather than a collection of manually written agent tasks.

3 Evaluation

3.1 Task Construction

Construction Settings. Benchmark authoring agent is instantiated with Claude-Sonnet-4.6 (Anthropic, 2026b). The full prompts are shown in Appendix C. Each candidate task is checked by the checker agent (test results exceeded 80 points), which requires an average of 2.67 repair iterations. All accepted tasks can be verified by executable verification programs without the need for LLM-assisted adjudication. Finally, human annotators check each accepted task for scenario realism, task completeness, instruction clarity, ground-truth correctness, and evaluator-rubric alignment. The annotation details are provided in Appendix A.7. All task construction, repair, and checker-based validation are fully automated, human involvement is only used for quality review.

Construction Budget and Scope. Each STAGE-Claw task requires automated construction, verification, evaluation, and human audit. Building an accepted task costs about 35 to 40$ and takes 1-2 hours, excluding model-evaluation runs. The time cost to each task for each model is 3 to 15 minutes, and the API cost is 0.17 to 6.55$. Considering the substantial time and API costs of constructing and evaluating tasks, we build a set of 40 challenging tasks and position it as a high-quality pilot benchmark rather than an exhaustive coverage of all personal-agent scenarios. We evaluate each model on each task once in our experiments and include a limited repeated-run diagnostic in Ap-

Model	Score	First-Pass	Time (s)	Tokens (M)	Cost ($)	Tool Calls
Claude-Opus-4.7	**77.1**	**80.0%**	422.8	1.242	$6.55	40.6
Claude-Sonnet-4.6	69.43	65.0%	733.8	1.265	$3.98	42.2
GPT-5.5	69.19	65.0%	316.4	0.405	$1.33	28.9
GPT-5.4	65.37	52.5%	**192.5**	**0.236**	$0.69	**21.8**
Gemini-3.1-Pro	65.5	67.5%	742.4	0.642	$2.65	52.6
DeepSeek-V4-Pro	59.78	60.0%	744.4	0.620	$0.28	28.3
Doubao-Seed-2.0-Pro	61.07	50.0%	325.6	0.531	$0.27	28.4
GLM-5	67.40	60.0%	423.7	0.694	$0.73	40.5
Kimi-K2.6	59.05	55.0%	641.3	0.921	$0.93	54.9
MiniMax-M2.7	47.53	35.0%	937.5	0.529	**$0.17**	34.8
Qwen3.5-Plus	59.74	57.5%	406.2	0.882	$0.30	59.3

Table 2: Performance and efficiency comparison of large language models on STAGE-Claw. Score is the average valid run task score. First-Pass denotes the proportion of tasks scoring above 60 in the first round. Time, Tokens, Cost, and Tool Calls report average run time, token usage, API cost, and tool invocations of each task. Bold values indicate the best result for each metric, and underlined values indicate the second-best result.

pendix A.6. Our goal is to demonstrate the feasibility and diagnostic value of automated state-based evaluation in realistic environments.

3.2 Evaluated Models

We evaluated 11 state-of-the-art models spanning multiple model families: Claude-Opus-4.7 (Anthropic, 2026a), Claude-Sonnet-4.6 (Anthropic, 2026b), DeepSeek-V4-Pro (DeepSeek, 2026), Qwen3.5-Plus (Alibaba Cloud, 2026), GPT-5.5 (OpenAI, 2026a), GPT-5.4 (OpenAI, 2026b), Gemini-3.1-pro-preview (Google, 2026), Doubao-Seed-2.0-Pro (ByteDance Seed, 2026), GLM-5 (GLM-5-Team et al., 2026), Kimi-k2.6 (Moonshot AI, 2026), and MiniMax-M2.7 (MiniMax, 2026). All models were assessed in the 40 benchmark tasks constructed for this study.

3.3 Evaluation Settings

All models were tested within the framework of OpenClaw, with reasoning disabled and temperature set to 0 when configurable (otherwise using the provider-fixed temperature). This setting minimizes sampling variability and improves reproducibility across runs. Each benchmark task was executed in an isolated operating-system environment, and a freshly initialized OpenClaw agent was instantiated for each run to avoid interference from persistent memory, cached state, or prior context.

Scores in Table 2 show each model's first valid run per task. A valid run requires a correctly reconstructed environment and a completed evaluation. Given that tasks involve interactions with real tools, including calendar events, emails, and reminders, executions were performed serially to avoid conflicts between concurrent tasks. This setup ensures that each trial reflects the agent's ability to complete the task based solely on the visible prompt and accessible tool interfaces, providing a controlled and reproducible evaluation environment.

3.4 Main Results

Table 2 reports the performance and efficiency of different LLMs in STAGE-Claw under the OpenClaw framework. Overall, the results reveal a clear trade-off between task performance and execution efficiency.

Model performance and execution efficiency exhibit a clear trade-off. Claude-Opus-4.7 achieves the strongest overall performance, with the highest average score of 77.1 and the highest First-Pass rate of 80.0%, demonstrating the highest reliability among all evaluated models. Claude-Sonnet-4.6 obtains the second-best average score, while Gemini-3.1-Pro achieves the second-best First-Pass rate. However, these strong-performing models generally require higher computational overhead, such as longer execution time, larger token consumption, or higher API cost. In contrast, GPT-5.4 shows the best efficiency profile, achieving the lowest latency, the lowest token usage, and the fewest tool calls while maintaining a competitive score. GPT-5.5 further provides a strong balance between performance and efficiency, approaching Claude-Sonnet-4.6 in score while using substantially fewer resources.

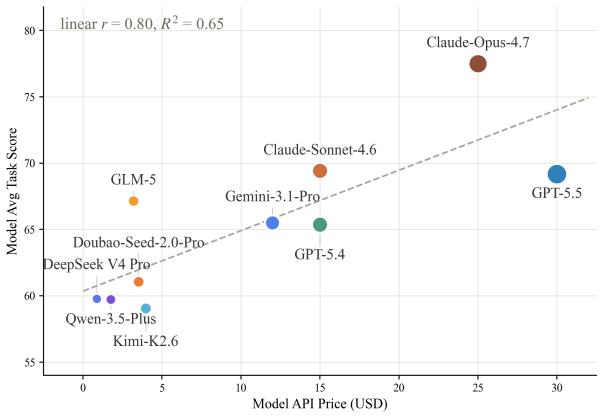

Figure 3: Performance comparison between the unit model api price, and the task score. The x-axis represents the unit price of the API, and the y-axis represents the average model task score.

Tool-use frequency does not directly determine task performance. Models such as Qwen3.5-Plus (59.3) and Kimi-K2.6 invoke tools (54.9) far more frequently than GPT-5.4 (21.8), yet obtain lower average scores. Conversely, GPT-5.4 completes tasks with the fewest tool calls while outperforming several models with heavier tool usage. This suggests that effective tool use depends more on planning quality, tool selection, and result verification than on the raw number of tool invocations. Excessive tool calls may therefore indicate inefficient coordination rather than stronger task-solving ability.

Higher API unit price is positively correlated with stronger task performance, but does not fully explain model quality. As shown in Figure 3, model average task score exhibits a strong positive linear correlation with model API price. This suggests that more expensive models tend to achieve better task performance on STAGE-Claw. Claude-Opus-4.7 is a representative example, combining a high API price with the highest average score. However, the correlation is not absolute. GLM-5 achieves a relatively high score at a low API price, while GPT-5.5 is more expensive but does not outperform Claude-Opus-4.7 or Claude-Sonnet-4.6. We speculate that GPT-5.5's relative underperformance may partly stem from its configuration with reasoning (thinking) modules turned off, which prioritizes execution efficiency over deliberative planning and multi-step reasoning. These observations indicate that price is an important but insufficient proxy for agent capability and that cost-effectiveness varies substantially across models.

These findings show that STAGE-Claw evaluates not only final task success, but also how reliably and efficiently models operate in realistic scenarios. The results highlight the need to jointly consider task performance, latency, realized run cost, token usage, API unit price, and tool-use behavior when assessing LLM agents.

4 Analysis

4.1 Failure Analysis

We further analyze non-passing runs to characterize common failure patterns. Since a single run may exhibit multiple errors, our analysis is multi-label rather than mutually exclusive. Figure 4.a summarizes the resulting distribution.

The most prevalent failure mode is ***Tool Failure***, appearing in 95.4% of non-passing runs. These failures involve missing, incomplete, or incorrectly routed writes to real tools such as calendar, notes, reminders, or email, indicating that agents often generate plausible intermediate outputs but fail to correctly update the underlying environment. The second most frequent category is ***Invalid Format*** (75.0%), including incorrect JSON format, missing required fields, incorrect filenames, or verification reports without required metadata. ***Trap and Reconciliation Failures*** (69.7%) and ***Verification Failures*** (59.2%) further show that agents struggle with source-of-truth selection, entity alignment, timezone and deadline reasoning, deduplication, noise filtering, and evidence-backed validation. ***Information Coverage Errors*** (52.6%), ***Incomplete Execution*** (49.3%), and ***Spurious Outputs*** (44.1%) are also common, suggesting that current agents have difficulty integrating reliable extraction, verification, and conservative state updates into a coherent workflow. Figure 4.b shows the percentages in model of failed outcomes.

4.2 Tool-Use Analysis

In this section, we analyze the frequency of tool calls and the errors that occur in the frequent tool calls. Figure 4.c reports both the tool call usage distribution and the failure cause composition, where failure causes are normalized over failed tool calls within each category. Among all categorized tool calls, Shell/Session tools are used most frequently, accounting for 79.8% of calls, followed by File I/O tools at 13.4%. State/Messaging tools are invoked much less often, representing only 6.7% of calls.

Despite their different usage frequencies, the failure patterns vary substantially across tool cat-

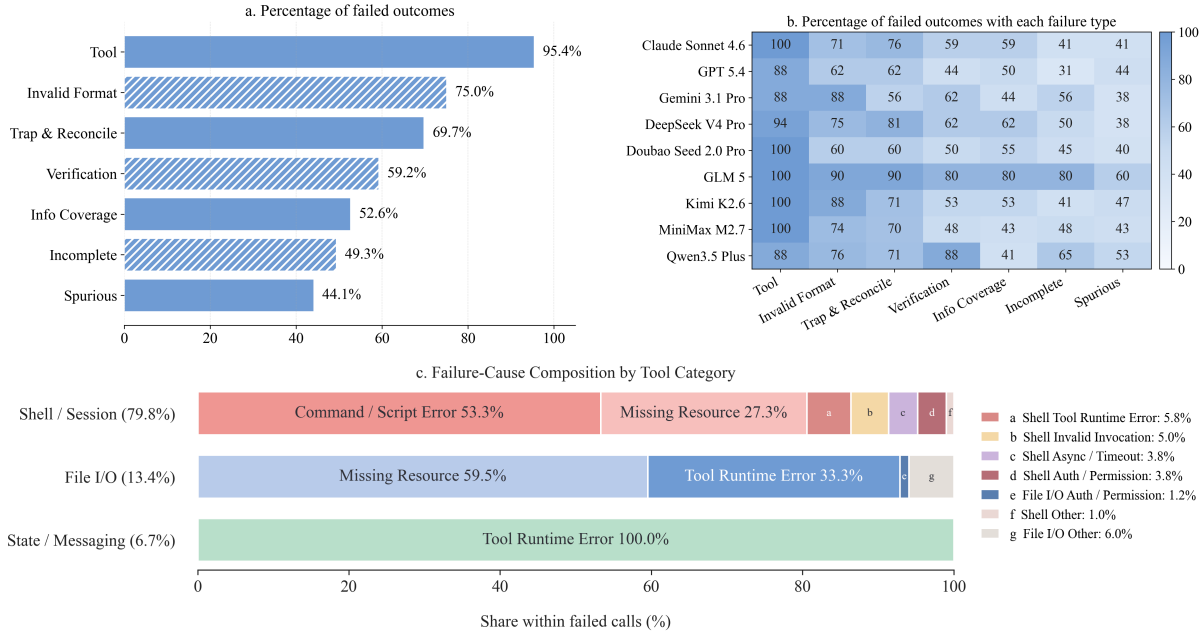

Figure 4: Failure-type analysis over non-passing outcomes. Categories are multi-label, so percentages do not sum to 100. ***a.*** prevalence of each failure type among failed outcomes. ***b.*** model-wise percentage of failed outcomes containing each type. ***c.*** proportion of tool usage and the error types for the corresponding tools

egories. Shell/Session failures are dominated by command or script errors (53.3%), followed by missing resources or paths (27.3%). This suggests that shell-based interaction primarily fails due to incorrect command construction, missing dependencies, or invalid assumptions about the runtime environment. File I/O failures exhibit a distinct pattern: missing resources or paths account for 59.5% of failed calls, while tool runtime errors account for 33.3%. This indicates that accurate resource localization and path validation are critical for reliable file operations. By contrast, all observed State/Messaging failures are attributed to tool runtime errors, suggesting that operations involving persistent application state or external communication channels remain particularly brittle.

Overall, tool failures are category-specific rather than driven by a single generic error mode. Shell/Session tools require more robust command generation and dependency checking. File I/O tools depend heavily on reliable resource validation, and State/Messaging tools require stronger recovery mechanisms for persistent-state updates.

4.3 Effectiveness of State-based Evaluation

To examine whether state-based evaluation can better capture unexpected failures in realistic scenarios, we construct a virtual-state variant of STAGE-Claw. In this setting, each task is modified to replace actual tool-state changes with textual outputs that simulate the expected state updates. We run this virtual setting on the same tasks using two representative models, DeepSeek-V4-Pro and Qwen3.5-Plus, and compare the results with the original state-based evaluation.

Model	State-based	Virtual	Δ
DeepSeek-V4-Pro	59.78	66.71	+6.93
Qwen3.5-Plus	59.74	64.57	+4.83
Cause category	#Tasks	Proportion	Avg. Δ
Execution failure	7	8.75%	+76.57
Real-state gap	13	16.25%	+14.05
Other errors	3	3.75%	-2.67

Table 3: The upper section shows a comparison between state-based and virtual evaluation, Δ denotes Virtual - State-based. The lower section breaks down the causes of these gaps.

As shown in Table 3, the virtual setting yields higher scores than the state-based setting for both models. However, this gap does not reflect better task completion. Instead, it suggests that output-only evaluation can overestimate agent performance by ignoring whether the agent actually modifies persistent tool states. In real tool-integrated environments, agents may fail due to execution failure and real-state gap, such as invalid state writes, missing generated artifacts, and tool-side effects. These failures are naturally exposed by state-based evaluation, but can be hidden when the task is reduced to textual state simulation. Therefore, the

Task	Base	Noise	Misleading	Conflict
AI Conference Tracker	77.0	$23.0_{-54.0}$	$23.0_{-54.0}$	$23.0_{-54.0}$
Benchmark Administrator	85.0	$56.0_{-29.0}$	$35.0_{-50.0}$	$49.0_{-36.0}$
Competitor Radar	62.0	$57.0_{-5.0}$	$55.0_{-7.0}$	$55.0_{-7.0}$
Email Message Triage Center	45.0	$40.0_{-5.0}$	$22.0_{-23.0}$	$22.0_{-23.0}$
Average	**67.3**	$\mathbf{44.0_{-23.3}}$	$\mathbf{33.8_{-33.5}}$	$\mathbf{37.3_{-30.0}}$

Table 4: Qwen3.5-Plus scores under manually constructed memory perturbations on four STAGE-Claw tasks. Subscripts indicate score drops relative to the Base setting.

comparison indicates that the state-based evaluation provides a more faithful assessment of agent capability in realistic tool-use scenarios.

4.4 Memory Perturbation Analysis

Persistent memory can also affect state-based agent execution, especially when it contains irrelevant, stale, or conflicting information. We therefore conduct a small diagnostic study on four STAGE-Claw tasks using Qwen3.5-Plus. For each task, we move key task-specific requirements from the visible instruction into the agent's persistent memory and lightly revise the task prompt so that the task remains natural and memory-dependent. We then manually construct three perturbation types: *Noise*, which adds irrelevant memory entries around the key facts, *Misleading*, which injects plausible but incorrect rules or values and *Conflict*, which introduces inconsistent versions of the same key facts.

Table 4 reports the resulting scores. All three perturbation types reduce performance relative to the base memory-dependent setting, with *Misleading* and *Conflict* causing larger drops in our sample. These results are intended as diagnostic evidence rather than a comprehensive benchmark-wide conclusion. They suggest that memory quality can influence both agent decisions and downstream state updates, motivating future evaluation protocols and agent mechanisms that track memory provenance, recency, and conflicts in addition to tool-execution reliability.

5 Related Work

Agent Benchmarks for Tool Use and Interactive Environments. Recent agent benchmarks have expanded LLM evaluation beyond isolated text generation toward tool use, multi-step reasoning, and interactive task completion. General-purpose benchmarks such as AgentBench (Liu et al., 2024) and GAIA (Mialon et al., 2024) evaluate planning, reasoning, web access, and external tool use, while function-calling benchmarks such as BFCL (Patil et al., 2025) focus on whether models can correctly select, parameterize, and invoke tools. Another line of work evaluates agents in interactive environments. Mind2Web (Deng et al., 2023), WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024), and WorkArena (Drouin et al., 2024) study web and enterprise workflows, and OSWorld (Xie et al., 2024) evaluates agents in real operating-system and desktop applications. AppWorld (Trivedi et al., 2024) and τ-bench (Yao et al., 2024) further emphasize controllable app or tool-agent-user interactions. These benchmarks provide progress in realistic interaction, but they are primarily designed around fixed tasks, simulated APIs, GUI control, or transactional tool use.

Personal-Agent Benchmarks and State-based Evaluation. Recent benchmarks have moved closer to OpenClaw-oriented personal agents, where LLMs interact with tools, memory, execution environments, and control logic (OpenClaw, 2026; Li et al., 2026). PinchBench (Kilo AI, 2026), WildClawBench (WildClawBench, 2026), ClawsBench (Li et al., 2026), and Claw-Eval (Ye et al., 2026) evaluate practical workflows such as scheduling, email, research, file management, and multi-turn tool use. Many existing agent benchmarks rely on manually curated tasks, sandboxed artifacts, simulated workspaces, or coarse rubrics, limiting their scalability and ability to diagnose persistent-state errors. STAGE-Claw addresses these limitations by automatically generating benchmark instances from task hint words, instantiating them in real-application environments, and evaluating agents via snapshots of the final system state.

Conclusion

This paper introduces STAGE-Claw, an automated state-based framework for constructing and evaluating realistic scenario benchmarks. By assessing state-based snapshots rather than textual responses, STAGE-Claw enables more faithful evaluation of agent ability in multi-tool workflows. In addition, the automated pipeline provides a practical path toward scaling realistic scenario agent benchmarks while preserving reproducibility and auditable evaluation. Experiments on 40 tasks show that current agents still struggle to complete complex tasks that involve multiple tool invocations in real scenarios. These results highlight the need for state-based evaluation and provide practical insights for building more reliable assistant agents.

Limitations

STAGE-Claw is designed for realistic state-based evaluation, but this design also brings limitations. First, the benchmark currently contains 40 accepted tasks, and the main evaluation follows a first-valid-run protocol with only limited repeated-run diagnostics. Thus, the results should be viewed as a controlled snapshot rather than a statistically exhaustive leaderboard. Second, constructing high-quality state-based tasks remains resource-intensive. Although task authoring, repair, and checker-based validation are automated after human-provided task hints, each accepted task still requires realistic scenario design, deterministic environment setup, executable verifier implementation, and quality checking, which together incur nontrivial time and API costs. Third, evaluation in real personal-computing environments means that scores may reflect not only model capability but also OpenClaw, tool wrappers, OS configuration, permissions, and runtime stability. This improves realism but may introduce platform-dependent failures.

References

Alibaba Cloud. 2026. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5.

Anthropic. 2025. Claude code overview. https://code.claude.com/docs/en/overview.

Anthropic. 2026a. Introducing Claude Opus 4.7. https://www.anthropic.com/news/claude-opus-4-7.

Anthropic. 2026b. Introducing Claude Sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6.

ByteDance Seed. 2026. Seed2.0. https://seed.bytedance.com/en/seed2.

DeepSeek. 2026. DeepSeek V4 Pro. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, and 1 others. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, and 1 others. 2026. GLM-5: From Vibe Coding to Agentic Engineering. *arXiv preprint arXiv:2602.15763*.

Google. 2026. Gemini 3.1 Pro Preview. https://deepmind.google/models/gemini/pro/.

Kilo AI. 2026. Pinchbench: Real-world benchmarks for ai coding agents. https://pinchbench.com/.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, and 1 others. 2026. Clawsbench: Evaluating capability and safety of llm productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2024. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pages 52989–53046.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

Mike A Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E Kelly Buchanan, and 1 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pages 9025–9049.

MiniMax. 2026. MiniMax M2.7: Early Echoes of Self-Evolution. https://www.minimax.io/news/minimax-m27-en.

Moonshot AI. 2026. Kimi K2.6: Advancing Open-Source Coding. https://www.kimi.com/blog/kimi-k2-6.

OpenAI. 2026a. GPT-5.5. https://openai.com/zh-Hans-CN/index/introducing-gpt-5-5/.

OpenAI. 2026b. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/.

OpenClaw. 2026. Openclaw: Personal ai assistant. https://openclaw.ai/.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. 2025. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, volume 2024, pages 9695–9717.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. 2024. 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.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

WildClawBench. 2026. Wildclawbench. https://github.com/InternLM/WildClawBench.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, and 1 others. 2026. Claw-eval: Toward trustworthy evaluation of autonomous agents. *arXiv preprint arXiv:2604.06132*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Appendix

A.1 Evaluation Separation and Blindness

To preserve evaluation validity, the benchmark construction protocol enforces a strict separation between the agent-facing task prompt and the evaluator-only materials. The agent receives only the question prompt and the initialized environment specified by the environment-construction document. In contrast, the evaluator has access to ground truth, which may contain gold answers, scoring scripts, structured checklists, or other verification procedures.

The agent-facing prompt must not include any of the following: gold labels, expected final answers, hidden constraints, scoring rubrics, evaluator scripts, checksum values, or explicit references to the verification logic. This constraint ensures that task success reflects the agent's ability to operate in the environment rather than its ability to exploit leaked evaluation information.

A.2 Design Criteria

Each task produced under this protocol is required to satisfy the following criteria.

1. **Multi-step execution.** The task must require at least three distinct operations, such as locating files, extracting information, transforming data, writing an output file, or interacting with an application.

2. **Cross-tool grounding.** The task must require at least two different tools or information sources. Examples include filesystem access and terminal commands, browser interaction and document parsing, or local data inspection and application state modification.

3. **Reproducibility.** The environment construction procedure must produce the same initial state on every run. All seeded files, directories,

and system configurations should be specified explicitly.

4. **Blind task execution.** The task prompt shown to the agent must not contain oracle information. The agent should only know the user-facing objective, the relevant working directory, and operational constraints.

5. **Objective gradability.** The final result must be verifiable without subjective human judgment. Acceptable evaluation methods include deterministic scripts, exact-match checks, structured rubrics with binary criteria, or reproducible checklist-based validation.

6. **Time independence.** The task must not depend on real clocks, real-time APIs, or mutable external states unless those states are explicitly mocked or seeded during environment construction.

A.3 Difficulty Categories

In order to simulate the difficulty and complexity of the task in a real situation, we have carried out complex processing on the task. The specific types of processing are shown in Table 5.

Category	Difficulty Mechanism
Data Inconsistency	Cross-source conflicts, Heterogeneous timestamp formats, Inconsistent status or Field values
Dependency Reasoning	Hidden dependencies, Topological ordering, Prerequisite Implementation requirements
Noise Filtering	Test records, Drafts, Deprecated items, Dead URLs, Empty or invalid entries
Entity Format	Aliases, Email mappings, Abbreviations, Canonical IDs, Category mapping
Output Precision	Strict field schemas, Deterministic sorting, Casing constraints, Date formats, Exact-match error messages

Table 5: Difficulty categories and included mechanisms

A.4 Snapshot Tools

A.5 Self-Review Checklist

Before a benchmark task is finalized, the author performs a self-review using the following checklist.

1. The environment builder can follow environment make documents without requiring additional clarification.

2. The setup procedure is deterministic and reproducible.

3. The agent-facing question prompt contains no ground-truth information.

4. The evaluator can score the task using only the final state and the files in ground truth documents.

5. The success condition is singular, explicit, and unambiguous.

6. Potential shortcuts or cheating strategies have been identified and blocked.

7. The task admits a plausible harder variant for future benchmark scaling.

A.6 Repeated-run diagnostic.

The main results in Table 2 are based on the first valid execution of each model-task pair. To examine run-to-run variability without incurring the full time and API cost of repeating all 11 models on all tasks, we conducted a limited three-run diagnostic on three models: Qwen3.5-Plus, DeepSeek-V4-Pro, and GLM-5. The pass@3 results of these three models are shown in Figure 5. This diagnostic analysis is intended to illustrate score variability under repeated executions and is not used to compute the aggregate results or model ranking in Table 2.

A.7 Human Audit and Filtering

We conduct human annotation to validate the quality of each constructed benchmark task before inclusion in the final benchmark. We hired three task annotators. Annotators review the complete task package, including the environment construction guide, the agent-facing prompt, and the evaluator-only ground-truth materials. The review focuses on whether the task is reproducible, clearly specified, blind to ground-truth information, multi-step, cross-tool, and time-independent. Annotators also check for potential shortcuts, information leakage, or inconsistencies between the task prompt and the evaluation materials. Tasks that fail any mandatory criterion are revised or excluded. This process ensures that the retained tasks provide reliable and deterministic measurements of agent performance. Table 7 shows the number of tasks we reviewed and

Environment Type	Snapshot Method	Recorded State
File system	Traverse the target environment directory using find XXX_Env/ -type f, sort the file list, and compute hashes for both file paths and file contents.	A deterministic digest of the file hierarchy and all task-related file contents.
Calendar events	Query calendar events in a fixed future window using icalbuddy eventsFrom:today to:+30d, and compute a hash of the normalized output.	A snapshot of scheduled calendar events within the evaluation window.
Reminders	Use AppleScript to query reminder names, e.g., osascript -e 'tell app "Reminders" to get name of reminders'.	The set of reminder entries visible to the system at snapshot time.
Notes	Use AppleScript to query note titles, e.g., osascript -e 'tell app "Notes" to get name of notes'.	The set of note records relevant to the initialized environment.
Email state	Use AppleScript or mail-client queries to count messages under task-specific labels, including drafts and flagged messages.	The number and status of task-relevant email records, such as drafts, labels, or flags.

Table 6: Snapshot methods for validating reproducible benchmark environments. Each snapshot records a deterministic representation of the corresponding environment state and can be compared before and after task execution.

Metric	Value
Candidate task instances	46
Retained benchmark tasks	40
Excluded after human audit	6
Retention rate	87.0%
Average repair iterations	2.67

Table 7: Statistics of the STAGE-Claw construction pipeline.

filtered. The human annotators only participated in task auditing but not in task modification.

B Case Studies

Case 1: correct content in the wrong state channel. In *Competitor Radar*, one Qwen3.5-Plus run produced a high-quality briefing and correctly resolved major data traps, including pricing reconciliation and exclusion of a rumored competitor. However, it wrote calendar information to an `.ics` file and notes to a Markdown file rather than creating real Apple Calendar events and an Apple Notes entry. The final score remained non-passing despite strong analytical content, illustrating why state-based evaluation is necessary: file artifacts that describe the intended actions are not equivalent to durable tool-state changes.

Case 2: early timeout before state construction. In *Project Progress Broadcaster*, a GLM-5 run timed out after reading the input files and before creating any required deliverables. The evaluator found no progress summary, verification report, reminders, notes, or URL-verification evidence. This case represents an incomplete-execution failure: the agent spent its budget on initial data gathering but did not transition to synthesis, validation, and state updates. Such failures are difficult to diagnose from final text alone, but are exposed directly by executable checks over missing files and tool state.

Case 3: ambiguous evidence and unsupported verification. In *Citation Gap Finder*, a DeepSeek-V4-Pro run correctly extracted many bibliography entries and identified several uncited or duplicate references, but failed the key ambiguous-citation trap by matching a single-author citation to an incompatible multi-author bibliography item. It also claimed URL verification without corresponding tool evidence and did not create the required Notes or Calendar artifacts. The case combines reasoning, provenance, and state-update failures: the report is superficially coherent, yet the evaluator shows that important claims are unsupported and required persistent actions are absent.

C Task Prompt

This appendix describes the prompt protocol used to construct reproducible, objectively scorable benchmark tasks for OpenClaw. Each benchmark task is represented as a self-contained directory containing environment construction instructions, an agent-facing task prompt, and ground-truth materials. The process separates task authoring, environment initialization, task execution, and evaluation

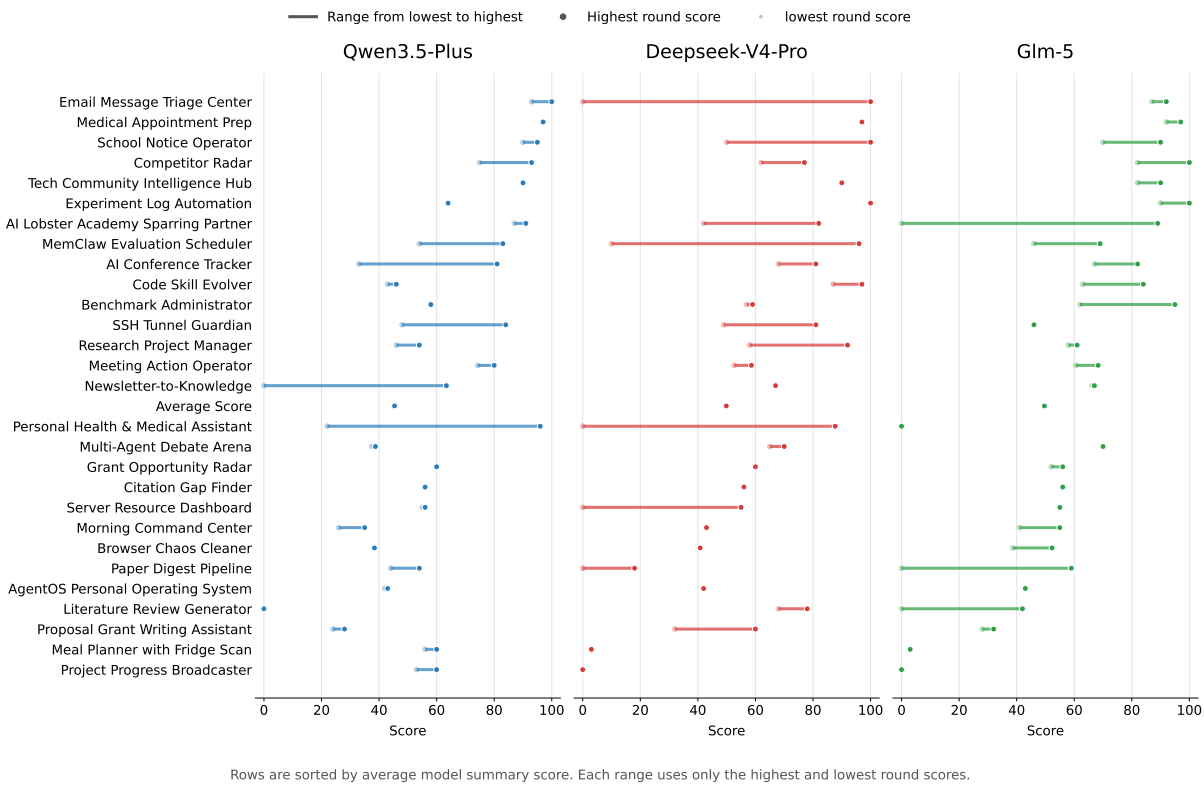

Figure 5: Range plot of three models across benchmark tasks. Horizontal line shows the score range across three evaluation rounds. Tasks are sorted by the average score across the three models.

in order to reduce information leakage and ensure reproducibility.

C.1 Benchmark Author Prompt

The benchmark author is instructed to design one reproducible, novel, and objectively scorable task. The author is explicitly prohibited from solving or scoring the task. A template is shown in Figure 6.

C.2 Environment Construction Prompt

The environment construction prompt is used to guide the environment builder in creating a deterministic task environment. It specifies the target directory, initialization workflow, and required ground-truth materials. A template is shown in Figure 7.

C.3 Agent Task Prompt

The agent-facing prompt is the only instruction shown to OpenClaw during benchmark execution. It describes the user objective and available task directory, but intentionally excludes all oracle information, scoring criteria, and ground-truth answers. A template is shown in Figure 8.

D Task Introduction

This section provides an introduction of the 40 STAGE-Claw tasks. The descriptions summarize user-facing objectives and required task capabilities, without revealing hidden ground truth, scoring rubrics, or verifier logic.

T01. *AI Conference Tracker*. This task requires the agent to consolidate conference schedules, email updates, user preferences, and venue information, produce an attendance plan, and create the corresponding calendar events, reminders, and notes.

T02. *AI Lobster Academy Sparring Partner*. This task requires the agent to schedule three exam-preparation study sessions for five members, assign practice topics, and create the associated calendar entries, notes, and reminders.

T03. *API Migration Planner*. This task requires the agent to analyze migration materials for a transition from a v1 API to a v2 API, reconcile conflicting information, and produce a phased migration plan.

T04. *AgentOS Personal Operating System*. This

Benchmark Author Prompt

```
You are the OpenClaw Benchmark Author. Design one reproducible, novel, objectively-
    scorable benchmark task. You do NOT solve tasks. You do NOT score tasks.

## Theme {{THEME}}

## Step 1Explore the Environment

Survey what OpenClaw can do and identify what is relevant to {{THEME}}.
Available capabilities include, but are not limited to:
- File read/write/search, terminal command execution
- Web search, URL fetch
- Email send/read, calendar add/query, reminders, notes
- Feishu message read, wiki/doc read/write
- System information: macOS version, disk, memory, CPU,
  running processes
- Chrome control: tabs, history, page JavaScript execution
- Weather query, stock/fund data
- macOS desktop notifications, multimedia control through Music.app

Answer these before designing:
1. Which two to four capabilities are most naturally exercised by
   {{THEME}}?
2. What real-world scenario would a user actually face in this theme?
3. What data or system state needs to be seeded before the agent starts?
4. What is the single unambiguous success condition?
5. How could an agent shortcut or cheat? How is that blocked?

## Step 2Design Requirements

The task must satisfy the following requirements:
- Multi-step: at least three distinct operations.
- Cross-tool: at least two different tools or information sources.
- Reproducible: env_make.md produces an identical state on every run.
- Blind: question.md contains no oracle information.
- Gradable: the result is verifiable by a script or structured checklist,
  without human judgment.
- Time-independent: the task does not rely on real clocks or real-time APIs.

## Step 3Build Task

/Users/three/Desktop/file_sort_eval_bench is an example of a
specific task. /Users/three/Desktop/template_bench is a task template.

env_make.md Environment setup guide for the builder, not the agent.
It must include: prerequisites, exact shell commands, idempotent setup steps, scene
    layout; teardown instructions, verification sanity checks.
question.md Task prompt shown to the agent.
It must include: the task objective, the expected deliverable, the working directory,
    operational constraints.
It must contain no ground truth, no scoring criteria, and no oracle
information.
Ground_truth/eval.md Evaluator-only scoring guide.
It must include: the expected final state, a scoring rubric specifying criteria,
    points, and how to check them, the passing threshold, common failure modes.

Build the task and store it in:
~/Desktop/XXX_Bench/

## Step 4Self-Review

Before finalizing the benchmark, verify the following:
- Can a builder follow env_make.md without ambiguity?
- Can an evaluator score the result without asking the agent?
- Is there a natural hard mode for future difficulty scaling?
```


Figure 6: Benchmark author prompt in STAGE-Claw.

Environment Construction Prompt

```
You are an AI assistant completing the following task using your available tools.
Task: {{TASK_NAME}}
Description: {{TASK_DESCRIPTION}}

## Scene Layout

Source directory:
~/Desktop/XXX_Bench/
Target directory for environment configuration:
~/Desktop/XXX_Env/{docs,images,code,data,...}/
Task initialization is not limited to files. The builder may configure any relevant
    part of the computer state, provided that the resulting environment is reproducible
    and can be verified.

## Ground Truth

Prepare gold answers or evaluation methods, such as assessment scripts
or structured assessment criteria, in:
~/Desktop/XXX_Bench/Ground_truth/

## Workflow

Describe how to construct the benchmark environment.
Step 1. Specify the prerequisite checks and create the required directory structure.
Step 2. Seed all task-related files, data, application state, or system state needed
    before the agent starts.
Step 3. Verify that the initialized environment matches the intended task state.
Step 4. Document teardown or reset procedures so that the benchmark can be
rerun from a clean state.

## Notes

- Plan before acting.
- If a step fails, document the reason rather than silently skipping it.
- Do not guess; use tools to verify uncertain facts.
- Setup steps should be idempotent whenever possible.
```

Figure 7: Environment construction prompt in STAGE-Claw.

Agent Task Prompt

```
You are a human assistant and your task is to complete the following
user request.

Your task-related files are stored in:~/Desktop/XXX_Env/
Please carefully plan and complete the task.
Task name: XXX
Task description:
XXX

Attention:
1. Use only the information and files available in the task environment
   unless the task explicitly permits other sources.
2. Complete the requested deliverable in the specified working directory.
3. Do not modify evaluator-only files or ground-truth materials.
4. Ensure that the final output is saved in the required format and
   location.
5. If intermediate files are created, keep them organized and avoid
   overwriting unrelated files.
```

Figure 8: Agent task prompt in STAGE-Claw.

task requires the agent to integrate multi-source information for a family gathering, resolve conflicts, compute the budget, and generate the final coordination plan.

T05. *Benchmark Administrator*. This task requires the agent to process AgentOlympics 2026 team registration information, filter eligible teams, assign competition tracks, and record the relevant calendar and note entries.

T06. *Browser Chaos Cleaner*. This task requires the agent to clean Firefox and Chrome bookmark exports by deduplicating entries, categorizing links, filtering invalid URLs, and generating a summary with follow-up reminders.

T07. *Citation Gap Finder*. This task requires the agent to audit in-text citations and bibliography entries in a technical report, identifying missing, unused, duplicated, or inconsistent references.

T08. *Code Skill Evolver*. This task requires the agent to revise a Python project according to 28 code-review comments, run the relevant tests, and document the implemented changes and verification results.

T09. *Competitor Radar*. This task requires the agent to integrate competitor information, verify feature links, generate a competitive-intelligence brief, and schedule appropriate follow-up actions.

T10. *Config Drift Detector*. This task requires the agent to process meeting-room booking requests from multiple sources, deduplicate entries, validate requests, resolve scheduling conflicts, and create valid calendar events.

T11. *Cross-Team Status Synthesizer*. This task requires the agent to synthesize status updates from multiple teams, resolve inconsistent reports, and produce an executive-facing project status summary.

T12. *Decision Log Keeper*. This task requires the agent to merge decision records from multiple sources, verify external references, produce a unified decision log, and create the corresponding calendar events.

T13. *Dependency Upgrade Operator*. This task requires the agent to analyze dependency versions in a software project, identify required upgrades, formulate an upgrade plan, and record related system-level follow-up items.

T14. *Docstring Debt Collector*. This task requires the agent to scan Python source files and API specifications, identify documentation debt, and create reminders for high-priority documentation issues.

T15. *Email Message Triage Center*. This task requires the agent to process RSVP emails for a team event, reconcile attendance status, and generate a verified participation report.

T16. *Experiment Log Automation*. This task requires the agent to analyze machine-learning experiment logs, identify the best completed run for each model family, and schedule a review calendar event.

T17. *Grant Opportunity Radar*. This task requires the agent to integrate grant opportunity tables, email updates, and institutional requirements, verify deadlines, and generate a structured tracking result.

T18. *Literature Review Generator*. This task requires the agent to select papers for a graduate-level course, organize them into four weekly reading themes, and create the corresponding calendar and reminder entries.

T19. *Local Dev Environment Fixer*. This task requires the agent to inspect project configuration files for tool-version conflicts, verify the local development environment, and generate a repair report.

T20. *Log-to-Incident Timeline*. This task requires the agent to parse production logs from multiple sources, filter noisy records, and construct an incident timeline.

T21. *Meal Planner with Fridge Scan*. This task requires the agent to generate a five-day meal plan based on fridge inventory, dietary restrictions, and the user's calendar schedule.

T22. *Medical Appointment Prep*. This task requires the agent to organize referral and medical documents, identify items requiring appointments, resolve conflicting information, and generate visit-preparation materials.

T23. *Meeting Action Operator*. This task requires the agent to extract action items from meeting transcripts, email summaries, and shared notes, then create the required system entries and reports.

T24. *MemClaw Evaluation Scheduler*. This task requires the agent to validate evaluation-task configurations, handle dependencies and resource allocation, schedule evaluation tasks, and generate a coordination report.

T25. *Morning Command Center*. This task requires the agent to synthesize information from multiple sources, generate a morning briefing for 2026-05-04, and create the corresponding system state.

T26. *Multi-Agent Debate Arena*. This task requires the agent to aggregate multi-round debate results, compute rankings, schedule the final round, and produce a verification report.

T27. *Newsletter-to-Knowledge*. This task requires the agent to extract, deduplicate, and categorize articles from five technical newsletters, then build a knowledge base and weekly digest.

T28. *Notebook Productionizer*. This task requires the agent to convert disorganized meeting notes into calendar events, reminders, and a readable structured report.

T29. *Onboarding Repo Guide*. This task requires the agent to coordinate Project 53 meeting materials, reconcile source conflicts, and create a meeting calendar entry, reminder, and explanatory note.

T30. *Paper Digest Pipeline*. This task requires the agent to merge paper data from multiple sources, deduplicate entries, filter records, verify links, and generate a weekly paper digest.

T31. *Personal Health & Medical Assistant*. This task requires the agent to organize personal health records and produce a current medication list, question list, and medical summary.

T32. *Project Progress Broadcaster*. This task requires the agent to integrate milestone data for the Phoenix Platform project, compute project status, and publish a structured progress update.

T33. *Proposal Grant Writing Assistant*. This task requires the agent to process IRB approval information from multiple institutions, verify approval status and expiration dates, and generate a tracking report.

T34. *Release Note Composer*. This task requires the agent to integrate release information from multiple sources, deduplicate overlapping items, verify external references, and produce accurate release notes.

T35. *Research Project Manager*. This task requires the agent to process literature records, resolve metadata conflicts, verify DOIs, and generate final references together with follow-up items.

T36. *SSH Tunnel Guardian*. This task requires the agent to audit SSH tunnel configurations and runtime status, identify high-priority maintenance items, and set monitoring reminders.

T37. *School Notice Operator*. This task requires the agent to process emergency school-closure information, notify relevant parties according to parent preferences, and create the necessary system records.

T38. *Server Resource Dashboard*. This task requires the agent to analyze server alerts and resource metrics, filter invalid alerts, and generate a resource-status and action report.

T39. *Slack Thread Summarizer*. This task requires the agent to extract key decisions and action items from Slack discussion threads, create calendar and reminder entries, and produce a structured summary.

T40. *Tech Community Intelligence Hub*. This task requires the agent to analyze multiple developer-community snapshots, compute topic popularity, identify controversial topics, and generate follow-up intelligence.