COCOABENCH: Evaluating unified digital agents in the wild

Shibo Hao[1,*,†], Zhining Zhang[1,*], Zhiqi Liang[1,*], Tianyang Liu[1,*], Yuheng Zha[1], Qiyue Gao[1], Jixuan Chen[1], Zilong Wang[1], Zhoujun Cheng[1], Haoxiang Zhang[1], Junli Wang[1], Hexi Jin[1], Boyuan Zheng[1], Kun Zhou[1], Yu Wang[1], Feng Yao[1], Licheng Liu[2], Yijiang Li[1], Zhifei Li[3], Zhengtao Han[3], Pracha Promthaw[1], Tommaso Cerruti[4], Xiaohan Fu[5], Ziqiao Ma[6], Jingbo Shang[1], Lianhui Qin[1], Julian McAuley[1], Eric P. Xing[7], Zhengzhong Liu[7], Rupesh Kumar Srivastava[7], Zhiting Hu[1]

[1]UC San Diego; [2]University of Cambridge; [3]UC Berkeley; [4]ETH Zurich; [5]Gray Swan AI; [6]University of Michigan; [7]MBZUAI IFM

[†]Corresponding author: s5hao@ucsd.edu

Abstract

LLM agents now perform strongly in software engineering, deep research, GUI automation, and various other applications, while recent agent scaffolds and models are increasingly integrating these capabilities into unified systems. Yet, most evaluations still test these capabilities in isolation, which leaves a gap for more diverse use cases that require agents to combine different capabilities. We introduce COCOABENCH, a benchmark for unified digital agents built from human-designed, long-horizon tasks that require flexible composition of **vision**, **search**, and **coding**. Tasks are specified only by an instruction and an automatic evaluation function over the final output, enabling reliable and scalable evaluation across diverse agent infrastructures. We also present COCOA-AGENT, a lightweight shared scaffold for controlled comparison across model backbones. Experiments show that current agents remain far from reliable on COCOABENCH, with the best evaluated system achieving only 45.1% success rate. Our analysis further points to substantial room for improvement in reasoning and planning, tool use and execution, and visual grounding.[1]

1 Introduction

LLM agents are showing strong potential across an expanding set of domains, including software engineering (Yang et al., 2024), GUI automation (Wang et al., 2025c), and report generation with deep research (OpenAI, 2025a). Recent agentic frameworks, e.g., OpenClaw (OpenClaw, 2026) and Claude Cowork (Anthropic, 2026b), as well as models, e.g., GPT-5.4 (OpenAI, 2026), Claude Sonnet 4.6 (Anthropic, 2026a), and Seed-2.0 (ByteDance Seed, 2026), aim to unify these capabilities into a single system, moving toward a unified digital agent that can assist humans with complex tasks. However, existing benchmarks still largely focus on a single domain or a single interaction mode (e.g., CLI-only (Jimenez et al., 2024), GUI-only (Xie et al., 2024), or predefined tool APIs (Li et al., 2026)), making them insufficient for systematically evaluating unified agent capabilities on diverse tasks in open environments.

COCOABENCH is designed to evaluate general purpose digital agents on complex tasks that require composing multiple core capabilities. We focus on three fundamental capabilities that are essential for a strong digital agent: (1) **coding** (or, more broadly, terminal use), which enables code-based problem solving, supports quantitative analysis, and allows agents to invoke structured tools and APIs; (2) **search**, which enables information seeking, navigation, and synthesis across online sources; and (3) **vision**, which enables agents to interpret visual

[*]Equal contribution.

[1]Project page: https://cocoabench.github.io/

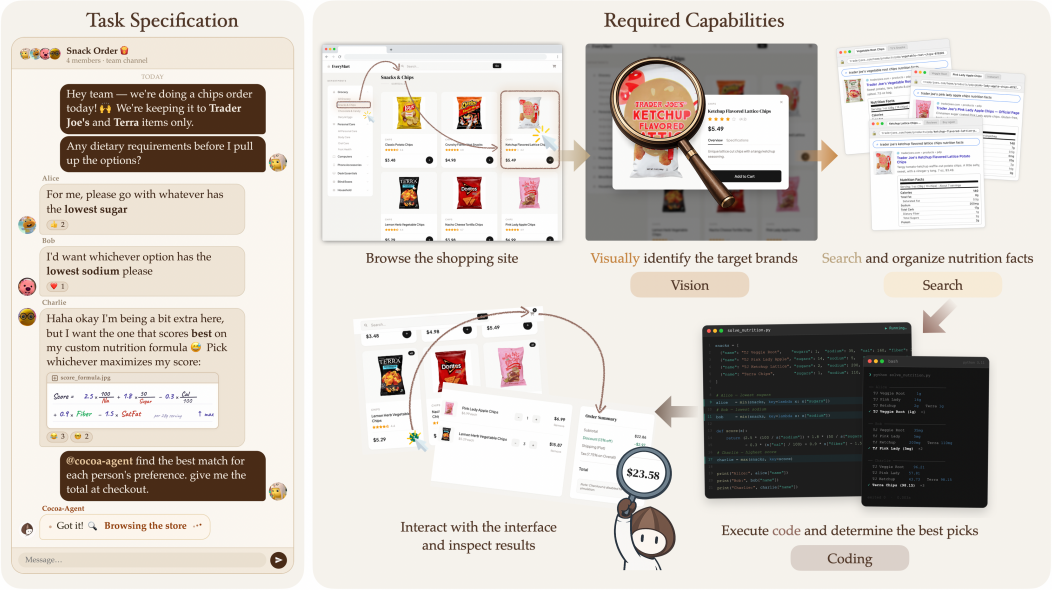

Figure 1: COCOABENCH evaluates agents on complex digital tasks that require flexible composition of core capabilities such as vision, search, coding. The shopping example shown here illustrates one such task and highlights the multi step, compositional nature of the benchmark.

inputs and interact with GUIs. Beyond mastering each capability in isolation, a strong agent must also *plan effectively and compose these capabilities adaptively* to achieve a target goal (Figure 1).

COCOABENCH tasks are specified minimally by an instruction and an evaluation function over the agent's final output, without being tied to a particular runtime, interface, or tool ecosystem. This design keeps the benchmark agnostic to specific agent infrastructures and requires agents to reason about tool use in an open world setting, rather than operate within pre-specified apps. To make evaluation reliable without sacrificing task complexity, we equip each task with an evaluation script, without relying on LLM judges or human evaluation. For action centric tasks, where correctness depends on multi-step interaction, we design outcome based proxy evaluators whose success strongly implies correct execution. This process to outcome transformation preserves open ended workflows while keeping evaluation reproducible and scalable.

We evaluate COCOABENCH in two settings: (1) using existing agent products as complete systems, and (2) under COCOA-AGENT, a lightweight shared scaffold that enables more controlled comparison across backbone models. Our experiment shows that the best-performing agent (GPT-5.4 under Codex) achieves a success rate of only 45.1%, and leading open-source models such as Kimi-k2.5 and Qwen3.5 reach only 11.8% and 9.8% respectively, highlighting significant room for improvement in current agent capabilities. We also find that scaffold design plays an important role. Coding-oriented scaffolds such as Codex and Claude Code generalize well beyond their original domain, serving as effective task solvers on COCOABENCH. Analysis of tool usage reveals that top-performing models allocate more of their actions to code execution, indicating that programmatic processing is an effective strategy for the multi-step reasoning and structured output formatting that COCOABENCH tasks demand. Our error analysis shows that current systems remain unreliable along three key dimensions of unified digital agency: reasoning and planning, tool interaction and execution, and visual grounding. We open-source COCOABENCH, including all task instructions, evaluation scripts, and the full implementation of COCOA-AGENT scaffold, to facilitate reproducible evaluation and future research on general purpose digital agents.

Benchmark	Application focus	Infra coupling	Verif. reward	Capabilities V	S	C
OSWorld	Computer use	VM + task setup	✓	✓	✗	✗
BrowseComp	Web research	Open	✓	✗	✓	✗
WebArena	Web interaction	Self-hosted sites	✓	✗	✓	✗
SWE-bench Pro	Repository-issue resolution	Repo container	✓	✗	✗	✓
TerminalBench-2	CLI task execution	Task container	✓	✗	✗	✓
MCP Atlas	MCP server orchestration	Fixed apps	✓	✗	✗	✓
ToolDecathlon	Cross-app tool using	Fixed apps	✓	✗	✗	✓
GDPval	Occupation-grounded tasks	Open	✗	✓	✓	✓
CocoaBench	Diverse digital tasks	Open	✓	✓	✓	✓

Table 1: Comparison of representative agent benchmarks by application focus, infrastructure coupling, reward verifiability, and core capability coverage (V=Vision, S=Search, C=Coding).

2 Related Work

2.1 Evaluating digital agents

As LLM-based agents expand from single-domain tools to general-purpose digital assistants, the need for comprehensive evaluation benchmarks has grown accordingly. Table 1 summarizes representative agent benchmarks that are widely used in recent frontier-model (e.g., Gemini-3.1 pro, GPT-5.4 and Claude-Opus-4.6) evaluations till March 2026. We compare these benchmarks along application focus, infrastructure coupling, reward verifiability, and required core capabilities (vision, search, and coding).

Existing benchmarks cover complementary slices of digital-agent evaluation. OSWorld (Xie et al., 2024) evaluates computer use in VM-based environments with task-specific setup. WebArena (Zhou et al., 2024) provides reproducible, self-hosted websites for evaluating functional task completion through web interaction, and VisualWebArena (Koh et al., 2024) extends this setting to tasks whose completion depends on visual grounding. WorkArena (Drouin et al., 2024) similarly studies browser-based knowledge work within the ServiceNow enterprise environment. These benchmarks enable controlled evaluation of GUI grounding, navigation, and computer-use control. Their success criteria, however, are generally implemented by inspecting the state of a particular website, application, or virtual machine, coupling the evaluation to the corresponding environment.

Other benchmarks isolate different capabilities or application domains. SWE-bench Pro (Deng et al., 2025) and TerminalBench-2 focus on repository issue resolution and CLI task execution, respectively. MCP Atlas (Bandi et al., 2026) and Tool Decathlon (Li et al., 2026) broaden tool-use coverage but operate over predefined tool or application interfaces. BrowseComp (Wei et al., 2025) evaluates difficult open-web information seeking, whereas GDPval (Patwardhan et al., 2025) emphasizes occupation-grounded deliverables and consequently relies on expert judgment. CocoaBench occupies a different point in this design space: it evaluates task-specific final outcomes without prescribing an agent runtime or execution path, while requiring agents to compose vision, search, and open-ended coding across diverse digital resources.

2.2 General Digital Agents

Large language model based agents have become capable of performing complex tasks across different digital environments, but existing systems typically operate within a single interaction modality. SWE-Agent (Yang et al., 2024) and OpenHands (Wang et al., 2025b) target software engineering, while Codex, Claude Code, and Terminus-2 (Merrill et al., 2026) operate in terminal environments. On the visual side, Aguvis (Xu et al., 2025), OpenCUA (Wang et al., 2025c), and UI-TARS (Wang et al., 2025a) enable agents to operate graphical interfaces through screenshot understanding and coordinate-based actions. Deep research agents (OpenAI, 2025a) address yet another axis, performing multi-step web search

and synthesis. While effective within their respective domains, these systems each rely on a single interaction modality and do not generalize across capability boundaries.

Recent systems such as OpenClaw (OpenClaw, 2026) and ChatGPT Agent (OpenAI, 2025b) aim to integrate browsing, coding, and visual interaction into a single agent, but systematic evaluation of such general-purpose agents remains challenging, as existing benchmarks typically assess only a subset of the required capabilities. COCOABENCH and COCOA-AGENT are designed to address this gap, providing tasks that explicitly require the composition of vision, search, and coding alongside a lightweight agent framework with integrated sandbox support for reproducible evaluation.

3 CocoaBench

3.1 Task construction

COCOABENCH consists of 153 human-authored tasks designed to evaluate unified agents on complex problem solving. We first identified practical scenarios in which agents are expected to provide assistance, covering research, entertainment, shopping, business, and other everyday tasks. For each scenario, we instantiated 3 to 5 concrete tasks to form the final COCOABENCH dataset. Task authors adhered to three key criteria:

- Each task should require the integration of multiple capabilities.
- Each task should pose a nontrivial challenge for humans in realistic settings.
- Dependencies on external resources should remain stable over time, so that task validity is not compromised by changes in third-party content.

Inclusive task settings. Unlike benchmarks that are tightly coupled to specific environments like OSWorld (Xie et al., 2024) or fixed tool ecosystems like Tool Decathlon (Li et al., 2026), each task in COCOABENCH is **minimally** defined and specified by an instruction and an evaluation function over agent outputs. For tasks requiring multimodal inputs or additional resources, we host the required assets online and include their URLs directly in the task instructions. This design makes the benchmark compatible with diverse agent infrastructures, including locally deployed ones, e.g., OpenClaw (OpenClaw, 2026), and the hosted or sandboxed ones, e.g., ChatGPT Agent Mode (OpenAI, 2026). It also allows tasks to be more diverse, rather than being constrained by specific environments or infrastructures. Furthermore, it evaluates whether agents can reason about which tools to use for each task in an open-world setting, instead of selecting only from a fixed, predefined toolset.

Automatic evaluation functions. Each task is paired with its own evaluation function, enabling automatic and reproducible assessment. Whenever possible, we require outputs in a unique structured format, e.g., `str`, `list`, or `dict`. In some real-world tasks, agents must take actions with an environment to complete the task, beyond answering questions. Directly verifying the action sequence, however, is often impractical. We therefore use proxy outcome verifiers based on automatically checkable end results, designed so that a correct outcome is unlikely without successful execution. For example, in the shopping task shown in Figure 1, we verify the final price returned by the agent, since obtaining the correct value typically requires both correct website interaction and correct reasoning over the user request. This enables scalable evaluation while retaining realistic and diverse task settings.

Quality control and shortcut prevention. All tasks, reference answers, and evaluation functions underwent multiple rounds of peer review. Reviewers checked that instructions were unambiguous, output formats were well defined, reference answers were correct, and the evaluator did not admit trivial shortcuts. Whenever possible, tasks require open-choice outputs with large answer spaces, for example, precise entity names, floating-point values, or nontrivial counts, rather than binary choices, which makes accidental matches and systematic gaming more difficult. Early versions of some tasks did expose unintended shortcuts, and these tasks were revised until at least two independent reviewers confirmed that the shortcut had been removed. We later converted it into a webpage backed by a

server, preventing agents from inspecting the source code and directly inferring the answer without performing the intended exploration. In addition, we map the sequence of actions an agent takes to a verification code. To obtain the correct code, the agent must execute the exact sequence of valid actions. Pilot runs were then used to distinguish agent failures from remaining task-design issues, and persistently ambiguous tasks were removed. Finally, a post-hoc audit of 70 successful trajectories found that 68 followed legitimate solution workflows, with one accidental evaluator false positive and one potentially memorized completion; full audit details are provided in Appendix E.

Resource stability. We consider long-term reproducibility at both construction and verification time. Self-hosted websites are served from fixed code, task documents are versioned in a dedicated repository, and third-party resources are restricted to long-lived content such as official videos and closed historical records. We additionally conducted a resource-level drift audit. Of the 153 tasks, 141 (92.2%) rely only on effectively immutable or versioned resources, while 12 (7.8%) involve resources with minor drift risk, such as externally hosted videos or historical versions of tax calculators. Appendix F describes the resource classes and audit criteria.

3.2 Task diversity and composition

Task domains. COCOABENCH consists of tasks spanning 9 diverse domains (Figure 2(a)), including Business, Culture, Education, Life, Logic & Puzzles, Science, Sports, Technology, and Travel. These scenarios closely mirror everyday challenges that can often be solved, or even optimally solved, given sufficient time and patience, through careful planning and deliberate navigation of task resources and the broader digital world.

Task resources. The tasks in COCOABENCH are supported by diverse resources, which are either carefully collected from the internet or provided by the task designers. These include webpages, videos, images, and documents in realistic environments. Notably, task designers hosted 17 websites and contributed artifacts such as Weights & Biases logs, their own ChatGPT conversations, and even a collection of Costco receipts, all of which help construct realistic and challenging tasks. We have made our best effort to ensure that these resources are easily acquired by anyone and remain stable. The distribution of resource types is shown in Figure 2 (b).

Target capabilities. We categorize the key capabilities required to solve each task into three main types: **Vision**, **Search**, and **Coding**. The primary labels are determined based on human annotation. As shown in Figure 2 (c), a task is labeled as *Vision* if visual information must be extracted to solve it correctly; *Search* if accessing and analyzing information from the internet is necessary; and *Coding* if writing code is considered important for solving the task efficiently and reliably. Notably, 98% of tasks require multiple capabilities, and their co-occurrence matrix is illustrated in Figure 3. Interestingly, although Coding is annotated as important for 56.2% of tasks, our later analysis in Section 5.3 shows that stronger agents rely on code execution even more broadly than expectation.

4 Experiment settings

4.1 Existing agentic systems

We evaluate representative agent systems on COCOABENCH to cover a range of agent designs and capability profiles. (1) **ChatGPT Agent Mode** (OpenAI, 2025b) is one of the earliest unified digital agents, with support for browsing, coding, and visual interaction in a sandbox environment. (2) **OpenClaw** (OpenClaw, 2026) is an open source framework for unified digital agency that can be deployed on personal computers. We instantiate it with GPT-5.4 thinking high (OpenAI, 2026) and Claude Sonnet 4.6 thinking high (Anthropic,

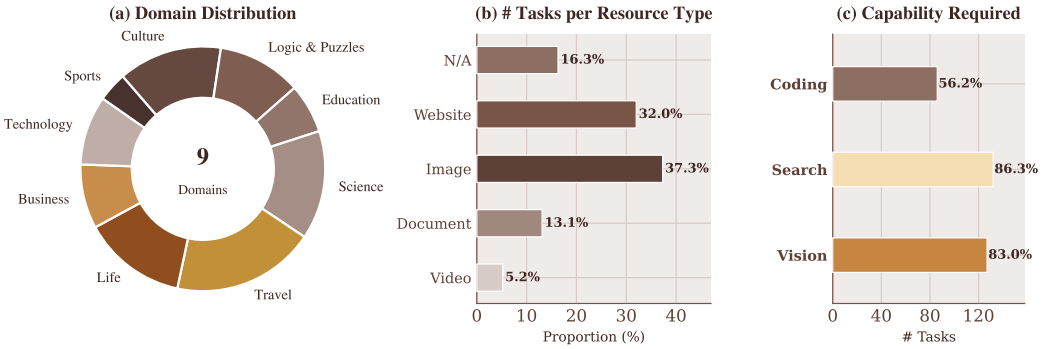

Figure 2: **Statistics of COCOABENCH.** (**a**) Distribution of task domains, covering a wide range of everyday topics. (**b**) Distribution of resource types required by the tasks. Documents include `.csv`, `.pdf`, and `.bibtex`. (**c**) Human-annotated key capabilities required for each task, including Vision, Search, and Coding. The Vision, Search, and Coding types are not mutually exclusive; numbers on the bars indicate the proportion of all tasks, and the x-axis represents the number of tasks.

2026a) as backbones. (3) **Codex**[2] and (4) **Claude Code**[3] are two representative coding agent products. We also use GPT-5.4 thinking high and Claude Sonnet 4.6 thinking high as the backbone model, respectively. (5) **OpenAI Deep Research** (OpenAI, 2025a) is included as a research-oriented agent for long horizon web information seeking and synthesis. We use the `o4-mini` version of it. Unless otherwise specified, each run uses a 30 minute wall clock budget with a maximum of 50 interaction turns.

4.2 COCOA-AGENT

Similar to the Bash-Only setting in SWE-Bench (Jimenez et al., 2024), although our full leaderboard compares arbitrary agentic systems, we also aim to compare the agentic capabilities of LLMs under a shared agent framework. We develop COCOA-AGENT, a unified agent scaffold that is intentionally lightweight and modular, allowing us to control agentic components and make comparisons more analytically interpretable. It is built on top of AIO Sandbox[4], an all-in-one sandbox runtime that integrates browser, shell, file system within a single Docker container. COCOA-AGENT adopts a ReAct-based scaffold, equipping model backbones with general purpose tools for browser interaction (both DOM-level APIs and screenshot-based GUI control), terminal execution, file manipulation, and code execution. With the integration of sandbox, it also enables safer execution and more scalable parallel evaluation. We also expect it to provide a practical foundation for future research on reinforcement learning for unified digital agents.

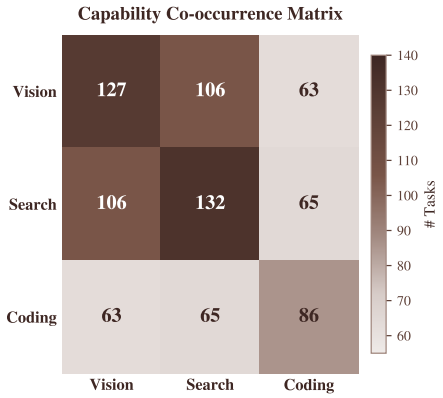

Figure 3: Co-occurrence matrix of required capabilities (Vision, Search, and Coding).

To evaluate model backbones under a consistent agent scaffold, we include: (1) **Claude Sonnet 4.6 (thinking high)**, (2) **GPT-5.4 (thinking high)**, (3) **Gemini-3.1-pro (thinking high)**, and (4) **Gemini-Flash-3.0**. We additionally include strong open-source multimodal

[2] `https://openai.com/codex/`
[3] `https://code.claude.com/docs/en/overview`
[4] `https://github.com/agent-infra/sandbox`

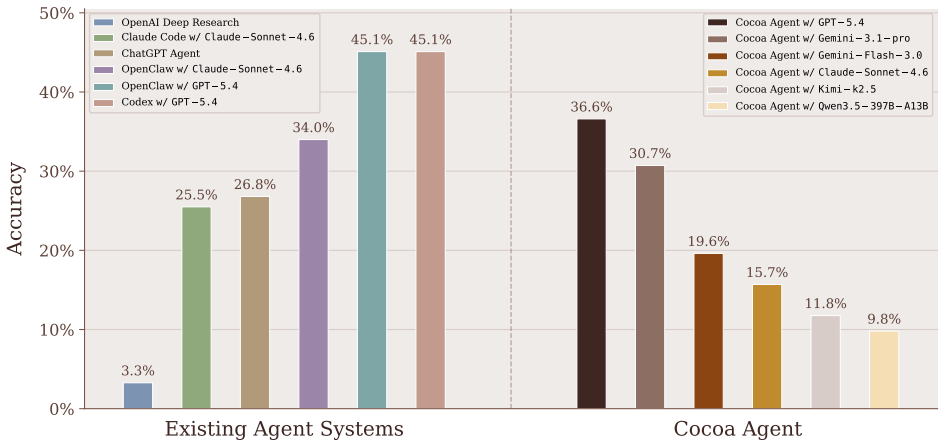

Figure 4: Overall performance on COCOABENCH for representative agent systems and model backbones under the shared COCOA-AGENT scaffold.

models: (5) **Kimi-k2.5** (Moonshot AI, 2026), an MoE model with 1T parameters and 32B active parameters, and (6) **Qwen3.5-397B-A13B** (Qwen Team, 2026), an MoE model with 397B parameters and 13B active parameters. Together, these choices cover both leading proprietary models and strong open-source multimodal alternatives.

5 Results and analysis

5.1 Overall results

We first report the main results (accuracy) on COCOABENCH for both representative existing agent systems and model backbones instantiated under COCOA-AGENT. Figure 4 shows two complementary views: the left compares complete agent systems in different scaffolds, while the right compares diverse backbones under a shared COCOA-AGENT scaffold.

From the model perspective, GPT 5.4 is the most consistently strong backbone across scaffolds. It attains 45.1% under both Codex and OpenClaw, and still reaches 36.6% under COCOA-AGENT, corresponding to the top three entries in the leaderboard. Claude Sonnet 4.6 can also be competitive, achieving 34.0% under OpenClaw, but its performance is less stable across other scaffolds, dropping to 25.5% in Claude Code and 15.7% in COCOA-AGENT. By contrast, the open source models remain clearly behind the leading proprietary models, with Kimi k2.5 and Qwen3.5 397B A13B reaching 11.8% and 9.8%, respectively. Overall, these results suggest that backbone quality still matters substantially, with GPT 5.4 standing out as the most robust model on COCOABENCH.

The agent scaffold also plays a crucial role. A notable finding is that scaffolds originally developed for coding, including CodeX and Claude Code, can already act as fairly general problem solvers on COCOABENCH. OpenClaw also appears to be a robust scaffold, yielding strong results with both GPT 5.4 and Claude Sonnet 4.6. While COCOA-AGENT is not the strongest-performing scaffold, it already attains reasonably strong performance with capable backbones, making base model comparisons meaningful. Given its simplicity and integrated sandbox, we believe it also serves as a promising baseline for future research on unified digital agents, including data engineering and reinforcement learning-based training.

5.2 Cost and model performance

We compare model performance against average cost and task completion time. The average cost per task ranges from \$0.5 to \$2.5, while average completion time ranges from 380s to 3400s. As shown in Figure 5, both comparisons show a consistent trend: CodeX achieves the best balance between cost efficiency and performance and lies on the Pareto frontier, while the other agents show no clear complementary advantages. This suggests that higher

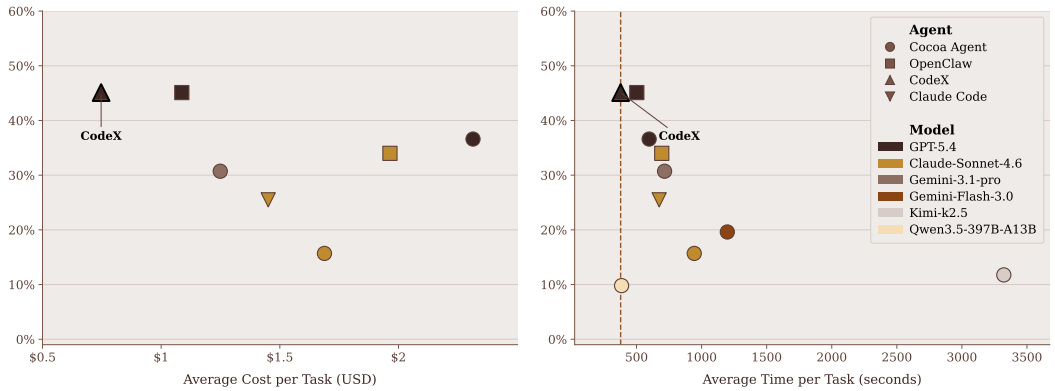

Figure 5: Accuracy-cost (left) and accuracy–time (right) trade-offs across agents. Marker shapes denote agent/scaffold types and colors denote base models.

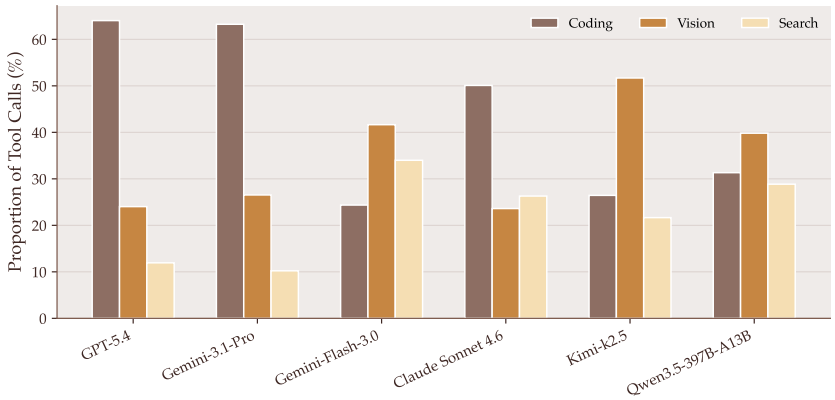

Figure 6: Per-model distribution of tool calls across the three capability categories under COCOA-AGENT.

monetary or time costs do not necessarily lead to better performance. For example, COCOA-AGENT w/ Qwen3.5-397B-A13B has a completion time comparable to that of CodeX but achieves 35.3% lower accuracy. Cost efficiency also depends strongly on the scaffolding design. Even with GPT-5.4 as the base model, Codex costs $0.75 per task, compared with $1.09 for OpenClaw and $2.31 for COCOA-AGENT.

5.3 Tool statistics

We further analyze the tool calls recorded under COCOA-AGENT across the six evaluated models to examine how different backbones utilize the available tools and compose core capabilities during task solving.

Tool call distribution. Figure 7 reports the aggregate call counts of the ten most frequently used tools across the 6 models under COCOA-AGENT. Coding tools dominate overall: `code_execute` and `shell_execute` together account for the largest share of total invocations, followed by browser level actions such as `browser_navigate` and `image_read`, with DOM-level interaction tools appearing at moderate frequency. This distribution re-

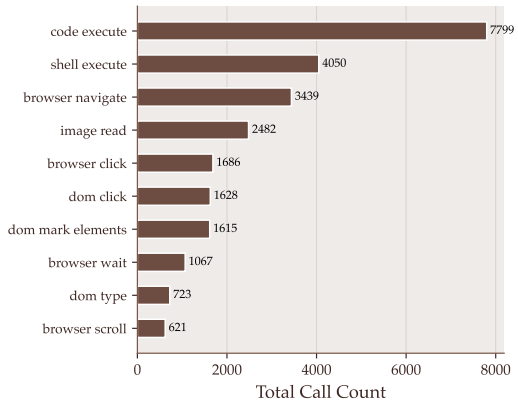

Figure 7: Top 10 most frequently used tools, ranked by total call count.

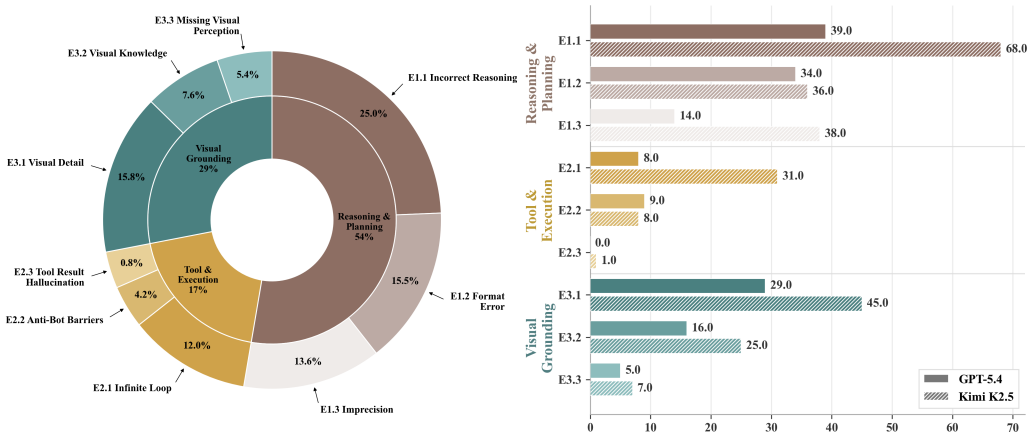

Figure 8: **Left:** Error type distribution across all six models evaluated on COCOABENCH using the COCOA-AGENT framework (based on 722 failure trajectories). **Right:** Comparisons of error distributions between GPT-5.4 and Kimi K2.5.

flects the compositional demands of COCOABENCH tasks, which generally require agents to both perceive and acquire information from diverse online sources and to process and synthesize that information into structured outputs. The prominence of coding tools suggests that programmatic execution is central to solving COCOABENCH tasks, providing a reliable approach for multi-step reasoning, data processing, and structured output formatting.

Tool usage and performance. To compare tool usage profiles across models, we map each tool to one of the three key capabilities defined in COCOABENCH: *vision*, *search*, *coding*, as detailed in Table 2. As shown in Figure 6, models differ substantially in their tool usage. GPT-5.4 and Gemini 3.1 Pro allocate over 60% of their tool calls to coding tools, using browser interaction primarily for information acquisition. Kimi-k2.5 and Gemini-Flash-3.0 exhibit the opposite profile: Kimi-k2.5 assigns 51.7% of its calls to vision tools, while Gemini-Flash-3.0 directs 34.0% toward DOM-level search operations. This divergence in tool usage is reflected in task performance. GPT-5.4 (64.0% coding, 36.6% SR) and Gemini 3.1 Pro (63.2% coding, 26.1% SR) achieve the highest success rates, whereas Kimi-k2.5 (26.4% coding, 11.8% SR) and Qwen3.5-397B-A13B (31.3% coding, 9.8% SR) allocate near 30% of their tool calls to coding and rank at the bottom. This pattern suggests that code execution serves a dual role in COCOABENCH: as an efficient action space that reduces the number of interaction steps required per subtask, and as an analytical tool that enables complex reasoning over gathered information. Stronger models leverage this by separating information acquisition (via vision and search) from downstream processing (via code), whereas weaker models underutilize programmatic processing and remain in the browser for both phases.

To complement these observational tool-use statistics, we conduct controlled capability-removal experiments and analyze performance by required capability combination. We additionally compare DOM-only and screenshot-only browser observations. The complete experimental setup and results are provided in Appendix D.

5.4 Error analysis

To understand where agents fail on COCOABENCH and what these failures reveal about the challenges of building unified digital agents, we conduct a structured error analysis over all six evaluated models, covering 712 failure trajectories out of 918 total task attempts. Failure causes are annotated by an LLM as judge (Claude Sonnet 4.6, grade prompt is provided in Appendix C.5). We organize the annotated error types into three classes. **Reasoning & Planning** (E1) describes cases where agents fail to devise an effective approach, reason imprecisely about crucial details, or lose track of task requirements such as the requested output format. **Tool & Execution** (E2) focuses on how the agent interacts with tools and interfaces, especially whether it can execute the right steps, in the right order, and recover when execution goes off track. **Visual Grounding** (E3) captures cases where agents fail

to properly perceive or interpret visual information, such as overlooking subtle but task critical details, confusing interface elements, or misreading visual content. The overall error distribution across these categories is shown in Figure 8 (left). Full category definitions and concrete examples are provided in Appendix C. We additionally validate the LLM-based failure annotations through a blinded human calibration study, reported in Appendix E. The LLM judge is used only for diagnostic failure analysis and never determines task success.

To better understand the gap between the leading agent model and other models, we compare the error distributions of GPT 5.4 and Kimi K2.5 (Figure 8, right). Relative to GPT 5.4, Kimi K2.5 fails more often on (E1.1) incorrect reasoning, suggesting weaker procedural knowledge for handling diverse scenarios. It also exhibits a substantially higher rate of format errors (E1.3), indicating that over long interaction horizons, it is more likely to lose track of instructions introduced earlier in the trajectory. In terms of tool use, Kimi K2.5 is more prone to infinite loops (E2.1): when tool outputs are unexpected, it is more likely to get stuck in repetitive tool calls and fail to recover. Finally, it underperforms noticeably on visual grounding errors, especially (E3.1) visual detail, indicating that it is less reliable at noticing fine grained visual information.

6 Conclusion

COCOABENCH is designed to evaluate unified digital agents beyond isolated capability tests, focusing instead on whether agents can flexibly compose vision, search, and coding to solve complex digital tasks. Across both end to end agent systems and controlled evaluations under the shared COCOA-AGENT scaffold, our results show that current systems still struggle to solve COCOABENCH reliably. Our analysis further suggests that coding is an important ingredient in strong agent performance, while error analysis reveals crucial weaknesses in planning and reasoning, tool use and execution, and visual grounding, which point to several promising directions for future work. We hope COCOABENCH and COCOA-AGENT can serve as useful foundations for future research on more capable general purpose digital agents.

References

Anthropic. Introducing claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, February 2026a. Anthropic product announcement.

Anthropic. Claude cowork by anthropic. https://www.anthropic.com/product/claude-cowork, January 2026b. Anthropic release page, accessed 2026-03-31.

Chaithanya Bandi, Ben Hertzberg, Geobio Boo, Tejas Polakam, Jeff Da, Sami Hassaan, Manasi Sharma, Andrew Park, Ernesto Hernandez, Dan Rambado, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

ByteDance Seed. Seed2.0. https://seed.bytedance.com/en/seed2, 2026. Official model page, accessed 2026-03-19.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

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

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=z53s5p0qhf.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces, 2026. URL https://arxiv.org/abs/2601.11868.

Moonshot AI. Kimi K2.5: Visual Agentic Intelligence. https://www.kimi.com/blog/kimi-k2-5, January 2026. Kimi technical blog, accessed March 30, 2026.

OpenAI. Deep research system card. https://cdn.openai.com/deep-research-system-card.pdf, February 2025a. Official system card, published 2025-02-25.

OpenAI. Introducing chatgpt agent: bridging research and action, July 2025b. URL https://openai.com/index/introducing-chatgpt-agent/.

OpenAI. Introducing gpt-5.4. https://openai.com/index/introducing-gpt-5-4/, March 2026. OpenAI release page, accessed 2026-03-19.

OpenClaw. Openclaw, 2026. URL https://github.com/openclaw/openclaw. Official GitHub repository, accessed 2026-03-19.

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

Qwen Team. Qwen3.5: Towards Native Multimodal Agents. https://qwen.ai/blog?id=qwen3.5, February 2026. Qwen blog, accessed March 30, 2026.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, Chong Liu, Daoguang Zan, Fuxing Leng, Hanbin Wang, Hao Yu, Haobin Chen, Hongyi Guo, Jing Su, Jingjia Huang, Kai Shen, Kaiyu Shi, Lin Yan, Peiyao Zhao, Pengfei Liu, Qinghao Ye, Renjie Zheng, Shulin Xin, Wayne Xin Zhao, Wen Heng, Wenhao Huang, Wenqian Wang, Xiaobo Qin, Yi Lin, Youbin Wu, Zehui Chen, Zihao Wang, Baoquan Zhong, Xinchun Zhang,

Xujing Li, Yuanfan Li, Zhongkai Zhao, Chengquan Jiang, Faming Wu, Haotian Zhou, Jinlin Pang, Li Han, Qi Liu, Qianli Ma, Siyao Liu, Songhua Cai, Wenqi Fu, Xin Liu, Yaohui Wang, Zhi Zhang, Bo Zhou, Guoliang Li, Jiajun Shi, Jiale Yang, Jie Tang, Li Li, Qihua Han, Taoran Lu, Woyu Lin, Xiaokang Tong, Xinyao Li, Yichi Zhang, Yu Miao, Zhengxuan Jiang, Zili Li, Ziyuan Zhao, Chenxin Li, Dehua Ma, Feng Lin, Ge Zhang, Haihua Yang, Hangyu Guo, Hongda Zhu, Jiaheng Liu, Junda Du, Kai Cai, Kuanye Li, Lichen Yuan, Meilan Han, Minchao Wang, Shuyue Guo, Tianhao Cheng, Xiaobo Ma, Xiaojun Xiao, Xiaolong Huang, Xinjie Chen, Yidi Du, Yilin Chen, Yiwen Wang, Zhaojian Li, Zhenzhu Yang, Zhiyuan Zeng, Chaolin Jin, Chen Li, Hao Chen, Haoli Chen, Jian Chen, Qinghao Zhao, and Guang Shi. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning, 2025a. URL https://arxiv.org/abs/2509.02544.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025b. URL https://arxiv.org/abs/2407.16741.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Zheng Boyuan, LI PEIHANG, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Hu Jiarui, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Y.Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open foundations for computer-use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025c. URL https://openreview.net/forum?id=6iRZvJiC9Q.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=tN61DTr4Ed.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction, 2025. URL https://arxiv.org/abs/2412.04454.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=mXpq6ut8J3.

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

A Contributors

Data Curation

Zhining Zhang[1], Tianyang Liu[1], Yuheng Zha[1], Qiyue Gao[1], Jixuan Chen[1], Hexi Jin[1], Boyuan Zheng[1], Shibo Hao[1]

Additional Task Authors

Yijiang Li[1], Tommaso Cerruti[5], Licheng Liu[6], Zhifei Li[4], Zhengtao Han[4], Pracha Promthaw[1]

Infrastructure

Zhiqi Liang[1], Junli Wang[1], Zilong Wang[1]

Evaluation

Haoxiang Zhang[1], Hexi Jin[1], Boyuan Zheng[1], Junli Wang[1], Zhiqi Liang[1], Yuheng Zha[1], Qiyue Gao[1], Jixuan Chen[1], Kun Zhou[1]

Conceptualization and Advising

Shibo Hao[1,†], Ziqiao Ma[3], Zhoujun Cheng[1], Yu Wang[1], Tianyang Liu[1], Feng Yao[1], Xiaohan Fu[7], Jingbo Shang[1], Lianhui Qin[1], Julian McAuley[1], Eric P. Xing[2], Zhengzhong Liu[2], Rupesh Kumar Srivastava[2], Zhiting Hu[1]

Affiliations: [1] UC San Diego; [2] MBZUAI IFM; [3] University of Michigan; [4] UC Berkeley; [5] ETH Zurich; [6] University of Cambridge; [7] Gray Swan AI.

[†]Corresponding author: s5hao@ucsd.edu.

B COCOA-AGENT

COCOA-AGENT Tool Interface. COCOA-AGENT exposes a structured tool interface organized into five categories, totaling 39 tools. The browser tools (17 tools) support fine-grained GUI interaction via both low-level pointer events (BROWSER_CLICK, BROWSER_DRAG_TO, BROWSER_SCROLL) and keyboard input (BROWSER_TYPE, BROWSER_HOTKEY), as well as viewport introspection (BROWSER_SCREENSHOT, BROWSER_GET_VIEWPORT_INFO) for screenshot-based visual grounding. The DOM tools (11 tools) complement this with programmatic access to page structure, including DOM_GET_TEXT, DOM_QUERY_SELECTOR, DOM_EXTRACT_LINKS, and DOM_MARK_ELEMENTS, enabling agents to read and interact with page content without relying solely on pixel-level perception. The file tools (9 tools) cover the full file manipulation workflow—reading, writing, listing, searching, and image reading—while the shell (SHELL_EXECUTE) and code execution (CODE_EXECUTE) tools provide terminal access and sandboxed Python/JavaScript interpretation. This decomposition ensures that each of the three core capabilities tested by COCOABENCH —vision, search, and coding—is supported by dedicated, composable primitives, while keeping the interface strongly typed to minimize tool hallucination.

C Failure Mode Taxonomy

We systematically categorize failures on COCOABENCH into three hierarchical layers based on extensive trajectory analysis across different agentic systems on 153 tasks. The taxonomy is organized around the *locus of failure*: whether the root cause resides in the agent's adaptive planning process (E1), its execution loop (E2), or its vision perceptual grounding in the environment (E3).

Table 2: Complete tool inventory of COCOA-AGENT. Tools are grouped by the three key capabilities defined in COCOABENCH: *Vision* (GUI-level browser interaction), *Search* (DOM-level content access and navigation), and *Coding* (code execution, shell commands, and file operations). The *Control* category contains only the task completion signal.

Ability	Tool	Description
Vision	`browser_click`	Click at screen coordinates (left/right/middle; single/double/triple)
	`browser_type`	Type text into the focused element
	`browser_press`	Press a single keyboard key
	`browser_hotkey`	Press a key combination (e.g., `Ctrl+C`)
	`browser_scroll`	Scroll the page by pixel offset
	`browser_move_to`	Move the cursor to absolute coordinates
	`browser_drag_to`	Drag from current position to absolute coordinates
	`browser_wait`	Wait for a specified duration
	`browser_screenshot`	Capture a screenshot of the current viewport
	`browser_get_viewport_info`	Return current URL and viewport dimensions
	`image_read`	Read an image file and return it as base64 for visual analysis
Search	`browser_navigate`	Navigate to a URL (DOM load)
	`dom_get_text`	Retrieve `innerText` of the page body
	`dom_get_html`	Retrieve full page HTML (truncated if long)
	`dom_query_selector`	Query elements by CSS selector; return tag, id, class, role, etc.
	`dom_extract_links`	Extract all hyperlinks (text + href), with optional filtering
	`dom_mark_elements`	Annotate interactive elements with unique BIDs; return element list
	`dom_click`	Click an element by BID
	`dom_type`	Type into an input element by BID
	`dom_scroll`	Scroll an element or the page by BID
Coding	`code_execute`	Execute Python or JavaScript in the sandbox; returns stdout/stderr
	`shell_execute`	Execute a shell command and return output
	`file_read`	Read file contents
	`file_write`	Write content to a file
Control	`task_complete`	Mark the task as finished and return an optional result string

C.1 Type 1 Reasoning & Planning

Planning error arises when the agent's high-level reasoning or decision-making process is fundamentally misaligned with the task objective. These failures occur prior to or independent of execution.

E1.1 Incorrect Reasoning The agent fails to construct a valid logical path to the target objective. This takes two forms: (1) *Goal displacement*: The agent solves a simplified subproblem instead of the actual task. (2) *Incorrect strategy*: The agent understands the goal but pursues a fundamentally flawed approach, often abandoning core constraints or selecting suboptimal algorithms during execution.

E1.1 | Incorrect Reasoning — goal displacement

eight-puzzle-game · COCOA-AGENT

Task: Interact with a live 8-puzzle to discover the hidden goal state, then solve the puzzle from the initial configuration using the minimum number of moves. Crucially,

the validation code is path-dependent: reaching the identical final state via a different sequence of moves yields a completely different code.

Trajectory: The agent successfully probes the API to infer the hidden goal state and computes the optimal 28-move solution via BFS. However, it executes these moves within its ongoing browser session—which already contains prior exploratory steps—and submits the resulting validation code.
Result: × got "2QGGPD" / expected "6CNTDL"

Explanation: The agent solved the sub-problem (finding the path) but failed the actual task constraint. It did not realize that the environment's validation code strictly requires a clean, optimal-path-only execution, treating "reaching the goal state" as erroneously equivalent to "executing the optimal sequence."

E1.1 | Incorrect Reasoning — missing satisfiability check

meeting-schedule-constraints · CodeX

Task: Assign time slots and rooms to four sessions based on constraint files. Crucially, these files detail varying individual participant preferences, containing a mix of overlapping availabilities and strict personal conflicts. Output a valid schedule satisfying everyone, or report that none exists.

Trajectory: The agent parses the constraints and enumerates assignments using `itertools.product`. It manages to align several schedules but overlooks the strict conflicts between specific individuals. It halts prematurely and submits a schedule that leaves at least one participant's requirement violated.
Result: × wrong schedule / expected "(none)"

Explanation: The individual participant constraints inherently conflict, making a universal schedule impossible (jointly unsatisfiable). The agent employed an incomplete reasoning strategy: it optimized for local agreement but failed to rigorously cross-check the personal conflicts among all individuals, missing the fact that the only correct answer is an empty schedule.

E1.2 Imprecision The agent executes the correct high-level procedure but returns an incorrect value due to execution-level inaccuracies. This manifests in two ways: In the *precision* variant, floating-point accumulation or premature rounding introduces a small but evaluation-critical offset to an otherwise sound computation. In the *scope* variant, the agent applies the correct algorithm to the wrong data boundary, such as failing to filter out items that should be excluded (e.g., counting appendix citations alongside the main body).

E1.2 | Imprecision — floating-point accumulation

yearly-costco-gas-receipts · CodeX

Task: Compute the weighted average gas price across a full year of Costco gas receipts, rounding the final output to exactly three decimal places.

Trajectory: The agent correctly identifies all 2023 receipts and formulates the right price-volume product logic. However, it iteratively sums these values using intermediate rounding rather than maintaining full floating-point precision until the final division.
Result: × got "4.217" / expected "4.216"

Explanation: The high-level mathematical procedure is completely sound, but the execution suffers from imprecision. By failing to defer rounding to the absolute final step, floating-point accumulation introduces a tiny but evaluation-critical offset of 0.001.

E1.2 | Imprecision — scope boundary

phd-student-paper-analysis · COCOA-AGENT

Task: Count how many times specific top-cited references appear strictly within the **main body** of a PhD thesis, explicitly excluding any appendices.

Trajectory: The agent parses the PDF and correctly identifies the target citation string (e.g., "Saparov and He, 2022"). It then executes a global text search across the entire document, tallying all matches without applying any structural or section-based filtering.
Result: × got 5 / expected 4

Explanation: The agent applied the correct counting algorithm but to the wrong data boundary. By blindly scanning the full document, it erroneously included citations located in the appendix, silently corrupting the final count due to a scope mis-specification.

E1.3 Format Error The agent derives the correct answer but fails to structure it for the evaluator. This occurs via: (1) *Tag omission*: Burying the valid answer in prose without the required `<answer>` tags. (2) *Partial delivery*: Submitting only a subset of a required multi-part output, treating the first completed field as the entire answer.

E1.3 | Format Error — partial delivery

tableau-profit-margin-analysis · CodeX

Task: Given a Tableau dashboard, extract two insights for a specified region: (a) the month with the highest profit margin, and (b) the year with the most volatile profit margins.

Trajectory: The agent successfully scrapes the underlying dashboard data, computing both the monthly profit margins and the yearly standard deviations. However, immediately upon resolving the first sub-question, it prematurely terminates the run and submits only the highest-margin month (May, 26.0%), silently discarding its volatility computations.
Result: × got correct answer to (a) only / expected answers to (a) and (b)

Explanation: The agent's analytical reasoning and data extraction were flawless, but it failed on output formatting constraints. By treating the first resolved sub-query as the completion of the entire prompt, it committed a partial delivery error, leaving the second requirement unfulfilled despite already possessing the correct data.

C.2 Type 2 Tool & Execution

While the high-level plan may be sound, Type 2 failures emerge as structural breakdowns in the agent's active interaction loop. The task halts not due to bad logic, but through poor execution: tool misuse, absent recovery mechanisms, or behavioral stagnation.

E2.1 Infinite Loop The agent fails to self-correct and falls into an endless execution cycle. This manifests in two primary ways: (1) *Repetition*: Blindly re-issuing identical actions despite receiving consistent error signals. (2) *Exhaustion*: Endlessly tweaking low-level parameters (e.g., scrolling, resizing) without realizing the overarching strategy is doomed. Crucially, this includes environment-induced loops (e.g., broken tool APIs), as the underlying failure is the agent's inability to recognize and escalate a systemic block.

E2.1 | Infinite Loop — exhaustion variant

nonogram · Claude Code

Task: Solve a nonogram puzzle provided as an image and identify the specific letter formed by the completed grid.

Trajectory: The agent attempts to transcribe the puzzle's numerical clues using a low-level visual extraction strategy. It expends its entire 51-turn budget iteratively cropping and reading tiny row and column regions via PIL, endlessly tweaking its focus area but never progressing to the actual solving phase.
Result: × got (none) / expected "w"

Explanation: The agent became trapped in an exhaustion loop. Instead of recognizing the severe inefficiency of manual pixel-level transcription and pivoting to a higher-level abstraction (e.g., passing the full image to a robust OCR tool once and writing a programmatic constraint solver).

E2.2 Anti-Bot Barriers The agent is blocked by websites' security mechanisms but fails to recognize the interruption. It misinterprets the interstitial bot-detection page as the actual target content, resulting in hallucinated answers or silent failures without user escalation.

E2.2 | Anti-Bot Barriers

neurips-citation-analysis · CodeX

Task: Retrieve current citation counts for specific NeurIPS papers using public academic indexing services.

Trajectory: The agent attempts to scrape an academic database but is intercepted by a Cloudflare bot-detection challenge. Failing to recognize the HTML response as a security barrier, it blindly parses the challenge page as if it were the target paper's record. Finding no citation metrics within the boilerplate security warning, it confidently outputs a citation count of zero.
Result: × got 0 / expected actual citation count

Explanation: The agent's failure stems entirely from its inability to contextualize interstitial network obstacles. Instead of detecting the anti-bot barrier and pivoting its retrieval strategy (or safely aborting), it treated the security interception as a valid, empty search result. This lack of situational awareness directly transformed a simple tool-access block into a confidently hallucinated factual error.

E2.3 Tool Result Hallucination The agent proceeds based on fabricated outputs or a corrupted memory state. (1) *Tool hallucination*: Invoking non-existent tools or fabricating execution results without actually issuing a call. (2) *Context truncation*: As the interaction history exceeds the context window, early instructions or critical findings are silently dropped. The agent continues unawares, leading to repetitive actions and progressive decoupling from the core objective.

E2.3 | Tool Result Hallucination — context truncation

glp1-pharmaceutical-analysis · COCOA-AGENT

Task: Extract and systematically compare primary endpoint data for GLP-1 drug trials across a dense corpus of 47 research papers.

Trajectory: The agent initiates a systematic review, successfully retrieving and extracting data from several papers in the early turns. However, as the dense action-observation history exceeds the context window limits, these early findings are silently evicted (compacted). Unaware of this memory loss, the agent repeatedly notes that prior results are "missing," prompting it to re-fetch the exact same endpoint data and restart sub-tasks from scratch. This endless cycle persists until the run halts at turn 71.

Result: × got (incomplete) / expected full comparative analysis

Explanation: This is a textbook manifestation of context truncation. The agent's logic did not fail; rather, it was forced to operate on a degraded, amnesic memory state. Because critical early findings were dropped to accommodate new observations, the agent lost its global state awareness. Instead of synthesizing building blocks into

a final comparison, it became trapped in a Sisyphean cycle of redundant retrieval, progressively decoupling from the overarching objective.

C.3 Type 3 Visual Grounding

While Type 1 and Type 2 failures stem from flawed strategy or broken interaction loops, Type 3 errors represent a fundamental perceptual disconnect. Here, the agent may reason correctly and execute actions smoothly, but ultimately fails because it misreads, overlooks, or cannot semantically map the visual state of the environment.

E3.1 Visual Detail The agent captures the global layout of a scene but fails to accurately resolve fine-grained visual features. This deficiency primarily manifests in three areas: *small-target detection* (missing or mislocalizing tiny icons and dense UI elements), *text recognition* (misreading small, low-contrast, or stylized fonts), and *thin-object perception* (misjudging the alignment, boundaries, or intersections of lines, arrows, and borders).

E3.1 | Visual Detail — measurement imprecision

mountain-video · CodeX

Task: Watch a mountain-climbing video and extract the exact elevation of each labeled step shown on the route diagram.

Trajectory: The agent successfully isolates the relevant video frame and applies both its Vision-Language Model (VLM) capabilities and OCR tools to parse the on-screen text. While it correctly reads the surrounding elevation fields, it misreads a single digit in the third-step marker's label, extracting 8710 m instead of the true value.

Result: × got 8710 m / expected 8690 m

Explanation: This failure highlights a critical deficit in fine-grained visual acuity. Even when directly combining holistic VLM perception with dedicated OCR processing, the agent failed to accurately resolve low-level pixel details. Misinterpreting just one digit corrupted the extracted measurement, shifting the final answer by 20 m and pushing it beyond the acceptable evaluation tolerance.

E3.2 Visual Knowledge The agent forms a correct visual representation of the scene but lacks the prior world knowledge to map these features to their corresponding textual concepts. The breakdown is semantic rather than perceptual: the agent clearly "sees" the depicted entities (e.g., specific people, locations, turn signal, or cultural artifacts) but cannot identify them due to insufficient parametric knowledge.

E3.2 | Visual Knowledge — color naming

taylor-swift-eras-tour · Claude Code

Task: Identify the dominant color of a set of Eras Tour album covers and map it to the closest standard color name.

Trajectory: The agent successfully samples the image palette and computes a mathematically precise RGB mean of (148, 103, 189). However, when converting this data into a semantic label, it abruptly classifies this clearly purplish coordinate as "gray" and submits the wrong answer.

Result: × got "gray" / expected "purple"

Explanation: The failure represents a fundamental gap in parametric world knowledge rather than perception. The agent flawlessly extracted the visual representation (the RGB values) but failed to align this data with the correct textual concept, demonstrating a breakdown in visual-to-semantic mapping.

E3.2 | Visual Knowledge — person misidentification

grammy-awards-singer-photo · CodeX

Task: Identify the two specific artists pictured together in a Grammy ceremony photo, then calculate the total combined number of awards they won between 2020 and 2023 (inclusive).

Trajectory: The agent correctly identifies Billie Eilish on the right but misidentifies Lana Del Rey on the left as "Taylor Swift." Proceeding with this flawed visual extraction, it queries the 2020–2023 award histories for Eilish and Swift, summing them to arrive at an incorrect final count.
Result: × got 3 / expected 7

Explanation: The numerical calculation error is merely a downstream symptom of an epistemic visual breakdown. The agent successfully perceived the facial features but lacked the parametric knowledge to map them to Lana Del Rey, defaulting to a superficially similar high-frequency concept (Taylor Swift). This initial visual-to-semantic misidentification poisoned the subsequent data retrieval, dooming the entire multi-hop task.

E3.2 | Visual Knowledge — directional convention

bear-statue-building-direction · CodeX

Task: Determine the cardinal direction a building faces, given a street-view image of a bear statue at a roundabout and the navigation instruction: "enter the roundabout and exit at the first exit."

Trajectory: The agent correctly observes the physical layout of the roundabout and the orientation of the statue. To deduce the path, however, it blindly applies a left-hand-traffic (right-hand drive) heuristic. Assuming traffic flows clockwise, it takes the immediate left branch as the "first exit," failing to recognize from the visual context that the scene is in a right-hand-traffic country.
Result: × got "northwest" / expected "northeast"

Explanation: This is a failure of contextual world knowledge. The agent correctly "read" the visual geometry but applied the wrong geographic prior. By erroneously defaulting to a left-hand-traffic convention, it simulated the roundabout navigation backwards (clockwise instead of counterclockwise), reversing the vehicle's bearing and ultimately producing the wrong cardinal direction.

E3.3 Missing Visual Perception The agent reads the page via DOM queries rather than inspecting the rendered visual output, causing it to miss content that exists only in the pixel buffer. Modern web applications frequently render data through `<canvas>` elements, SVG overlays, or JavaScript-driven frameworks that expose no semantic content to the DOM; the agent either reports such content as absent or fabricates plausible values from contextual priors.

E3.3 | Missing Visual Perception — DOM opacity

tableau-profit-margin-analysis · CodeX

Task: Read a live Tableau dashboard and visually extract the month with the highest profit margin and the year with the most volatile margins for a given region.

Trajectory: Attempting to bypass visual processing entirely, the agent invokes a back-end scraper API (`tableauscraper`) to pull the raw underlying data fields. Relying strictly on this textual payload, it identifies "May" at "26.0%" and confidently submits it without ever inspecting the rendered dashboard pixels.
Result: × got May / 26.0% / expected March / 15.3%

Explanation: The agent fell victim to a semantic "mirage." Modern BI tools like Tableau perform complex client-side aggregations that only manifest in the final visual render. By relying exclusively on the raw, unaggregated API payload rather than the visual ground truth, the agent extracted mathematically correct but visually absent data, completely failing the inherently perceptual task.

E3.3 | Missing Visual Perception — SVG measurement mismatch

quadrant-visual-issues · CodeX

Task: Render a Mermaid quadrant chart, identify visual issues such as overlapping labels (Q2), and find the minimum `chartWidth` (in multiples of 100) that visually resolves all overlaps (Q3).

Trajectory: The agent renders the chart using Playwright but attempts to detect label collisions by querying DOM bounding boxes (`getBoundingClientRect()`) rather than employing actual visual inspection. The DOM queries falsely report that overlaps persist at a width of 800 px. Trusting this API over its own visual capabilities, the agent needlessly widens the chart until the DOM clears the overlaps at 1200 px.

Result: × got Q3 = 1200 / expected Q3 = 800 (Q1 and Q2 were correct)

Explanation: The agent attempted to solve a strictly visual geometry problem using non-visual DOM structures. In complex SVGs, structural bounding boxes often overlap even when the rendered text pixels do not. By blindly trusting this programmatic mirage over the actual pixel-level rendering, which clearly showed overlaps were resolved at 800 px—the agent overcompensated and submitted a grossly inflated width.

C.4 More Error Analysis

Figures 9–16 present the full failure-mode breakdown for every model and scaffold configuration evaluated on COCOABENCH. Each donut chart uses the revised three-tier taxonomy described in Section C.1– C.3: the inner ring shows the aggregate share of **Reasoning & Planning** (E1, brown), **Tool & Execution** (E2, gold), and **Visual Grounding** (E3, teal); the outer ring details the active leaf subcategories (E1.1–E3.3), with arcs shaded from dark to light in descending frequency within each group, and parenthesized values indicating raw occurrence counts. Displayed percentages in the outer ring reflect each subcategory's share of *total failure-mode mentions* (i.e. the sum of all Ex.x codes assigned across all failed runs); a single trajectory may contribute multiple failure modes.

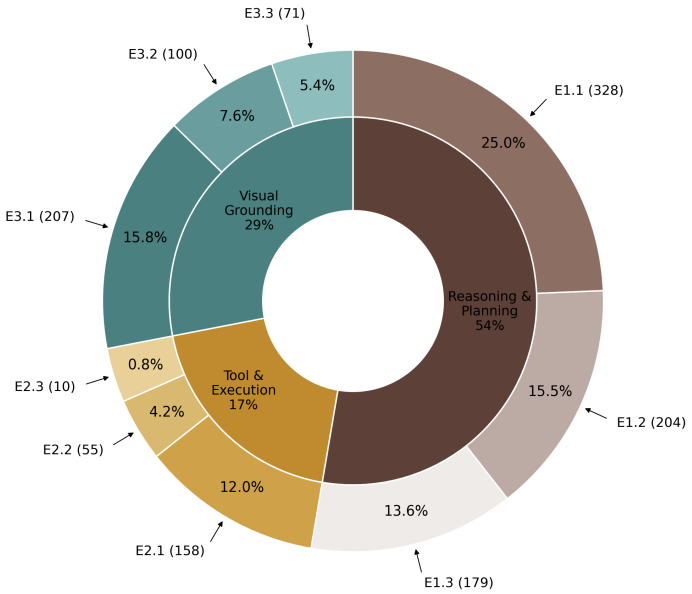

Figure 9: Aggregate failure distribution across all 6 models evaluated under COCOA-AGENT on COCOABENCH. The inner ring: Reasoning & Planning (54%), Tool & Execution (17%), Visual Grounding (29%). The outer ring details all 9 active subcategories (E1.1–E3.3). Displayed percentages reflect each subcategory's share of total failure-mode mentions across all 722 failed runs.

Figure 10: COCOA-AGENT with Claude Sonnet 4.6 failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (50%), Tool & Execution (17%), Visual Grounding (33%). The outer ring details 9 active subcategories. Displayed percentages reflect each subcategory's share of total failure-mode mentions across all 114 failed runs.

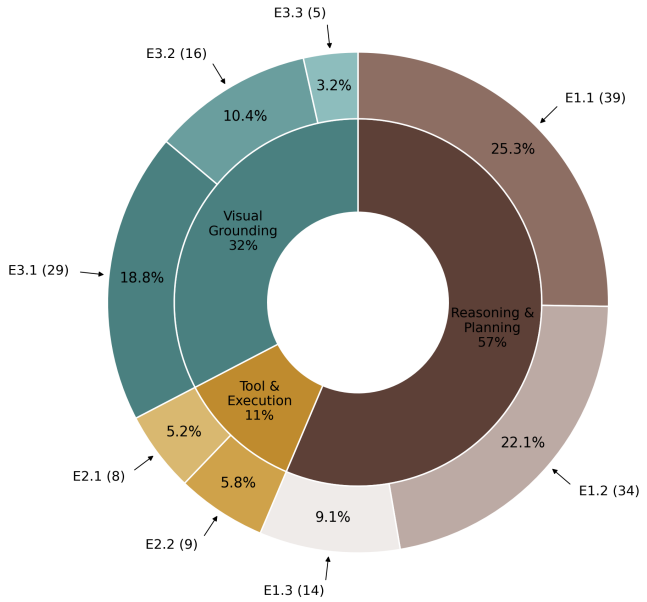

Figure 11: COCOA-AGENT with GPT-5.4 failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (57%), Tool & Execution (11%), Visual Grounding (32%). The outer ring details 8 active subcategories. Displayed percentages reflect each subcategory's share of the total failure-mode mentions across all 100 failed runs.

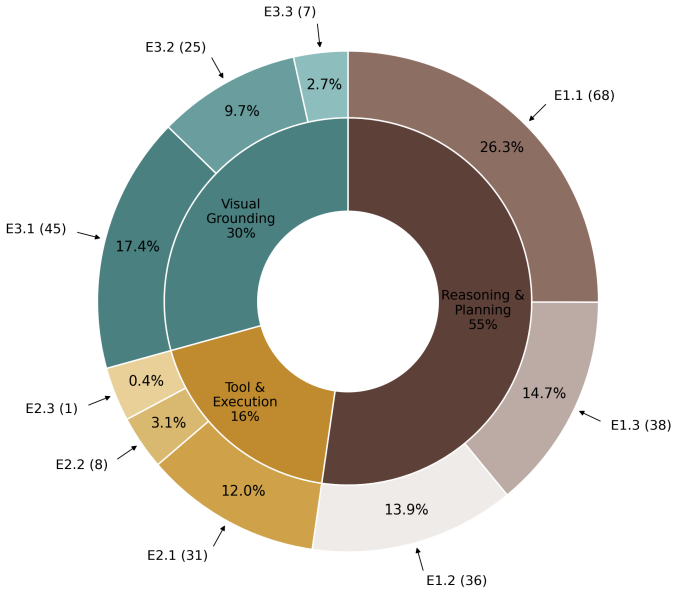

Figure 12: COCOA-AGENT with Kimi-k2.5 failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (55%), Tool & Execution (15%), Visual Grounding (30%). The outer ring details 9 active subcategories. Displayed percentages reflect each subcategory's share of the total failure-mode mentions across all 137 failed runs.

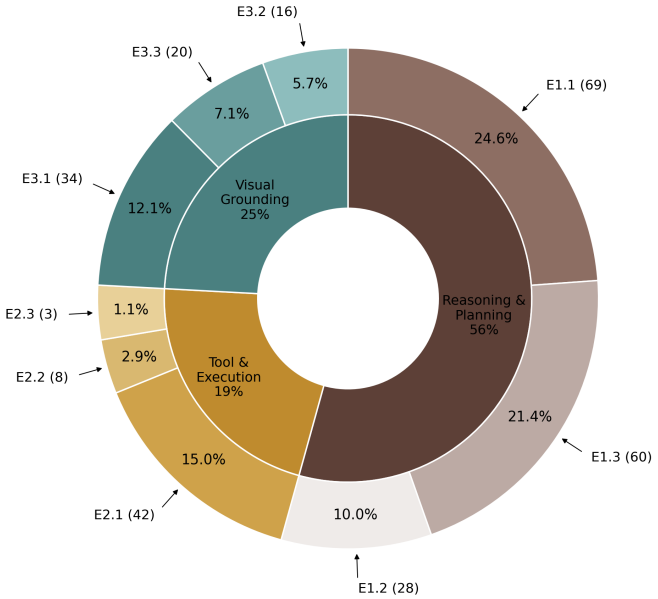

Figure 13: COCOA-AGENT with Qwen3.5-397B failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (56%), Tool & Execution (19%), Visual Grounding (25%). The outer ring details 9 active subcategories. Displayed percentages reflect each subcategory's share of the total failure-mode mentions across all 139 failed runs.

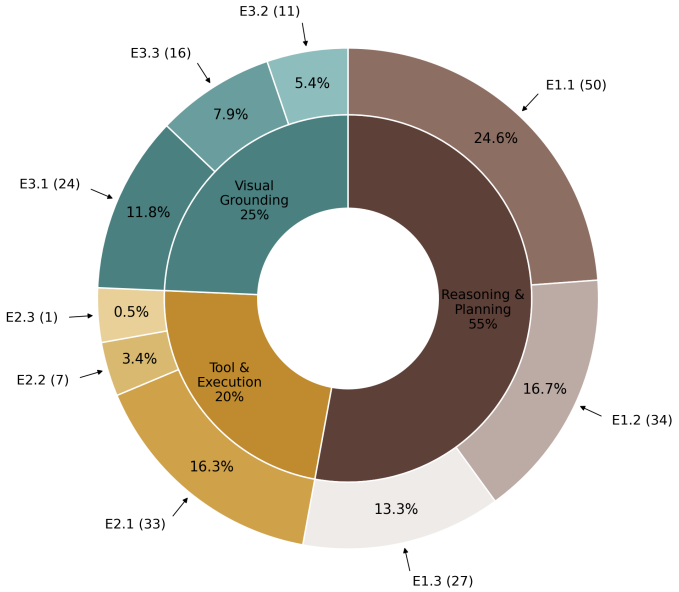

Figure 14: COCOA-AGENT with Gemini 3.1 Pro Thinking failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (55%), Tool & Execution (20%), Visual Grounding (25%). The outer ring details 9 active subcategories. Displayed percentages reflect each subcategory's share of the total failure-mode mentions across all 107 failed runs.

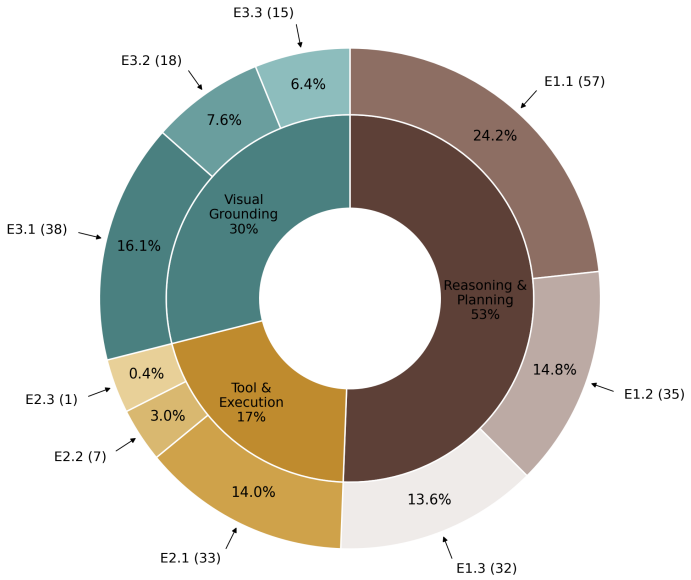

Figure 15: COCOA-AGENT with Gemini 3 Flash failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (53%), Tool & Execution (17%), Visual Grounding (30%). The outer ring details 9 active subcategories. Displayed percentages reflect each subcategory's share of the total failure-mode mentions across all 125 failed runs.

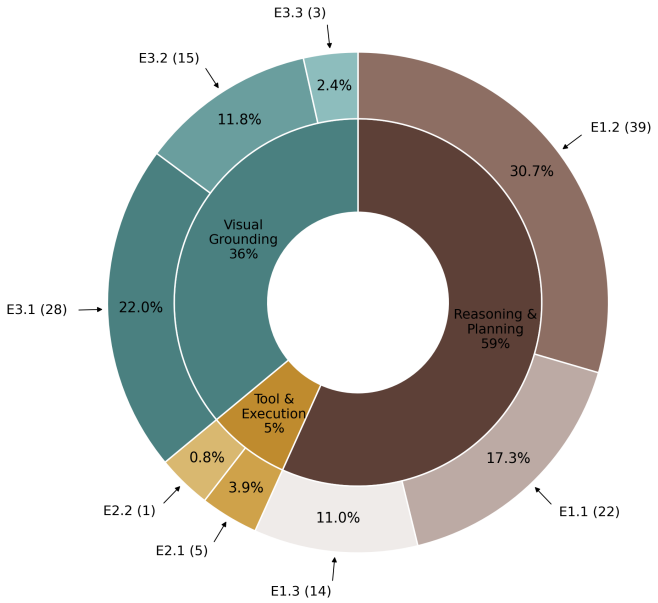

Figure 16: OpenAI Codex failures on COCOABENCH mapped to the failure taxonomy. The inner ring: Reasoning & Planning (59%), Tool & Execution (5%), Visual Grounding (36%). The outer ring details 8 active subcategories. Displayed percentages reflect each subcategory's share of total failure-mode mentions across all 86 failed runs.

C.5 LLM-as-Judge: Error Classification Prompt

We use an LLM-as-judge (claude-sonnet-4-6) to assign one or more failure-mode codes from the Error taxonomy (Sections C.1–C.3) to each failed trajectory. The judge receives five inputs per run: the task description (README), the evaluator's expected answer, a run summary (status, iteration count, agent answer, evaluator feedback), and a compact text trace reconstructed from the execution log—screenshots and base-64 blobs are removed, and per-action observations are truncated to a fixed character budget by action type (e.g. 400 chars for DOM element lists, 0 chars for pure screenshots, 600 chars default). The full prompt is reproduced below; the taxonomy section mirrors the definitions in Sections C.1–C.3 exactly.

The judge is instructed to *include* a category whenever the failure *partially or primarily* matches its description, erring on the side of inclusion; a single trajectory may receive multiple codes. A free-text `OTHER:<phrase>` escape is provided for failures that genuinely fall outside the taxonomy, though in practice fewer than 0.5% of classified runs use it. The judge is asked to reply with *only* a comma-separated list of codes (e.g. `E1.1, E3.3`) so that the output is unambiguously parsable with a short regex; no chain-of-thought is elicited and no system prompt is used.

Prompt | LLM-as-judge classification prompt

```
[Role]
You are an expert AI benchmark analyst. A benchmark agent has failed the task
below. Classify this failure into one or more categories from the taxonomy
provided.

[Classification guidelines]
- Apply a category if the failure partially or primarily matches its
description. Err on the side of inclusion rather than exclusion.
- A single run may exhibit multiple failure modes; list all that apply.
- If no existing Ex.x captures the core failure, append OTHER:<one short
phrase>. Only use OTHER when truly nothing fits.

[Taxonomy]
Type 1 --- Reasoning & Planning
E1.1 Incorrect Reasoning / E1.2 Imprecision / E1.3 Format Error

Type 2 --- Tool & Execution
E2.1 Infinite Loop / E2.2 Anti-Bot Barriers / E2.3 Tool Result Hallucination

Type 3 --- Visual Grounding
E3.1 Visual Detail / E3.2 Visual Knowledge / E3.3 Missing Visual Perception

[Per-run context]
Task: <task_name>

### Task Instruction
<full README.md of the task>

### Scoring / Expected Answer
<EXPECTED = ... lines extracted from test.py>

### Run Summary
- Run status : <passed | failed | error>
- Iterations : <N>
- Agent answer : <task_result or answer field>
- Eval feedback: <evaluator feedback string>

### Agent Action Trace (screenshots removed; observations truncated)
<compact trace: [N] THINK: ... / ACTION: ... / OBS: ...>
<total capped at 1M chars via head+tail truncation>

[Output instruction]
```

```
Which failure categories apply to this run?
Reply with ONLY a comma-separated list of Ex.x codes (and optionally
OTHER:<short-phrase>). Nothing else.
Example: E1.1, E3.3, OTHER:dependency-missing
```

Observation truncation budget. To keep the trace within the model's effective context while preserving the most diagnostic signal, observations are truncated per action type before the trace is assembled (Table 3). Truncation is applied as a *head* slice (the first L characters), so the opening structure of each observation—page title, status code, element count—is always retained. The total trace is then hard-capped at 10^6 characters using a symmetric head+tail window; in practice, fewer than 2% of runs hit this cap.

Table 3: Observation truncation limits per action type used when building the judge prompt.

Action Type	Limit (chars)	Rationale
DOM & Navigation (Prone to long lists; only page context needed)		
`dom_get_text`	500	
`dom_mark_elements`	400	
`dom_get_html`	300	
`browser_navigate`	300	
`browser_get_viewport_info`	200	
Visual & Binary Data (Non-textual data; omitted entirely)		
`browser_screenshot`	0	Pure image data
`image_read`	0	Base-64 encoded blob
(All other actions)	600	Default text limit

D Capability and Observation Analyses

D.1 Performance by required capability combination

We report the success rate of every evaluated agent after stratifying tasks by their human-annotated required capability combination in Table 4.

The results reveal distinct specializations and limitations across language models. Under the shared COCOA-AGENT scaffold, Gemini 3.1 Pro performs strongly on vision+search tasks, but falls behind GPT-5.4 on tasks requiring coding. The vision+search+coding tasks also amplify differences among agent systems. They are particularly difficult for weaker backbones—for example, COCOA-AGENT with Claude Sonnet 4.6 reaches 9.52%—while a well-designed agent system such as Codex with GPT-5.4 reaches 54.76%.

D.2 Capability-removal ablations

The human capability labels describe task requirements, but do not by themselves establish which capabilities are important for an agent's observed performance. We therefore disable one capability pathway at a time under the controlled COCOA-AGENT scaffold with GPT-5.4 while leaving the remaining scaffold unchanged, since GPT-5.4 is the strongest and most stable configuration in our experiments. Table 6 specifies the intervention used for each ablation.

As shown in Table 5, removing coding causes the largest reduction, from 36.6% to 23.53% (-13.1 points), while removing search causes the second-largest reduction, to 28.10% (-8.5 points). This supports our claim that these tasks require capability composition rather than a single isolated skill.

The drop in the no-vision setting is relatively small. Case studies show that, after `browser_screenshot` and `image_read` are disabled, agents often process images and videos in code using PIL, OpenCV, OCR, and frame extraction. This is consistent with our finding

Agent	V+S (64)	S+C (23)	V+C (21)	V+S+C (42)	Overall (153)
Codex w/ GPT-5.4	43.75%	30.43%	47.62%	**54.76%**	**45.10%**
OpenClaw w/ GPT-5.4	**48.44%**	**39.13%**	**57.14%**	38.10%	**45.10%**
OpenClaw w/ Claude Sonnet 4.6	32.81%	26.09%	42.86%	38.10%	33.99%
ChatGPT Agent	37.50%	8.70%	28.57%	21.43%	26.80%
Claude Code w/ Claude Sonnet 4.6	29.69%	17.39%	23.81%	26.19%	25.49%
OpenAI Deep Research	4.69%	0.00%	4.76%	2.38%	3.27%
COCOA-AGENT w/ GPT-5.4	37.50%	**39.13%**	52.38%	26.19%	36.60%
COCOA-AGENT w/ Gemini 3.1 Pro	40.62%	13.04%	33.33%	23.81%	30.72%
COCOA-AGENT w/ Gemini Flash 3.0	23.44%	8.70%	23.81%	19.05%	19.61%
COCOA-AGENT w/ Claude Sonnet 4.6	21.88%	8.70%	19.05%	9.52%	15.69%
COCOA-AGENT w/ Kimi-k2.5	15.62%	0.00%	19.05%	9.52%	11.76%
COCOA-AGENT w/ Qwen3.5-397B-A13B	14.06%	0.00%	14.29%	7.14%	9.80%

Table 4: Success rate by human-annotated capability combination. V, S, and C denote vision, search, and coding. Parentheses give the number of tasks in each capability group. The best result in each column is bolded; ties are all bolded.

Setting	Available pathways	Success rate	Δ vs. full
Full	V + S + C	**36.6%**	–
−C (no-code)	V + S	23.53%	−13.1%
−S (no-search)	V + C	28.10%	−8.5%
−V (no-vision)	S + C	35.95%	−0.7%

Table 5: Capability-removal ablations for COCOA-AGENT with GPT-5.4 on all 153 tasks. V, S, and C denote vision, search, and coding, respectively.

that coding can be unexpectedly powerful for agents. However, the resulting trajectories are substantially less direct and less efficient. Among successful trajectories on the shared set of prompts, the no-vision setting requires approximately 1.7× more output tokens and 1.75× more tool calls than the full setting.

Please note that this ablation is meaningful only within GPT-5.4's current approximately 36% solvable range, and capability removal may matter more on the harder tail.

D.3 Web observation modality

We separately evaluate web-interaction modality ablations to examine which form of computer-use observation the agent relies on. This constitutes a separate axis from capability removal.

As shown in Table 7, DOM-only obtains slightly higher overall performance than the other settings. These results should be interpreted with caution: COCOABENCH is designed to evaluate the composition of coding, search, and vision, rather than to isolate web-observation modality, as also reflected in the small and mixed differences across task groups.

Qualitative inspection of agent trajectories reveals a trade-off. With DOM-based structured representations, agents may miss purely visual information; for example, content rendered as SVG can be silently omitted. This corresponds to the Missing Visual Perception category (E3.3), for which we provide a case study in Appendix C. Conversely, structured representations are highly effective on data-heavy tasks such as statistical tables because the agent can retrieve the structured content and process it directly in code. The full COCOA-AGENT setting provides both interaction modalities. A dedicated, controlled study of observation modalities is an interesting direction for future work.

Setting	Intervention
−C (no-code)	Disable `shell_execute` and `code_execute`.
−S (no-search)	Remove general internet access from the sandbox while whitelisting URLs explicitly required by the task prompt.
−V (no-vision)	Disable `browser_screenshot` and `image_read`; code-based loading, OCR, image processing, and video-frame extraction remain possible.

Table 6: Implementation of the capability-removal interventions.

Setting	Browser observation	Overall	V+S	V+C	S+C	V+S+C
Full	DOM + screenshots	36.60%	37.50%	**52.38%**	**39.13%**	26.19%
DOM-only	DOM/HTML/text; no screenshots	**40.52%**	**40.62%**	47.62%	**39.13%**	**40.48%**
Screenshot-only	Screenshots; no DOM/raw page text	36.60%	37.50%	47.62%	26.09%	35.71%

Table 7: Web-observation modality ablations for COCOA-AGENT with GPT-5.4. Best values in each column are bolded.

E Evaluation Reliability Audits

E.1 Construction-time safeguards

Outcome-based evaluation intentionally permits multiple valid execution paths. However, relying solely on the final output as a proxy for success can create false positives if the answer can be guessed or recovered through unintended shortcuts.

We mitigate this risk through a two-phase safeguard. During construction, task authors are required to avoid small answer spaces and prefer open-choice outputs, such as exact entity names, floating-point values, and nontrivial counts. Subsequently, during verification, at least two independent reviewers inspect the task, evaluator, and available resources for any unintended shortcuts. Ultimately, tasks are revised until both reviewers confirm that success strictly requires the intended information gathering, reasoning, or interaction.

For a better illustration, the 8-puzzle task serves as a representative example. It was initially implemented as a pure HTML page. However, to prevent agents from directly inspecting the source code and inferring the answer without actual exploration, we later converted it into a server-backed webpage. In addition, we map the agent's sequence of actions to a verification code. To obtain the correct code, the agent must execute the exact sequence of valid actions, ensuring no steps are bypassed.

E.2 Post-hoc successful-trajectory audit

To quantify residual risk after construction, two authors independently inspected 70 successful COCOA-AGENT trajectories produced by different models. We found only two anomalous cases as follows.

Evaluator gaming. One task asks the agent to identify an artist from an image and then count the artist's awards. GPT-5.4 misidentified the artist and also miscounted the awards, but the two errors coincidentally produced the expected final number, 6. The evaluator therefore produced a false positive.

Memorized. In a street-view recognition and spatial-navigation task, Gemini 3.1 Pro returned the exact location in a single step without tools or explicit reasoning. This suggests that the model may have memorized the location and guessed the navigation outcome.

These audit results, alongside our initial construction and verification safeguards, demonstrate that successful completions overwhelmingly follow legitimate workflows. This ensures the benchmark accurately reflects agents' capabilities in the wild.

Stage	Visibility and procedure	Mean human–LLM IoU
Fully blinded	Initial independent human annotations; humans and LLM are fully blinded	71.53%
Post-discussion	Human and LLM labels fully unblinded for joint review of discrepancies	87.98%

Table 8: Category-level IoU between human annotations and the Claude Sonnet 4.6 failure-mode classifications on 50 failed trajectories.

E.3 Human calibration of failure annotations

We conduct a human calibration study of the LLM-judge error analysis. Three authors independently and manually annotated 50 failed trajectories under a double-blind setup, where human annotators could not see one another's labels or the labels produced by Claude Sonnet 4.6. We then computed the category-level alignment between the human labels and the Claude Sonnet 4.6 classifications using intersection over union (IoU).

The average IoU between the initial independent human annotations and the Claude Sonnet 4.6 classifications is 71.53%. After a comprehensive cross-review phase in which the human and LLM ratings were fully unblinded for joint discussion of discrepancies, the post-discussion IoU increases to 87.98%. While this high human–LLM agreement validates the LLM judge as a reliable tool for analyzing failure-mode distributions, we expressly prohibit its use in evaluating any final agent success rates.

F Resource Stability Audit

F.1 Resource provenance and hosting

The resources required by COCOABENCH fall into three provenance classes. First, self-hosted websites use fixed code and are served from stable hosting such as GitHub Pages. Second, third-party websites are chosen for demonstrated long-term stability; examples include official YouTube videos that have been available for years, closed historical GitHub milestones, and ChatGPT conversation-history pages. Third, task documents are stored in a separate GitHub repository so that the corresponding URLs remain persistently accessible.

F.2 Task-level drift audit

We conduct a resource-level drift audit, classifying the resources required by each task as either static/versioned or live. Among the 153 tasks, 12 tasks (7.8%) involve resources with minor drift risk, including YouTube videos or historical versions of tax calculators. The resources used by the remaining tasks are effectively immutable.