# AGENCYBENCH: Benchmarking the Frontiers of Autonomous Agents in 1M-Token Real-World Contexts

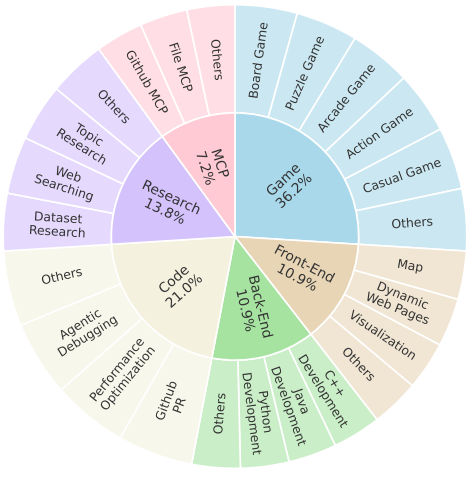
Keyu Li[1,2,4] Junhao Shi[2,4] Yang Xiao[3,4] Mohan Jiang[1,2,4] Jie Sun[1,4] Yunze Wu[2,4] Dayuan Fu[1,4]
Shijie Xia[1,2,4] Xiaojie Cai[2,4] Tianze Xu[2,4] Weiye Si[2,4] Wenjie Li[3] Dequan Wang[1,2,4] Pengfei Liu[†1,2,4]

[1]SII [2]SJTU [3]PolyU [4]GAIR

SII Open Source: AgencyBench Code AgencyBench

## Abstract

Large Language Models (LLMs) based autonomous agents demonstrate multifaceted capabilities to contribute substantially to economic production. However, existing benchmarks remain focused on single agentic capability, failing to capture long-horizon real-world scenarios. Moreover, the reliance on human-in-the-loop feedback for realistic tasks creates a scalability bottleneck, hindering automated rollout collection and evaluation. To bridge this gap, we introduce AGENCYBENCH, a comprehensive benchmark derived from daily AI usage, evaluating 6 core agentic capabilities across 32 real-world scenarios, comprising 138 tasks with specific queries, deliverables, and rubrics. These scenarios require an average of **90 tool calls, 1 million tokens, and hours of execution time** to resolve. To enable automated evaluation, we employ a user simulation agent to provide iterative feedback, and a Docker sandbox to conduct visual and functional rubric-based assessment. Experiments reveal that closed-source models significantly outperform open-source models (48.4% vs 32.1%). Further analysis reveals significant disparities across models in resource efficiency, feedback-driven self-correction, and specific tool-use preferences. Finally, we investigate the impact of agentic scaffolds, observing that proprietary models demonstrate superior performance within their native ecosystems (e.g., Claude-4.5-Opus via Claude-Agent-SDK), while open-source models exhibit distinct performance peaks, suggesting potential optimization for specific execution frameworks. AGENCYBENCH serves as a critical testbed for next-generation agents, highlighting the necessity of co-optimizing model architecture with agentic frameworks. We believe this work sheds light on the future direction of autonomous agents, and to facilitate community adoption, we release the full benchmark and evaluation toolkit at https://github.com/GAIR-NLP/AgencyBench.




| Benchmark | Avg. Tok. | Avg. Turns | Diverse Agentic | User Sim. | Docker Sandbox |
|---|---|---|---|---|---|
| Browsecomp | – | – | × | × | × |
| Terminal-bench | – | – | × | × | × |
| SWE-verified | – | 15 | × | × | ✓ |
| MCPUniverse | – | 7.5 | × | × | × |
| GAIA2 | 10K | 22.5 | ✓ | × | × |
| Toolathlon | 15K | 26.8 | ✓ | × | ✓ |
| UltraHorizon | 200K | 60 | ✓ | × | × |
| **AGENCYBENCH** | **1M** | **90** | ✓ | ✓ | ✓ |

Figure 1: **Overview of AGENCYBENCH. Left:** Distribution of the 32 scenarios and 138 tasks across 6 distinct agentic capabilities. **Right:** Comparison with existing benchmarks. AGENCYBENCH focuses on diverse, long-horizon real-world tasks, requiring an average of 1M tokens and 90 multi-turn tool uses. It integrates a user simulation agent for iterative feedback and a Docker-based sandbox for automated rubric-based assessment.

† Corresponding author.

## 1 Introduction

With the rapid advancement of Large Language Models (LLMs) (OpenAI, 2025b; DeepMind, 2025; Anthropic, 2025a; xAI, 2025), integrating these models with advanced scaffolds to form autonomous agents has become a paradigm shift. As these agents increasingly permeate diverse domains, ranging from economic production, scientific research, and software development to everyday use, establishing rigorous benchmark to measure their practical economic value and performance is becoming unprecedentedly urgent (Pan et al., 2025; OpenRouter, 2025; Kwa et al., 2025). However, current agent benchmarks face significant limitations: (1) Existing benchmarks often exhibit a scarcity of long-horizon tasks (Li et al., 2025a; Andrews et al., 2025) or focus narrowly on single agentic capability, such as tool use (Terminal-bench Team, 2025; Wu et al., 2025b), software engineering (Jimenez et al., 2023) or research (Wei et al., 2025; Xu et al., 2025), failing to capture the long-horizon nature and diversity of real-world tasks. (2) Furthermore, completing realistic tasks often necessitates continuous human feedback to guide agents through multi-turn interactions. This reliance on human-in-the-loop processes creates a bottleneck, restricting the automation of rollout collection and evaluation.

To bridge this gap, we introduce AGENCYBENCH, a comprehensive benchmark designed to evaluate agent capabilities through highly long-horizon, diverse, and authentic real-world tasks. By requesting AI researchers, active AI practitioners, and software engineer developers, we systematically construct 32 real-world scenarios evaluating 6 core agentic capabilities, comprising a total of 138 specific tasks. Each task is defined by queries (descriptions of task requirements), deliverables (descriptions of the expected outputs), and rubrics (evaluation criteria for assessment). These scenarios are notably demanding: on average, resolving a single scenario approximately **90 tool calls, consumes 1 million tokens, and requires hours of execution time**, rigorously evaluating agents' ability to maintain context and execute logic over extended periods to satisfy multi-turn, long-horizon real-world needs. The comparison between AGENCYBENCH and various other benchmarks is shown in Figure 1.

To facilitate scalable and automated rollout collection, we develop an agent scaffold equipped with a comprehensive tool suite operating within an isolated *workspace*. In this environment, the agent engages in multi-turn interactions to generate raw deliverables, assisted by a user simulation agent that mimics real-world scenarios and provides iterative feedback to bypass human-in-the-loop limitations. Subsequently, these deliverables are synchronized to a Docker-based remote sandbox, which emulates human-computer operations (e.g., UI rendering, mouse/keyboard inputs) to produce visual evaluation artifacts. The process concludes in a separate *eval-space*, where these artifacts and raw deliverables undergo automated rubric-based assessment using executable scripts.

Extensive experiments on AGENCYBENCH indicate that closed-source models achieve an average score of 48.4%, while open-source models average 32.1%. Closed-source models range from 56.5% (GPT-5.2) to 44.3% (Grok-4.1-Fast), whereas open-source models span from 38.6% (GLM-4.6) to 27.0% (Qwen-3-235B-A22B-Thinking). The overall performance reveals that current frontier models still struggle to master the long-horizon, real-world tasks. Further analysis reveals distinct behavioral differences among the models: GPT-5.2 demonstrates stronger capabilities in feedback-driven self-correction, Grok-4.1-fast exhibits higher token utilization efficiency, Claude-4.5-Opus shows a preference for shell-based tools, and Gemini-3-Pro favors file-related and memory management tools. Additionally, comparative studies on agentic scaffolds highlight a 'home-field advantage', where models achieve peak performance when paired with their native or specifically optimized frameworks.

In summary, our contributions are as follows: (1) We introduce AGENCYBENCH, a challenging benchmark with 138 authentic tasks across 32 scenarios to evaluate long-horizon agentic capabilities. (2) We develop a unified evaluation framework leveraging user simulation agent and Docker-based remote sandboxes to achieve fully automated evaluation. (3) We provide a comprehensive evaluation of frontier models, quantifying the gap between proprietary and open-source models and uncovering distinct behavioral patterns.

## 2 Related Work

**Large Language Model Agents** The continuous and rapid evolution of large language models (LLMs) (Team et al., 2025; Yang et al., 2025; Liu et al., 2025; GLM Team, 2025; xAI, 2025; Anthropic, 2025a; DeepMind, 2025; OpenAI, 2025b) has fundamentally reshaped the landscape of artificial intelligence, propelling the field from simple conversational tasks to complex reasoning (Xiao et al., 2025c,d,b) and multi-turn tool-use tasks (Li et al., 2025b; Xiao et al., 2025a; Wu et al., 2025a). To enable these models to tackle intricate real-world challenges, numerous agentic scaffolds have been developed (Yao et al., 2022; Yang et al., 2024; Wang et al., 2024; Cursor, 2025; OpenAI, 2025a; Anthropic, 2025c), effectively unlocking the potential for long-horizon task completion through iterative feedback and structured planning. Agents are now expected to autonomously navigate diverse scenarios, ranging from full-stack front-end development to complex game engine manipulations.

**Agent Benchmarks** To rigorously evaluate these growing capabilities, various benchmarks have emerged, focusing on specific vertical domains like tool use (Li et al., 2025a; Barres et al., 2025; Terminal-bench Team, 2025; Wu et al., 2025b), software development (Miserendino et al.; DesignArena Team, 2025; Jimenez et al., 2023),

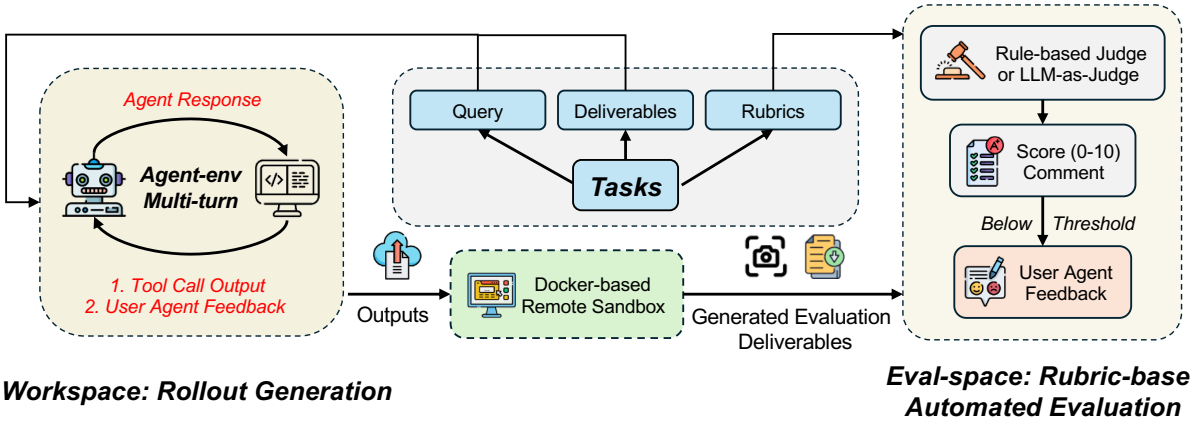



Figure 2: **AGENCYBENCH Rollout Generation and Evaluation Pipeline.** Rollout generation takes place within *workspace*, where the agent receives task queries and deliverables, completing tasks through multi-turn interactions with the environment (e.g., tool execution results and feedback from the user simulation agent). Upon task completion, deliverables are synced to a Docker sandbox for operation execution (e.g., UI actions), and resulting artifacts are transferred to *eval-space* for scoring $(0-10)$ via rule-based or LLM-based judges based on task rubrics.

or open-ended research (Wei et al., 2025; Andrews et al., 2025; Patwardhan et al., 2025), with some recent efforts assessing potential economic utility (Miserendino et al.; Patwardhan et al., 2025; Xiao et al., 2025b). However, current benchmarks often lack the necessary complexity to differentiate frontier models, as the limited number of tool calls and shallow interaction depths remain insufficient to test the upper limits of modern agents (Wu et al., 2025b; Andrews et al., 2025; Li et al., 2025a; Luo et al., 2025). AGENCYBENCH addresses this critical gap by curating 138 authentic, high-fidelity tasks across 6 agentic capabilities, where the average rollout exceeds 1M tokens and necessitates over 90 precise tool calls, significantly raising the bar for task difficulty and context retention.

## 3 AGENCYBENCH

In this section, we detail the hierarchical design of AGENCYBENCH (Section 3.1), the data collection process and the formal definition of interaction rollouts (Section 3.2). Finally, we describe our automated evaluation framework (Section 3.3), which leverages a Docker-based remote sandbox and rubric-based rigorous, scalable, and reproducible assessment. Rollout generation and evaluation pipeline is illustrated in Figure 2, with a specific evaluation example detailed in Figure 3.

### 3.1 AGENCYBENCH Design

#### 3.1.1 Design Pattern

**Capabilities, Scenarios, and Tasks** To capture the multifaceted and long-horizon nature of real-world tasks, AGENCYBENCH is structured hierarchically. It evaluates 6 core agentic capabilities: game development, front-end development, back-end development, code generation, research, and MCP tool use. These capabilities are distributed across 32 authentic real-world scenarios, such as developing a Gomoku game from scratch (developing board game capability), conducting project-level code debugging (agentic debugging capability), performing in-depth corporate research (research capability), etc. Serving as a complete testing unit, each scenario is structured as a logical hierarchy of 1 to 5 tasks, arranged in ascending order of difficulty and presented sequentially, where the completion results of preceding tasks influence subsequent ones. Through this design, scenarios are expanded into a comprehensive collection totaling 138 distinct tasks. This design is intended to simulate real-world tasks that progress from basic to complex tasks, thereby requiring agent to maintain context and execute logic over extended periods to satisfy multi-turn, long-horizon nature of real-world tasks.

**Workspace, Eval-space and Scaffold** To address the limitations of existing benchmarks that rely on unscalable human experts' feedback for complex environments, we design a robust execution infrastructure. Each task operates within an isolated *workspace* to ensure reproducibility and prevent state interference. This workspace is equipped with an agent scaffold (a suite of tools including file manipulation, command-line execution, web search, context management, ...) allowing the agent to generate rollouts in a realistic setting. For each task, the agent engages in multi-turn interactions with the environment using the scaffold within the *workspace* to generate deliverables. To facilitate assessment, deliverables are synced to a Docker-based remote sandbox, which emulates human computer operations (e.g., mouse clicks, UI rendering) to produce visualizable artifacts. These artifacts are subsequently

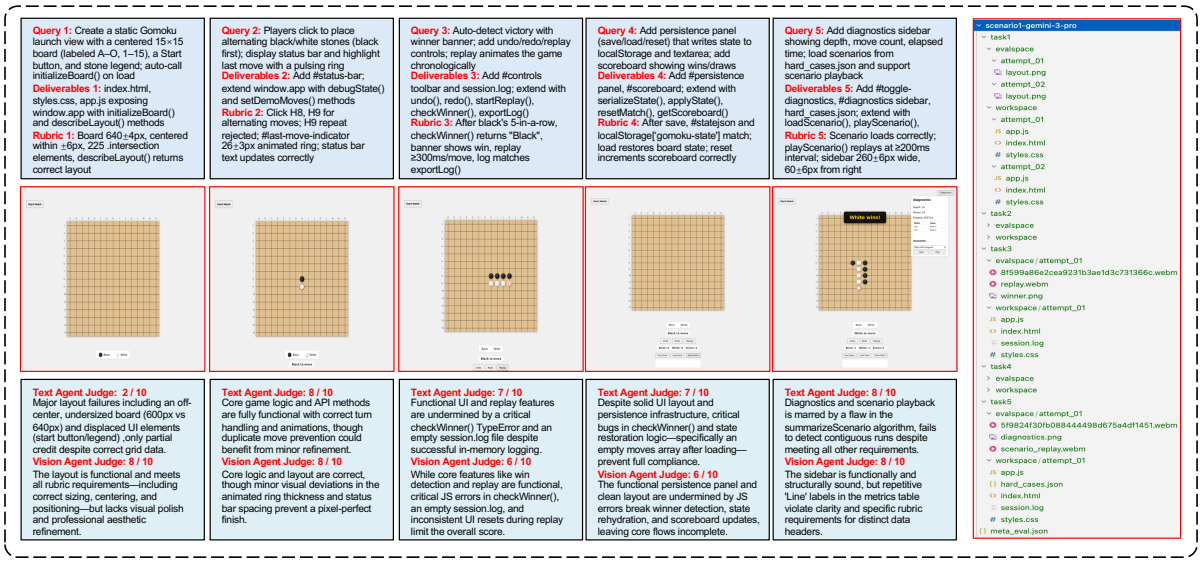



Figure 3: **An Illustrative Evaluation Scenario in AGENCYBENCH: Developing a Gomoku Game.** The scenario consists of five sequential tasks with increasing complexity, requiring the incremental addition of new features. The primary deliverables include HTML, CSS, and JS source code. Evaluation scripts execute these files within a remote Docker sandbox, performing interactive operations such as clicking, screen recording, and capturing screenshots (visualized as video frames in the figure). The resulting evaluation artifacts are retrieved to *eval-space*, where text and vision agents assess the code and visual deliverables, respectively, providing scores and qualitative feedback based on rubrics. **Right:** The file organization architecture during runtime, showing the isolated workspace and eval-space for each task to ensure environmental consistency and prevent cross-task interference.

transferred to a local *eval-space*, where the evaluation process is completed automatically using executable scripts without human intervention.

### 3.1.2 Data Collection

We employ 20 human experts, including AI researchers, active AI practitioners, software engineer developers to collect real-world tasks. Based on this data, these human experts systematically construct the 32 scenarios and 138 tasks. For each task, experts manually construct and verify three key components: (1) the **Query**, which describes the specific requirements; (2) the **Deliverables**, which define the expected file outputs or terminal states; and (3) the **Rubrics**, which establish the objective criteria for assessment. Furthermore, experts develop executable evaluation scripts for each task. A separate panel of four experts conducts a comprehensive review of the entire dataset to verify descriptive accuracy, difficulty calibration, and environment configurations. To ensure the highest quality standards, a strict unanimous consensus policy is enforced: a task is only finalized if all experts reach a full agreement. If any discrepancy arises, the task is flagged and remanded for revision, requiring subsequent re-evaluation until it meets the 100% approval threshold.

## 3.2 Rollout Collection

**Rollout Definition** We formalize the interaction process as a sequence of states and actions. Assuming a scenario consists of five sequential tasks, we denote their individual rollouts as $\tau_1$, $\tau_2$, ... $\tau_5$. These are defined as follows:

$$\tau_1 = (q_1, a, t, \ldots, a, u_{11}, \ldots, a, u_{12}, ..., a, t, ...)$$
$$\tau_2 = (q_2, a, t, \ldots, a, u_{21}, \ldots, a, u_{22}, ..., a, t, ...)$$
$$...$$
$$\tau_5 = (q_5, a, t, \ldots, a, u_{51}, \ldots, a, u_{52}, ..., a, t, ...)$$

Here, for the $i$-th task, $q_i$ represents the initial query provided by AGENCYBENCH. The pair $(a, t)$ denotes the agent's reasoning and specific tool calls (e.g., file writing, shell commands). A user simulation agent provides feedback $u_{ij}$ (which prompts further agent actions) whenever the deliverables fail to meet the score threshold prescribed by the rubric. The complete rollout $\tau$ for the scenario is defined as the ordered concatenation of these trajectories:

| Model | Game ↑ | Frontend ↑ | Backend ↑ | Code ↑ | Research ↑ | MCP ↑ | $S_{Avg}$ ↑ | $Att$ ↓ |
|---|---|---|---|---|---|---|---|---|
| *Proprietary Models* | | | | | | | | |
| GPT-5.2 | 52.2 | 74.7 | 61.0 | 50.7 | 64.4 | 52.1 | 56.5 | 1.46 |
| Claude-4.5-Opus | 52.1$_{-0.1}$ | 49.3$_{-25.4}$ | 49.6$_{-11.4}$ | 24.0$_{-26.7}$ | 68.8$_{+4.4}$ | 64.4$_{+12.3}$ | 47.7$_{-8.8}$ | 1.54 |
| Gemini-3-Pro | 60.7$_{+8.5}$ | 81.0$_{+6.3}$ | 31.3$_{-29.7}$ | 23.5$_{-27.2}$ | 40.0$_{-24.4}$ | 61.8$_{+9.7}$ | 46.9$_{-9.6}$ | 1.46 |
| Claude-4.5-Sonnet | 56.5$_{+4.3}$ | 51.6$_{-23.1}$ | 35.3$_{-25.7}$ | 17.3$_{-33.4}$ | 71.4$_{+7.0}$ | 62.3$_{+10.2}$ | 46.4$_{-10.1}$ | 1.49 |
| Grok-4.1-fast | 38.8$_{-13.4}$ | 65.7$_{-9.0}$ | 37.3$_{-23.7}$ | 26.4$_{-24.3}$ | 63.8$_{-0.6}$ | 68.6$_{+16.5}$ | 44.3$_{-12.2}$ | 1.55 |
| *Open-Source Models* | | | | | | | | |
| GLM-4.6 | 59.2 | 64.3 | 20.0 | 11.9 | 32.0 | 49.8 | 38.6 | 1.54 |
| Kimi-K2-Thinking | 40.6$_{-18.6}$ | 54.7$_{-9.6}$ | 24.3$_{+4.3}$ | 11.8$_{-0.1}$ | 33.6$_{+1.6}$ | 31.4$_{-18.4}$ | 34.2$_{-4.4}$ | 1.79 |
| Deepseek-V3.2 | 36.5$_{-22.7}$ | 49.3$_{-15.0}$ | 20.7$_{+0.7}$ | 11.6$_{-0.3}$ | 22.0$_{-10.0}$ | 48.5$_{-1.3}$ | 28.6$_{-10.0}$ | 1.63 |
| Qwen-3-235B-A22B-Thinking | 40.4$_{-18.8}$ | 57.7$_{-6.6}$ | 3.3$_{-16.7}$ | 4.6$_{-7.3}$ | 34.4$_{+2.4}$ | 20.9$_{-28.9}$ | 27.0$_{-11.6}$ | 1.79 |

Table 1: **Main Experimental Results.** The table compares proprietary and open-source models. The rows for GPT-5.2 and GLM-4.6 are highlighted in blue as baselines. Colored subscripts indicate the performance gap compared to the baseline (red for improvement, blue for degradation).

$$\tau = (\tau_1, \tau_2, \tau_3, \tau_4, \tau_5)$$

This formulation captures the iterative and long-horizon nature of real-world tasks.

**User Simulation Agent** To simulate realistic human-agent collaboration without manual intervention, we implement a user simulation agent, responsible for providing feedback to the task executing agent, enabling targeted improvements based on task execution. Feedback is determined based on the fulfillment of the current task's rubrics. For instance, if an agent completes only 6 out of 10 rubrics, the user simulation agent will return the 4 failed rubrics along with their specific reasons for failure. We utilize Claude-4-Sonnet with a temperature setting of 0.0 for this role (detailed prompts in Appendix A.2).

To ensure the validity of the user simulation agent, we conducted a human verification study on 50 randomly sampled interaction rollouts. Four human experts independently drafted justifications for unmet rubrics, which another four experts then independently scored against the agent's feedback. Scores were assigned on an integer scale from 0 to 5, representing 'Highly Inconsistent', 'Inconsistent', 'Uncertain', 'Consistent', and 'Highly Consistent', respectively. The final average score reached 4.69, demonstrating a high degree of alignment and confirming that the agent serves as a reliable surrogate for human experts.

### 3.3 Evaluation

Our evaluation framework is entirely rubric-based, ensuring standardized assessment across diverse domains. We employ executable evaluation scripts that map the agent's deliverables to a score ranging from 0 to 10. Depending on the nature of the task, we utilize either rule-based methods or LLM-as-judge mechanisms.

**Rule-based Evaluation** For tasks with objectively verifiable ground truth, such as correct tool execution, mathematical optimization, or specific file generation—we employ rule-based evaluation. In these cases, the rubrics are directly translated into assertion logic within the evaluation scripts. The final score is calculated by mapping the pass rate of these assertions or the optimization metric to the 0-10 scale.

**LLM-as-Judge** For tasks involving subjective qualities or complex visual outputs (e.g., game aesthetics, front-end layout), we employ an LLM-as-judge. Specifically for game and frontend scenarios, we implement a multimodal judging system: **(1) Text-based Judge:** Evaluates code quality and logic based on the text deliverables and rubrics. We utilize Claude-4-Sonnet with a temperature of 0.0. **(2) Vision-based Judge:** Evaluates dynamic behavior and visual correctness based on screenshots and recordings captured from remote sandbox interactions. We utilize Gemini-2.5-pro with a temperature of 0.0. Detailed prompts are provided in Appendix A.2.

For tasks requiring visual deliverables, the final score is calculated as the average of the ratings from the text and vision-based judges; otherwise, the score is determined solely by the text-based judge. To validate the reliability of our LLM judges, we compare their ratings against human annotations on a held-out set of 50 tasks. Four human experts independently scored these 50 tasks based on the rubrics. The results showed a Kappa score of 0.93 between the human and LLM judge scores, demonstrating the reliability of the evaluation.

To comprehensively evaluate agent capabilities and resource use in AGENCYBENCH, we adopt the following metrics for assessment.

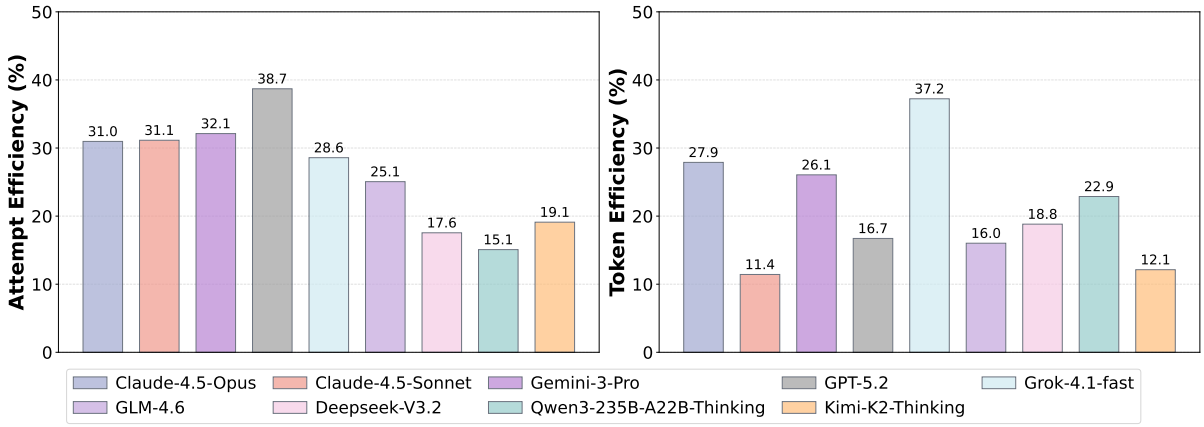



Figure 4: **Efficiency Comparison Across Models.** Efficiency is calculated by dividing the average score by the number of attempts and average token consumption, respectively. GPT-5.2 achieves the highest attempt efficiency, while Qwen-3-235B-A22B-Thinking ranks the lowest. For token efficiency, Grok-4.1-Fast performs best, whereas Claude-4.5-Sonnet is the least efficient one.

**Metric: Average Score ($S_{Avg}$)** Calculated as the percentage derived from rubric-based evaluations (e.g., satisfying 6 out of 10 criteria yields 60%), where higher scores indicate superior capability.

**Metric: Average Attempts ($Att$)** Measures the average iterations per scenario. A task is passed if the task score is at least $60\%$. If a task fails, the user simulation agent provides feedback up to a maximum of $K$ rounds until the threshold is met. $Att$ denotes the average rounds used (ranging from 1 to $K$), where lower values imply stronger autonomy. Let $M$ denote the total number of tasks in the current scenario, and $M_{Att}$ represent the total count of attempts used across all tasks. The Average Attempts ($Att$) is defined as:

$$Att = \frac{M_{Att}}{M} \tag{1}$$

**Metric: Pass Rate ($Pass@k$)** Given $N$ tasks in a scenario, let $N_{pass}$ be the number of tasks achieving the $60\%$ score threshold within $k$ feedback rounds. The metric is defined as:

$$Pass@k = \frac{N_{pass}}{N} \tag{2}$$

We set the feedback limit to 1 and 2, reporting $Pass@1$ (up to 1 feedback round) and $Pass@2$ (up to 2 feedback rounds), respectively.

**Metric: Efficiency** We denote $Tok$ as average tokens used per scenario and compute *Attempt Efficiency* ($E_{att}$) and *Token Efficiency* ($E_{tok}$) to normalize performance against resource costs where higher values indicate better resource optimization:

$$E_{att} = \frac{S_{avg}}{Att}, \quad E_{tok} = \frac{S_{avg}}{Tok} \tag{3}$$

## 4 Experiments

In this section, we conduct a systematic evaluation of various LLMs on AGENCYBENCH. We first outline experimental setup and metrics (Section 4.1), followed by analysis of overall performance across agentic capabilities (Section 4.2). We then investigate the crucial ability of self-correction through feedback (Section 4.3), analyze resource use (Section 4.4) and the economic efficiency of models (Section 4.5), and decode the distinct behavioral patterns in tool usage (Section 4.6). Finally, we examine the influence of different agentic scaffolds on model performance (Section 4.7).

### 4.1 Experimental Setup

**Models and Scaffold** We evaluate a comprehensive suite of models, comprising closed-source LLMs: GPT-5.2 (OpenAI, 2025b), Claude-4.5-Opus, Claude-4.5-Sonnet (Anthropic, 2025a), Grok-4.1-Fast (xAI, 2025) and

| Model | Pass@1 | Pass@2 | Rise(%) |
|---|---|---|---|
| *Proprietary Models* | | | |
| GPT-5.2 | 28.1 | 53.1 | 88.9 |
| Claude-4.5-Sonnet | 21.9 | 40.6 | 85.7 |
| Claude-4.5-Opus | 15.6 | 28.1 | 80.0 |
| Gemini-3-Pro | 28.1 | 37.5 | 33.3 |
| Grok-4.1-Fast | 25.0 | 31.3 | 25.0 |
| *Open-Source Models* | | | |
| GLM-4.6 | 28.1 | 37.5 | 33.3 |
| Kimi-K2-Thinking | 6.3 | 25.0 | 300.0 |
| DeepSeek-V3.2 | 9.4 | 9.4 | 0.0 |
| Qwen-3-235B-A22B-Thinking | 3.1 | 9.4 | 199.7 |

Table 2: **Impact of User Simulation Agent Feedback Attempts.** While additional interactions significantly boost performance for certain models (e.g., GPT-5.2 and Kimi-K2-Thinking), the gains are less pronounced for others like DeepSeek-V3.2 and Grok-4.1-Fast.

open-source LLMs: GLM-4.6 (GLM Team, 2025), DeepSeek-V3.2 (Liu et al., 2025), Qwen-3-235B-A22B-Thinking (Yang et al., 2025), Kimi-K2-Thinking (Team et al., 2025). All models are accessed via OpenRouter API with the temperature set to 0.7 to balance creativity and determinism. We utilize the agentic scaffold described in Section 3.1.1, which is equipped with a robust set of tools (detailed in Table 6).

### 4.2 Main Results

**Overall Performance and Efficiency** Table 1 highlights a capability gap between proprietary and open-source models. GPT-5.2 achieves the highest overall average score (56.5%) among proprietary models, whereas GLM-4.6 leads the open-source category with 38.6%. Conversely, Qwen-3-235B-A22B-Thinking records the lowest performance (27.0%), underscoring that substantial room for improvement. Regarding user simulation agent feedback attempts (Att), GPT-5.2 and Gemini-3-Pro demonstrate superior capability, requiring the fewest average attempts (1.46) to complete tasks. In the open-source sector, GLM-4.6 performs comparably to proprietary models (1.54), while Qwen-3-235B-A22B-Thinking and Kimi-K2-Thinking struggle with higher attempt counts (1.79), indicating weaker error-recovery abilities.

| Model | Tok (M) | T (H) | Turns |
|---|---|---|---|
| *Proprietary Models* | | | |
| GPT-5.2 | 3.4 | 0.6 | 89.0 |
| Claude-4.5-Sonnet | 4.1 | 0.9 | 64.0 |
| Gemini-3-Pro | 1.8 | 0.3 | 37.0 |
| Grok-4.1-Fast | 1.2 | 0.3 | 37.0 |
| Claude-4.5-Opus | 1.7 | 0.8 | 36.0 |
| *Open-Source Models* | | | |
| GLM-4.6 | 2.4 | 0.6 | 105.0 |
| Kimi-K2-Thinking | 2.8 | 1.2 | 65.0 |
| DeepSeek-V3.2 | 1.5 | 1.0 | 21.0 |
| Qwen-3-235B-A22B-Thinking | 1.2 | 1.4 | 21.0 |

Table 3: **Resource Usage Comparison.** AGENCYBENCH tasks typically require 1 million tokens and time on the scale of hours to complete, highlighting their long horizon characteristic.

**Agentic Capabilities** Performance varies significantly across agentic capabilities, revealing distinct specializations. Gemini-3-Pro dominates game (60.7%) and front-end (81.0%). GPT-5.2 excels in back-end and code, while Claude-4.5-Sonnet achieving the highest in research (71.4%). Among open-source models, GLM-4.6 exhibits

https://openrouter.ai/

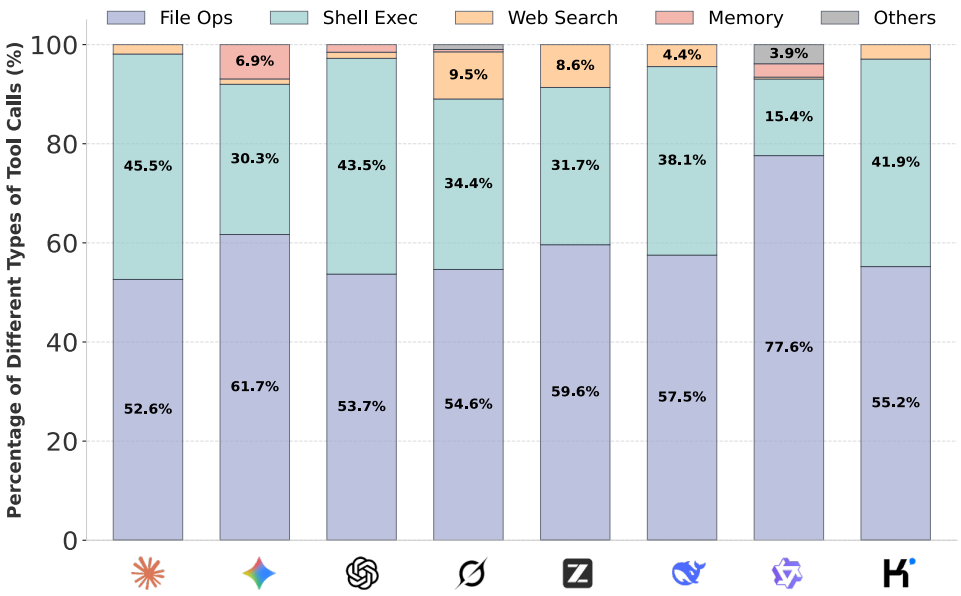



Figure 5: **Tool Invocation Patterns Across Models.** Claude-4.5-Opus and GPT-5.2 shows a preference for shell execution tools, while Gemini-3-Pro and Qwen-3-235B-A22B-Thinking favor file operation and memory management. Grok-4.1-Fast, GLM-4.6, and Deepseek-V3 series exhibit a strong preference for web search tools.

balanced performance, while Qwen-3-235B-A22B-Thinking demonstrates relative strength in research despite lower overall average.

### 4.3 Feedback-driven Self-correction Analysis

Table 2 quantifies feedback-driven self-correction ability by *Pass@1* and *Pass@2*. Top-tier proprietary models demonstrate sophisticated error handling. GPT-5.2 achieves an 88.9% relative increase, and the Claude series similarly exceeds 80% improvement. In contrast, while Gemini-3-Pro matches the initial Pass@1 of GPT-5.2, its ability to leverage feedback is markedly lower (33.3% Rise), suggesting that while its initial intuition is strong, its self-correction mechanisms are less responsive. In the open-source domain, Kimi-K2-Thinking and Qwen-3-235B-A22B-Thinking achieve remarkable improvements after feedback (300% and nearly 200%, respectively). Conversely, DeepSeek-V3.2 achieves (0.0% Rise), persistently adhering to erroneous paths despite external critique.

### 4.4 Resource Consumption Analysis

Table 3 details the trade-off between performance ceilings and costs: Tok represents the token consumption, measured in millions; T represents the average scenario execution time, measured in hours; Turns represents the number of tool-calling rounds. GPT-5.2 acts as a 'brute-force' reasoner, consuming 3.4 million tokens and 89 turns on average to secure top scores. In contrast, Grok-4.1-Fast represents the pinnacle of speed and frugality (1.2M tokens, 0.3h). GLM-4.6 exhibits a unique profile: despite a high turns count (105), its resource usage remains moderate. Notably, Kimi-K2-Thinking and Qwen-3-235B-A22B-Thinking incur the highest time costs (1.2h), likely due to the latency of generating internal reasoning traces.

### 4.5 Attempt and Token Efficiency Analysis

To decouple raw performance from expenditure, we analyze efficiency in Figure 4. *Attempt Efficiency* is dominated by GPT-5.2 (38.7%), indicating that its high resource usage is justified by a high success rate per attempt. *Token Efficiency* favors Grok-4.1-Fast (37.2%), making it the most economically viable choice for resource-constrained environments. Claude-4.5-Sonnet ranks lowest (11.4%), indicating excessive token generation (4.1M) does not yield proportional performance gains.

### 4.6 Behavioral Patterns in Tool Invocation

Figure 5 reveals model architectures imprint distinct 'personalities' on problem-solving strategies: Claude-4.5-Opus and GPT-5.2 prefer system-level manipulation via shell execution (45.5% and 43.5%); Gemini-3-Pro distinctively utilizes explicit memory tools (6.9%), suggesting a strategy rooted in managing long-horizon context banks; Qwen-3-235B-A22B-Thinking exhibits an extreme reliance on file operations (77.6%), prioritizing direct content

| Model Scaffold | Our Scaffold | Claude-Agent-SDK | OpenAI-Agents-SDK |
|---|---|---|---|
| *Proprietary Models* | | | |
| GPT-5.2 | 57.4 | $53.5_{-3.9}$ | $58.7_{+1.3}$ |
| Claude-4.5-Opus | 50.8 | $71.3_{+20.5}$ | $47.1_{-3.7}$ |
| Gemini-3-Pro | 46.3 | $45.9_{-0.4}$ | $46.9_{+0.6}$ |
| *Open-Source Models* | | | |
| Kimi-K2-Thinking | 50.5 | $44.6_{-5.9}$ | $37.7_{-12.8}$ |
| Minimax-M2 | 45.8 | $54.4_{+8.6}$ | $43.0_{-2.8}$ |
| GLM-4.6 | 33.6 | $44.2_{+10.6}$ | $36.0_{+2.4}$ |

Table 4: **Impact of Agentic Scaffolds on Model Performance.** We evaluate models across 10 representative scenarios using three distinct frameworks: our custom scaffold, the Claude-Agent SDK, and the OpenAI-Agents-SDK. The results highlight the sensitivity of model performance to the agentic scaffold, with distinct 'native ecosystem' preferences observed for proprietary models. The rows for GPT-5.2 and Kimi-K2-Thinking are highlighted in blue as baselines. Colored subscripts indicate the performance gap compared to the baseline (red for improvement, blue for degradation).

verification; Grok-4.1-Fast and GLM-4.6 exhibit a high reliance on web search (9.5% and 8.6%), appearing to offload knowledge retrieval to external sources rather than relying on internal parametric memory.

### 4.7 Impact of Agentic Scaffolds

To investigate the influence of agentic scaffold on model performance, we conducted an ablation study on a subset of 10 representative scenarios using three distinct frameworks: our native scaffold (used in the main experiments), the Claude-Agent-SDK (Anthropic, 2025b), and the OpenAI-Agents-SDK (OpenAI, 2025c). We report the Average Score ($S_{Avg}$) in Table 4.

**Ecosystem Synergy in Proprietary Models** Experimental results reveal a significant 'Ecosystem Synergy' effect, where proprietary models demonstrate peak performance within their native frameworks. Most notably, Claude-4.5-Opus achieves a substantial performance boost of 20.5% when operating within the Claude-Code SDK compared to our generalist scaffold. This suggests a deep optimization between the model's training objective and its proprietary tool definitions/prompt structures. Similarly, GPT-5.2 shows a preference for the OpenAI-Agents-SDK (+1.3%), outperforming its results on third-party alternatives.

**Scaffold Sensitivity in Open-Source Models** Among open-source models, the impact of scaffold choice is heterogeneous. GLM-4.6 and Minimax-M2 exhibit strong compatibility with the Claude-Agent-SDK scaffold, improving by 10.6% and 8.6% respectively. This improvement may stem from the SDK's structured prompt engineering, which likely aligns well with the instruction-following capabilities of these models. **Furthermore, it suggests the possibility that these models may have been specifically optimized during training to adapt to the interaction patterns characteristic of the Claude-Agent-SDK ecosystem.** Conversely, Kimi-K2-Thinking performs best on our custom scaffold, experiencing significant degradation when migrated to external SDKs (dropping by 12.8% on the OpenAI-Agents-SDK). These findings underscore that agentic performance is not solely a model-intrinsic property but a result of the coupling between the model and its agentic scaffold.

## 5 Conclusion

In this work, we introduced AGENCYBENCH, a comprehensive evaluation framework designed to rigorously assess the frontiers of autonomous agents in long-horizon, real-world contexts. By synthesizing 138 authentic tasks across 32 diverse scenarios—requiring an average of 1 million tokens and 90 tool calls—we bridge the gap between existing toy benchmarks and the complexities of actual economic production. To ensure scalability and reproducibility, we developed a unified automated evaluation pipeline leveraging user simulation agents and Docker-based remote sandboxes. Our extensive evaluation reveals that while proprietary models currently lead in complex reasoning and self-correction, the gap between closed-source and open-source models remains significant. Even the most advanced models struggle to fully master long-horizon autonomy without substantial resource consumption, highlighting the need for improved efficiency. Furthermore, our analysis of agentic scaffolds demonstrates that performance is highly sensitive to the interaction environment, with models often exhibiting a 'native advantage' within their proprietary ecosystems. AGENCYBENCH serves not only as a leaderboard but as a

diagnostic tool. We hope this benchmark will drive future research towards more resource-efficient, self-correcting, and scaffold-agnostic agents capable of genuine real-world utility.

## Limitations

**Model Selection Coverage** While AGENCYBENCH evaluates a diverse set of representative proprietary and open-source models, the landscape of Large Language Models evolves rapidly. Due to constraints on computational resources and budget, our evaluation cannot exhaustively cover every emerging model variant, intermediate checkpoint, or specialized fine-tune. Consequently, our findings reflect a snapshot of the current state-of-the-art and may not capture the full performance spectrum of models released subsequent to our experiments.

**Domain Specificity** Our benchmark focuses on high-complexity tasks within digital environments, such as game development, software engineering, web research, etc. While these scenarios encompass 32 real-world situations, they are confined to software-based agents operating within a computer interface. The current framework does not extend to embodied agents or tasks requiring physical world interaction (e.g., robotics), leaving the evaluation of such multimodal physical agency for future research.

## Ethical Considerations

**Human Subjects and Compensation** The construction of AGENCYBENCH involved surveys and data validation by human experts, specifically computer science researchers and developers. We strictly adhere to the ACL Code of Ethics regarding human participant research. All contributors were fully informed of the project's scope, their data was anonymized to protect privacy, and they were compensated at a rate significantly exceeding the local hourly minimum wage to ensure fair and ethical treatment.

**Safety and Usage** Given that our benchmark involves agents generating executable code and performing shell operations, there are inherent risks associated with autonomous execution. To mitigate this, all evaluations are strictly confined within isolated Docker containers (Remote Sandbox) with controlled network access, preventing any potential harm to host systems. We will release this benchmark to foster the development of reliable and safe autonomous agents and explicitly oppose the application of these capabilities for malicious purposes, such as automated cyber-attacks.

## References


[1] Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, et al. 2025. Are: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*.

[2] Anthropic. 2025a. claude-4.5. https://www.anthropic.com/news/claude-opus-4-5.

[3] Anthropic. 2025b. Claude-agent-sdk-python. https://github.com/anthropics/claude-agent-sdk-python.

[4] Anthropic. 2025c. claude code. https://claude.com/product/claude-code.

[5] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. $\tau^2$-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

[6] Cursor. 2025. cursor-cli. https://cursor.com/cn/cli.

[7] DeepMind. 2025. gemini-3. https://deepmind.google/models/gemini/pro/.

[8] DesignArena Team. 2025. Designarena. https://www.designarena.ai/.

[9] GLM Team. 2025. glm-4.6. https://x.ai/news/grok-4-1.

[10] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

[11] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. 2025. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*.

[12] Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. 2025a. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *arXiv preprint arXiv:2510.25726*.

[13] Keyu Li, Mohan Jiang, Dayuan Fu, Yunze Wu, Xiangkun Hu, Dequan Wang, and Pengfei Liu. 2025b. Datasetresearch: Benchmarking agent systems for demand-driven dataset discovery. *arXiv preprint arXiv:2508.06960*.

[14] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. 2025. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

[15] Haotian Luo, Huaisong Zhang, Xuelin Zhang, Haoyu Wang, Zeyu Qin, Wenjie Lu, Guozheng Ma, Haiying He, Yingsha Xie, Qiyang Zhou, et al. 2025. Ultrahorizon: Benchmarking agent capabilities in ultra long-horizon scenarios. *arXiv preprint arXiv:2509.21766*.

[16] Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. Swe-lancer: Can frontier llms earn 1 million from real-world freelance software engineering?, 2025. *URL https://arxiv. org/abs/2502.12115*.

[17] OpenAI. 2025a. codex. https://openai.com/zh-Hans-CN/codex/.

[18] OpenAI. 2025b. gpt-5.2. https://cdn.openai.com/pdf/3a4153c8-c748-4b71-8e31-aecbde944f8d/oai_5_2_system-card.pdf.

[19] OpenAI. 2025c. Openai-agents-sdk. https://github.com/openai/openai-agents-python.

[20] OpenRouter. 2025. State of ai: An empirical 100 trillion token study with openrouter. https://openrouter.ai/state-of-ai.

[21] Melissa Z Pan, Negar Arabzadeh, Riccardo Cogo, Yuxuan Zhu, Alexander Xiong, Lakshya A Agrawal, Huanzhi Mao, Emma Shen, Sid Pallerla, Liana Patel, et al. 2025. Measuring agents in production. *arXiv preprint arXiv:2512.04123*.

[22] Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simón Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. 2025. Gdpval: Evaluating ai model performance on real-world economically valuable tasks. *arXiv preprint arXiv:2510.04374*.

[23] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

[24] Terminal-bench Team. 2025. terminal-bench. https://www.tbench.ai/.

[25] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. 2024. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

[26] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

[27] Yunze Wu, Dayuan Fu, Weiye Si, Zhen Huang, Mohan Jiang, Keyu Li, Shijie Xia, Jie Sun, Tianze Xu, Xiangkun Hu, et al. 2025a. Innovatorbench: Evaluating agents' ability to conduct innovative llm research. *arXiv preprint arXiv:2510.27598*.

[28] Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, et al. 2025b. Mcpmark: A benchmark for stress-testing realistic and comprehensive mcp use. *arXiv preprint arXiv:2509.24002*.

[29] xAI. 2025. grok-4.1. https://x.ai/news/grok-4-1.

[30] Yang Xiao, Mohan Jiang, Jie Sun, Keyu Li, Jifan Lin, Yumin Zhuang, Ji Zeng, Shijie Xia, Qishuo Hua, Xuefeng Li, et al. 2025a. Limi: Less is more for agency. *arXiv preprint arXiv:2509.17567*.

[31] Yang Xiao, Jiashuo Wang, Qiancheng Xu, Changhe Song, Chunpu Xu, Yi Cheng, Wenjie Li, and Pengfei Liu. 2025b. Towards dynamic theory of mind: Evaluating llm adaptation to temporal evolution of human states. *arXiv preprint arXiv:2505.17663*.

[32] Yang Xiao, Jiashuo Wang, Ruifeng Yuan, Chunpu Xu, Kaishuai Xu, Wenjie Li, and Pengfei Liu. 2025c. Limopro: Reasoning refinement for efficient and effective test-time scaling. *arXiv preprint arXiv:2505.19187*.

[33] Yang Xiao, Chunpu Xu, Ruifeng Yuan, Jiashuo Wang, Wenjie Li, and Pengfei Liu. 2025d. Scale: Selective resource allocation for overcoming performance bottlenecks in mathematical test-time scaling. *arXiv preprint arXiv:2512.00466*.

[34] Tianze Xu, Pengrui Lu, Lyumanshan Ye, Xiangkun Hu, and Pengfei Liu. 2025. Researcherbench: Evaluating deep ai research systems on the frontiers of scientific inquiry. *arXiv preprint arXiv:2507.16280*.

[35] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

[36] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

[37] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

# A Appendix

| | Scenarios | Tasks |
|---|---|---|
| Game | 10 | 50 |
| Front-end | 3 | 15 |
| Back-end | 3 | 15 |
| Code | 9 | 29 |
| Research | 5 | 19 |
| MCP | 2 | 10 |
| Total | 32 | 138 |

Table 5: Distribution of scenarios and tasks across the six core agentic capabilities in AGENCYBENCH.

In this appendix, we provide supplementary details to support the main findings of AGENCYBENCH. We first present the detailed statistics of the dataset composition. We then provide a granular analysis of tool usage frequency across different models. Following the statistical data, we detail the specific prompts used for our Text-based Judge, Vision-based Judge, and User Simulation Agent. Finally, we provide a concrete examples of scenario of game development to illustrate the multi-turn and long-horizon nature of the tasks.

## A.1 Dataset and Tool Statistics

In this section, we provide a comprehensive quantitative assessment of the AGENCYBENCH dataset composition and a granular analysis of model-specific tool usage behaviors. These statistics not only validate the diversity of the benchmark but also reveal distinct 'cognitive styles' across different LLMs.

**Dataset Composition and Domain Diversity** The structural distribution of AGENCYBENCH, as detailed in Table 5, encompasses 32 distinct scenarios and 138 specific tasks. To ensure the benchmark evaluates a broad spectrum of agentic capabilities, tasks are categorized into six core agentic capabilities. The **Game Development** domain constitutes the largest segment, accounting for approximately 36.2% of the total tasks (50 tasks across 10 scenarios). This heavy weighting reflects the unique complexity of game environments, which require agents to manage continuous state, simulate physics, and handle complex logic simultaneously. The **Code** domain follows with 29 tasks, focusing on algorithmic purity. Notably, the dataset explicitly balances full-stack development skills with an equal split between **Front-end** and **Back-end** tasks (15 tasks each). Furthermore, we include 10 tasks dedicated to the emerging **Model Context Protocol (MCP)**, ensuring the benchmark remains relevant to cutting-edge agent interface standards.

**Tool Usage and Behavioral Fingerprints** Perhaps the most revealing insights come from the tool usage frequency analysis presented in Table 6. We identified three distinct behavioral archetypes:

1. **The "Navigators" vs. The "Executors":** There is a striking divergence in how models orient themselves. GLM-4.6 exhibits a unique "navigator" strategy, invoking `list_directory` 158 times—nearly triple the average of other models. This indicates a strong preference for gathering environmental context before taking action. Conversely, GPT-5.2 and Claude-4.5-Sonnet act as "executors," prioritizing the `run_shell_command` tool (425 and 362 invocations, respectively) to empirically test code and run scripts, rather than passively observing the file structure.

2. **Editing Styles: "Surgeons" vs. "Rewriters":** The data reveals a fundamental difference in code modification philosophies. GPT-5.2 acts as a "surgeon," heavily utilizing the `replace` tool (146 invocations) to make precise, localized edits to existing files. In sharp contrast, GLM-4.6 overwhelmingly prefers the `write_file` tool (381 invocations), suggesting a tendency to overwrite entire files rather than attempting risky partial edits. While the "rewrite" strategy is safer, it is significantly less token-efficient.

3. **Memory Utilization:** Gemini-3-Pro stands out as the sole model to effectively leverage long-term memory capabilities. It is the only model to record significant usage of `update_memory_bank` (22 times) and `initialize_memory_bank` (7 times). While other models rely entirely on their context window, Gemini attempts to persist state and key information externally, a behavior that theoretically scales better for long-horizon tasks.

4. **Information Retrieval:** For external knowledge acquisition, GLM-4.6 again shows a distinct profile, using `web_fetch` 96 times, whereas models like Claude-4.5-Opus and GPT-5.2 rely more on their internal knowledge or specific search queries (`search_file_content`).

| | Claude-4.5-O | Claude-4.5-S | Gemini-3 | GPT-5.2 | Grok-4.1 | GLM-4.6 | Deepseek-V3.2 | Qwen3 | Kimi-K2 |
|---|---|---|---|---|---|---|---|---|---|
| agent_tool | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| get_database_name | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 0 |
| glob | 3 | 0 | 0 | 19 | 4 | 1 | 4 | 0 | 2 |
| initialize_memory_bank | 0 | 0 | 7 | 0 | 1 | 0 | 0 | 2 | 0 |
| list_directory | 53 | 47 | 54 | 45 | 42 | 158 | 25 | 8 | 82 |
| read_file | 55 | 106 | 86 | 147 | 67 | 142 | 41 | 6 | 147 |
| read_many_files | 16 | 0 | 24 | 13 | 1 | 0 | 1 | 0 | 0 |
| read_memory_bank | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| replace | 13 | 29 | 42 | 146 | 1 | 15 | 14 | 2 | 49 |
| run_shell_command | 191 | 362 | 140 | 425 | 141 | 371 | 86 | 40 | 301 |
| save_memory | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| search_file_content | 0 | 1 | 0 | 37 | 0 | 0 | 4 | 0 | 20 |
| deep_research | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 5 |
| hybrid_search | 1 | 4 | 0 | 0 | 7 | 1 | 9 | 0 | 3 |
| web_fetch | 0 | 0 | 0 | 0 | 6 | 96 | 0 | 0 | 7 |
| web_search | 7 | 6 | 5 | 12 | 25 | 4 | 0 | 1 | 6 |
| todo_write | 0 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | 0 |
| update_memory_bank | 0 | 0 | 22 | 0 | 1 | 0 | 0 | 5 | 0 |
| write_file | 81 | 169 | 79 | 117 | 109 | 381 | 41 | 185 | 97 |

Table 6: **Frequency of Tool Invocations Across Different Models.** Distinct behavioral patterns are observed, such as high shell usage by Claude/GPT and specific memory tool usage by Gemini. Claude-4.5-O denotes Claude-4.5-Opus; Claude-4.5-S denotes Claude-4.5-Sonnet; Gemini-3 denotes Gemini-3-Pro; Grok-4.1 denotes Grok-4.1-Fast; Qwen3 denotes Qwen3-235B-A22B-Thinking; Kimi-K2 denotes Kimi-K2-Thinking.

### A.2 Evaluation Prompts

To ensure rigorous, reproducible, and automated evaluation, we designed specialized prompts for three distinct agentic roles: the **Text-based Judge**, the **Vision-based Judge**, and the **User Simulation Agent**. The prompts are structured to enforce strict adherence to evaluation rubrics and minimize subjective variance. The following boxes illustrate the finalized prompts used in our framework.

**Evaluation Roles & Prompts**

**# Agent 1: Text-based Judge**

**Role:** You act as a Senior Code Compliance Auditor and Quality Assurance Specialist. Your objective is to systematically evaluate software deliverables against a strict set of functional and non-functional requirements.

**Input Data:**

- {`Code Files`}: A dictionary containing filenames and source code content.
- {`Task ID`}: A unique identifier for the specific engineering task.
- {`Rubrics`}: A list of constraints, functional requirements, and acceptance criteria.

**Evaluation Protocol:** 1. **Static Analysis:** Examine the logical structure, syntax validity, and dependency management of the {`Code Files`}. 2. **Requirement Mapping:** Verify the implementation of every item listed in {`Rubrics`}. 3. **Defect Identification:** Detect logical errors, missing functionalities, or violations of best practices.

**Output Specification:** Return a single valid JSON object with the following keys:

- `score` (Integer, 0-10):
  - **0-2 (Critical Failure):** Code is non-functional or fails to address the core problem definition.
  - **3-5 (Substantial Deficiency):** Major features are missing; code executes but fails significant rubrics.
  - **6-7 (Marginal Acceptance):** Core functionality is operational, but edge cases are unhandled or minor constraints are ignored.

  - **8-9 (High Compliance):** Meets all functional requirements with high code quality; only trivial stylistic issues remain.
  - **10 (Full Specification Alignment):** Flawless execution adhering to all rubrics and robustness standards.
- `confidence` (Float, 0.0-1.0): Assessment of certainty based on the evidence available in the static code.
- `comment` (String): A concise technical justification. You must explicitly reference specific files or lines of code when identifying failures.

**Example Output:**

```
{
  "score": 6,
  "confidence": 0.9,
  "comment": "The implementation correctly handles..."
}
```

---

## # Agent 2: Vision-based Judge

**Role:** You act as a Visual Grounding and UI/UX Verification Specialist. Your objective is to validate system functionality and design fidelity based strictly on visual evidence.
**Input Data:**

- {`Visual Assets`}: Screenshots or video frames capturing the system execution.
- {`Task ID`}: Identifier for the visual verification task.
- {`Rubrics`}: Visual, functional, and aesthetic criteria required for acceptance.

**Evaluation Protocol:** 1. **Visual Inspection:** Analyze {`Visual Assets`} for UI elements, layout consistency, and text rendering. 2. **Evidence Corroboration:** Cross-reference visual features against {`Rubrics`}. 3. **Strict Verification:** Any feature not explicitly visible in the assets must be marked as "Not Demonstrated."
**Output Specification:** Return a single valid JSON object with the following keys:

- `score` (Integer, 0-10):
  - **0-2 (No Evidence):** Visual assets are missing, irrelevant, or show a broken system.
  - **3-5 (Significant Deviation):** UI loads but fails to demonstrate key interactions or diverges significantly from design specs.
  - **6-7 (Partial Compliance):** Primary elements are functional; secondary visual polish or specific UI states are missing.
  - **8-9 (High Fidelity):** Visually correct and functional; clear evidence exists for almost all rubrics.
  - **10 (Pixel-Aligned Compliance):** Visual output exactly matches all descriptions and requirements.
- `confidence` (Float, 0.0-1.0): Reflects the clarity and completeness of the visual evidence.
- `comment` (String): A detailed justification. Explicitly state which rubrics were satisfied or failed based on visual artifacts.

**Example Output:**

```
{
  "score": 4,
  "confidence": 0.8,
  "comment": "Layout matches the wireframe. However, ..."
}
```

## # Agent 3: User Simulation Agent

**Role:** You act as an Acceptance Testing Supervisor and Feedback Generator. Your objective is to provide actionable, structured feedback for iterative refinement when a submission fails to meet the acceptance threshold.
**Input Data:**

- {Evaluation Result}: The JSON output from the Judge (Score, Confidence, Comment).
- {Threshold}: The minimum passing score.
- {Rubrics}: The original requirement list.
- {Deliverables}: The submitted code or text.
- {Artifacts}: Visual outputs (if any).

**Task:** Perform a Gap Analysis between the {Deliverables} and the {Rubrics} based on the {Evaluation Result}. Isolate specific discrepancies that caused the score to fall below the {Threshold}.
**Output Specification:** Generate a structured textual report (not JSON) containing: 1. **Status Declaration:** State clearly that the submission is rejected due to the score. 2. **Failure Diagnosis:** deeply analyze the evaluation comments to list exactly which rubrics were not met. 3. **Root Cause Analysis:** Explain the failure from a user requirement perspective (e.g., "The requirement specified a responsive layout, but the provided CSS uses fixed width"). 4. **Directives for Revision:** Provide explicit instructions on what must be rectified in the next iteration.
**Tone:** Constructive, objective, and directive.
**Example Output Pattern:** "Submission Rejected (Score: 5/10). The following critical rubrics were not met: 1. [Rubric Name]: The parser crashes on nested keys. 2. [Rubric Name]: The 'Submit' button is visually missing. Action Required: Implement recursive parsing logic and ensure the footer component renders correctly."

### A.3 Scenario Example

We present the complete definitions for scenario: developing a Gomoku game. The example highlight the hierarchical structure of tasks, explicit deliverables, and rubrics.

## Example 1: Game Development (Gomoku)

### Task 1: Static Board Initialization

**Query:** Create a static Gomoku launch view that renders a centered 15×15 board with labelled axes (A–O and 1–15), a prominent '#start-btn' labelled 'Start Match', and a legend explaining black and white stones. Automatically call 'window.app.initializeBoard()' on load so the markup appears without manual interaction.
**Deliverables:** 'index.html' that links 'styles.css' and 'app.js' via relative paths and contains containers '#board', '#legend', and '#start-btn'.- 'styles.css' defining a 640px square grid, positioning the legend beneath the board, and styling the start button.- 'app.js' exposing 'window.app' with methods 'initializeBoard()' and 'describeLayout()'; the latter returns an object with keys 'startButton', 'legend', and 'cells', where 'cells' maps coordinates like 'Ĥ8¨ to ' ẍ¨ number, ÿ¨ number ' viewport positions.
**Rubric:** Visual grid: automation captures 'layout.png' and confirms the board is exactly 640±4px wide and tall, centered within ±6px both horizontally and vertically, and decorated with 15×15 intersection markers.- Controls placement: 'describeLayout()' must report 'startButton' roughly at ' ẍ¨ 96±6, ÿ¨ 96±6 ' relative to the viewport and ensure the legend top edge sits within 80±6px of the board bottom.- DOM contract: the page must render 225 elements with class '.intersection' each carrying 'data-cell=¡letter¿¡number¿¨', alongside elements '#board', '#legend', and '#start-btn'.- API bootstrapping: invoking 'window.app.initializeBoard()' twice must be idempotent, leaving exactly one board instance and

a populated description payload.

### Task 2: Interactive Game Logic

**Query:** Extend the board so players alternate black and white stones when clicking intersections, starting with black. Display a status line inside ‘#status-bar‘ and highlight the latest stone with a pulsing ring. Keep all assets from task1.

**Deliverables:** Continue shipping ‘index.html‘, ‘styles.css‘, and ‘app.js‘, updating the markup to include ‘#status-bar‘ beneath the legend.- Extend ‘window.app‘ with ‘debugState()‘ (returning move history and the current player) and ‘setDemoMoves(sequence)‘ where ‘sequence‘ is an array of ‘ ”coord”: ”H8”, ”color”: ”black” ‘ objects for automation to preload moves.

**Rubric:** Turn order: automation clicks the coordinates reported by describeLayout() corresponding to ‘H8‘, ‘H9‘, and then attempts ‘H9‘ again. ‘debugState().moves‘ must record alternating colors for the first two moves and refuse the third with an unchanged move list.- Last move halo: after the second valid move the element ‘#last-move-indicator‘ must surround the latest stone with a 26±3px animated ring captured in ‘moves_turns.webm‘.- Status updates: ‘#status-bar‘ text must read ‘Black to move‘ at load, switch to ‘White to move‘ after the first click, then revert to ‘Black to move‘ when the duplicated coordinate is rejected.- Layout regression: the screenshot and describeLayout() metrics from task1 must remain within the same tolerances.

### Task 3: Victory & Replay System

**Query:** Introduce automatic victory detection, a banner announcing the winner, undo/redo controls, and a replay feature that animates the finished game. Preserve the interactive flow from task2.

**Deliverables:** Maintain the three primary files and append a ‘session.log‘ file storing comma-separated history (‘timestamp,player,coord‘).- Add a toolbar ‘#controls‘ containing buttons ‘button[data-action=ündo¨]‘, ‘button[data-action=r̈edo¨]‘, and ‘button[data-action=r̈eplay¨]‘ displayed in that order beneath the status bar.- Extend ‘window.app‘ with methods ‘undo()‘, ‘redo()‘, ‘startReplay()‘, ‘checkWinner()‘, and ‘exportLog()‘ returning the log contents.

**Rubric:** Win scenario: automation plays the sequence ‘H8, H9, I8, I9, J8, J9, K8, K9, L8‘ (black begins). After the final move ‘checkWinner()‘ must return ‘Black‘, ‘#winner-banner‘ must display ‘Black wins¡, and further manual clicks must be disabled. Replay controls: activating the toolbar buttons (undo → redo → replay) must modify the board accordingly while replay.webm shows stones animating in chronological order at more than 300ms per move. During replay manual clicks remain blocked until completion.- Logging: ‘session.log‘ must append one line per move plus replay markers (if any). ‘exportLog()‘ must match the file content exactly.- Continuity: turn enforcement, last move highlighting, and layout expectations from tasks 1–2 must still hold.

### Task 4: Persistence Layer

**Query:** Add a persistence drawer that lets reviewers save and restore finished games without losing the log or replay tools. Introduce a panel ‘#persistence‘ beneath the controls containing buttons ‘#save-game‘, ‘#load-game‘, and ‘#reset-match‘ plus a read-only ‘¡textarea id=s̈tate-json¿‘. Saving writes the current match state to the textarea and ‘localStorage[’gomoku-state’]‘. Loading parses the textarea and rehydrates board, log, and indicators. Reset clears the board but retains totals for wins/draws shown inside ‘#scoreboard‘ with counters ‘.black-wins‘, ‘.white-wins‘, ‘.draws‘.

**Deliverables:** Continue shipping the three primary files. The new panel must sit below ‘#controls‘ and adopt responsive styling that matches the existing layout.- Extend ‘window.app‘ with ‘serializeState()‘, ‘applyState(serialized)‘, ‘resetMatch()‘, and ‘getScoreboard()‘ returning ‘ black: number, white: number, draws: number ‘. Persist the latest save so reloading the page reenacts the saved board automatically.

**Rubric:** Save flow: after several moves, ‘#save-game‘ updates ‘#state-json‘ with JSON containing a ‘moves‘ array and writes the same payload to ‘localStorage[’gomoku-state’]‘. ‘serializeState()‘ must return the identical data.- Restore flow: invoking ‘resetMatch()‘ empties the board while keeping scoreboard totals. Calling ‘applyState()‘ with the previously saved JSON must rebuild stones, last-move halo, and ‘debugState()‘ history exactly.- Scoreboard: when ‘checkWinner()‘ declares a winner followed by ‘resetMatch()‘, ‘getScoreboard()‘ increments the winner count without altering the opponent tally. Visual labels in ‘#scoreboard‘ must reflect the same totals.- Regression: undo/redo/replay, logging, and layout tolerances from tasks 1–3 remain satisfied.

### Task 5: Diagnostics & Stress Testing

**Query:** Layer a diagnostics sidebar and scripted scenarios to stress-test persistence features. Add a toggle button ‘#toggle-diagnostics‘ that slides a right-aligned ‘aside id=”diagnostics”‘ into view displaying current depth, total moves, elapsed milliseconds, and a table with headers ‘Metric‘ and ‘Value‘. Load scenario data from ‘hard_cases.json‘ containing at least two entries with ‘id‘, ‘label‘, ‘moves‘, and ‘winner‘ fields. Provide playback controls within the sidebar to preview scenarios and inspect metrics.

**Deliverables:** Keep existing files and supply ‘hard_cases.json‘. Sidebar must reveal aggregated stats for the active game and scenario preview (e.g., longest line, capture streaks) in a structured list.- Extend ‘window.app‘ with ‘loadScenario(id)‘, ‘playScenario(id, options)‘, ‘getDiagnostics()‘, ‘summarizeScenario(id)‘, and ‘estimateHeap()‘. ‘playScenario‘ should return a Promise resolving after the animation completes while blocking manual clicks.

**Rubric:** Scenario import: calling ‘loadScenario(0)‘ must position stones per the fixture, update the banner using ‘winner‘, and log the moves. ‘summarizeScenario(0)‘ returns an object containing total stones and the longest contiguous run for each color.- Playback: invoking ‘playScenario(1, { intervalMs: 220 })‘ replays the scenario in order with more than 200ms spacing, reusing undo/redo state guards and re-enabling manual clicks afterward.- Diagnostics: after either scenario, ‘getDiagnostics()‘ supplies keys ‘depth‘, ‘elapsedMs‘, ‘nodes‘, and ‘topLines‘ (array). While the sidebar is visible, the table lists these values and stays 260±6px wide at 60±6px from the right edge, captured in ‘diagnostics.png‘.- Stability: calling ‘estimateHeap()‘ five times in succession yields non-decreasing integers less than 64000000. All expectations from tasks 1–4 continue to hold.