AgentEye-Bench: Benchmarking Multimodal Agents on Image-Grounded Tool-Use Tasks

Anonymous ACL submission

Abstract

Evaluating whether language model agents can extract and reason over image-grounded inputs through programmatic tool use is essential for understanding real-world agentic capability. Existing benchmarks either assess visual understanding without tool execution, or agentic tool use with text-only inputs, leaving image-grounded agentic reasoning largely unmeasured. We introduce AgentEye-Bench, a benchmark of 45 tasks spanning three categories: Code Execution from Image, Math Reasoning from Image, and Data Reasoning from Image, all evaluated within sandboxed Docker containers. In each task, images are placed in the container and inaccessible to the language model as API inputs; the agent must write and execute code to extract image content and produce a deterministic, unit-test-verified output. Evaluation of eleven frontier model routes from Google, OpenAI, and Anthropic shows that the best routes reach only 57.8% Pass@1, with Data Reasoning from Image proving the most challenging category on average. These results suggest that tool-mediated image perception is a critical bottleneck for current language model agents, and we release AgentEye-Bench to support reproducible evaluation of future image-grounded agentic systems.

1 Introduction

Recent advances in large language models drove a shift from single-turn question answering to autonomous agents capable of operating in interactive environments through multi-step tool use. A natural extension of this capability is image-grounded agentic reasoning: the ability to extract information from visual inputs and translate it into executable actions that produce verifiable results. Yet measuring this capability precisely remains difficult, as existing evaluation frameworks treat visual perception and tool execution as separate research problems, leaving their intersection largely unmeasured.

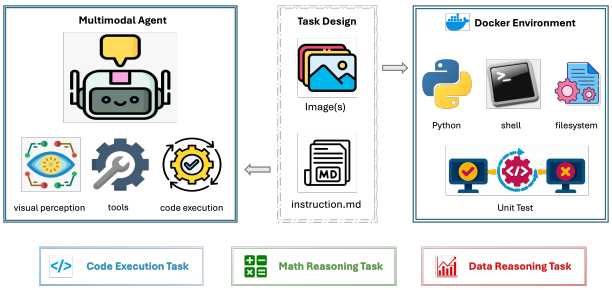

Figure 1: Overview of AgentEye-Bench. The agent receives a plain-text instruction specifying the paths of one or more image files; the image files themselves are copied into a Docker container at launch time and are accessible only via the filesystem. To process image content, the agent selects and executes appropriate tools (e.g., OCR libraries, image-processing scripts) within the container; images are *not* accessible through the large language model's vision API. The container's execution environment (Python, shell, filesystem, etc.) serves as the working surface; task-specific unit tests then verify the agent's written output, producing a binary Pass@1 verdict. Tasks span three categories: Code Execution from Image, Math Reasoning from Image, and Data Reasoning from Image.

Two parallel lines of work address these problems in isolation. Tool-augmented agentic benchmarks such as Terminal-Bench (Team, 2025), Toolathlon (Li et al., 2026a), and SkillsBench (Li et al., 2026b) evaluate agents on complex multi-step tasks within sandboxed terminal environments using deterministic unit tests, but all task inputs are plain text, with no image-grounded reasoning required. Conversely, visual benchmarks such as MMBench (Liu et al., 2024), MMMU-Pro (Yue et al., 2025), and SWE-bench Multimodal (Yang et al., 2025) provide image inputs, but either evaluate visual understanding through static question answering without tool execution, or restrict image types to a narrow domain such as web UI screenshots. As a result, no existing benchmark simultaneously requires diverse visual perception,

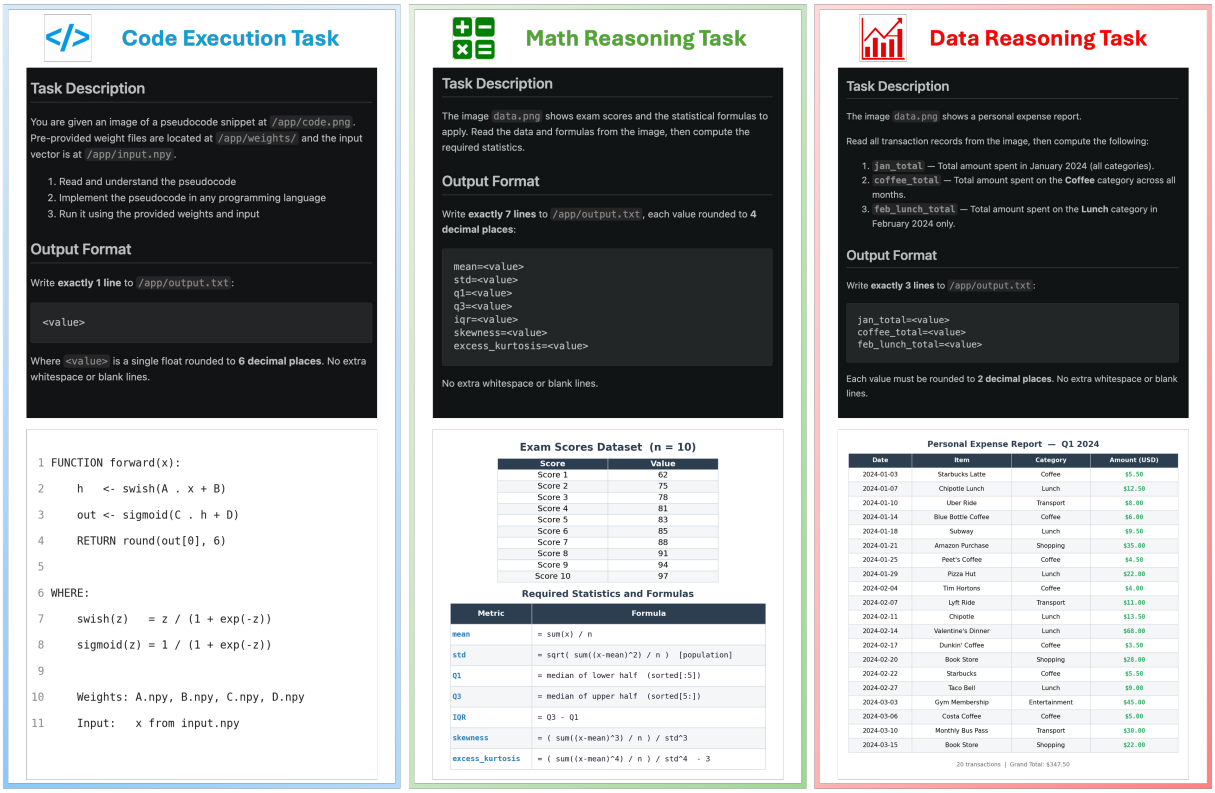

Figure 2: Representative tasks from each AgentEye-Bench category. In each case, the agent receives a plain-text instruction and must extract information from an image inside a Docker container to produce a verifiable output. *Left*: Code Execution from Image — interpreting a pseudocode snippet and running it with provided weights to compute a numeric result. *Center*: Math Reasoning from Image — reading exam scores and statistical formulas from an image, then computing seven descriptive statistics. *Right*: Data Reasoning from Image — parsing a personal expense report and aggregating totals across specified categories and months.

programmatic image extraction, and deterministic output verification, leaving a structural gap in the evaluation of image-grounded agentic reasoning.

To address this gap, we introduce AgentEye-Bench, as shown in Figure 1, a benchmark for evaluating language model agents on tasks that require extracting structured information from images and producing verifiable outputs through tool use in sandboxed Docker environments. AgentEye-Bench comprises 45 tasks across three categories (Code Execution from Image, Math Reasoning from Image, and Data Reasoning from Image), organized into 15 sub-task types with three difficulty-graded instances each, as illustrated in Figure 2. Every task presents one or more images as container files that are inaccessible to the language model as API inputs; the agent must write and execute code to perceive image content and produce a deterministic output verified by unit tests.

AgentEye-Bench makes two design choices that set it apart from prior evaluations. First, rather than providing images through a vision API, we place images exclusively in the Docker container; to access image content, the agent must write and execute code, making tool-mediated perception a first-class evaluation target. Second, all tasks are evaluated with deterministic unit tests rather than LLM-as-judge scoring, ensuring that results are fully reproducible across runs and immune to evaluation noise from judge model variability (Ye et al., 2026). Together, these choices allow AgentEye-Bench to test image-grounded agentic reasoning in a controlled, reproducible setting that no existing benchmark provides.

Through evaluation of several frontier models from Google, OpenAI, and Anthropic, we find that the best model routes reach only 57.8% Pass@1, with Data Reasoning from Image proving the most challenging category across all evaluated model families. In summary, our contributions are:

- We introduce AgentEye-Bench, a benchmark of 45 image-grounded agentic tasks evaluated with deterministic unit tests inside Docker containers, covering three cognitively distinct

forms of visual reasoning.

- We identify a structural gap between tool-augmented and visual benchmarks and design AgentEye-Bench to bridge it, with all tasks verified against reference solutions prior to inclusion.
- We evaluate eleven frontier model routes and provide verifier-grounded failure analysis, revealing that visual recovery errors most often cascade into reasoning or state-reconstruction failures.

2 Related Work

Large language models evolved from single-turn conversational systems (OpenAI, 2022) to autonomous agents capable of multi-step planning, tool invocation, and interaction with external environments (Yao et al., 2023; OpenAI, 2026a). Contemporary systems such as the GPT (OpenAI, 2026b), Gemini (Google DeepMind, 2026), and Claude (Anthropic, 2026) series can follow complex instructions, write and execute code, and coordinate long-horizon action sequences, driving demand for evaluation frameworks that go beyond static benchmarks. This shift spurred a growing body of agentic benchmarks designed to measure how well language model agents complete open-ended tasks in executable, realistic settings.

Tool-Augmented Agentic Benchmarks. Several benchmarks evaluate agents in sandboxed environments with deterministic unit-test verification but restrict task inputs to plain text. Terminal-Bench (Team, 2025, 2026b) presents 89 tasks covering compilation, system configuration, and binary analysis inside Docker containers. Toolathlon (Li et al., 2026a) spans 108 tasks across 32 categories of command-line software, and SkillsBench (Li et al., 2026b) covers 86 tasks across 11 technical domains. ClawsBench (Li et al., 2026c) targets simulated office workflows such as email and calendar management, verified against database snapshots. While these benchmarks thoroughly evaluate multi-step tool use, none require agents to perceive or reason over visual content.

Benchmarks with Visual Inputs. A separate line of work incorporates image inputs. MM-Bench (Liu et al., 2024) offers over 3,000 multiple-choice visual understanding questions; although answers can be automatically scored, it is a static QA benchmark that requires neither tool execution nor artifact generation. Claw-Eval (Ye et al., 2026) supports image, video, and document inputs inside Docker containers, but evaluates outputs with an LLM judge, making results non-deterministic. SWE-bench Multimodal (Yang et al., 2025) combines image inputs with code execution and unit-test verification across 617 JavaScript bug-fixing tasks, yet all images are web UI screenshots, limiting coverage to a single visual domain. OfficeQA Pro (Opsahl-Ong et al., 2026) provides document images with a Python tool chain, but fewer than 3% of tasks require visual reasoning as the primary challenge.

Positioning. AgentEye-Bench bridges the gap between these two traditions. Unlike tool-augmented benchmarks, every task requires the agent to extract information from an image; unlike existing visual benchmarks, all tasks demand programmatic tool use inside a sandboxed Docker environment and are verified by deterministic unit tests. Moreover, AgentEye-Bench covers three structurally distinct visual domains: code screenshots and flowcharts, mathematical formulas and data tables, and structured documents such as receipts and charts. The closest prior work, SWE-bench Multimodal, restricts images to web UI screenshots. Table 1 summarizes these distinctions.

3 Methodology

AgentEye-Bench is a benchmark for evaluating agents on tasks that require extracting image content through programmatic tool use and producing verifiable outputs in a sandboxed Docker environment. Tasks range from implementing an algorithm depicted in a flowchart, to evaluating a mathematical formula shown in an image, to computing totals from a receipt photograph.

3.1 Task Formulation

AgentEye-Bench task consists of a plain-text instruction, one or more image files, a Docker container, a set of unit tests, and a reference solution. The instruction describes what the agent must accomplish and provides the file paths of the image files within the container; images are not embedded in the instruction itself. The unit tests verify the task outcome by checking the contents of the agent's output file against a known ground truth; they do not inspect the agent's commands or intermediate outputs. This is intentional: AgentEye-

Benchmark	Image Input	Tool Use	Verifiable Output	Sandboxed
Terminal-Bench (Team, 2025)	✗	✓	✓	✓
Claw-Eval (Ye et al., 2026)	✓	✓	◇[1]	✓
Toolathlon (Li et al., 2026a)	◇[2]	✓	✓	✓
SkillsBench (Li et al., 2026b)	◇[2]	✓	✓	✓
ClawsBench (Li et al., 2026c)	✗	✓	✓	✓
MMBench (Liu et al., 2024)	✓	✗	✓	✗
MMMU-Pro (Yue et al., 2025)	✓	✗	✓	✗
SWE-bench Multimodal (Yang et al., 2025)	✓	✓	✓	◇[3]
OfficeQA Pro (Opsahl-Ong et al., 2026)	◇[2]	✓	✓	✓
AgentEye-Bench (Ours)	✓	✓	✓	✓

Table 1: Comparison of related benchmarks. ✓ = fully supported; ◇ = partially supported; ✗ = not supported. [1] Claw-Eval uses LLM-as-judge with human review, not purely deterministic. [2] A small minority of tasks involve images; the majority are text-only. [3] Docker isolation details are not fully specified in the original paper.

Bench is an outcome-driven benchmark where each agent is free to use any approach to extract information from the images and produce the required output.

AgentEye-Bench tasks are interactive and share a common image-to-solution abstraction. Once the instruction and Docker container are provided, the agent must use terminal tools to recover task-relevant information from image files, organize the recovered content into an intermediate task state, apply the task's rules or computations, and write a deterministic result to a designated output file. This abstraction is not a rigid execution trace: different tasks instantiate the stages differently, and some stages may be repeated or collapsed. For example, a Code task may recover a visual program and execute it, a Math task may recover formulas and variables before computing an answer, and a Data task may recover semi-structured records before producing a CSV or JSON file. Tasks follow the Harbor task format and are run using the Harbor harness (Team, 2026a) with Terminus-2 as the agent scaffold (Team, 2026b). Terminus-2 serves as a fixed scaffold; our evaluation focuses on the performance of different underlying LLM backends rather than on the scaffold itself.

3.2 Task Quality Control

AgentEye-Bench separates task-quality review from model scoring. Model performance is scored only by deterministic unit tests, but task inclusion uses a rubric-based review process adapted from Terminal-Bench (Team, 2026b). The first quality gate checks general terminal-benchmark validity: the reference solution must pass the verifier; an empty or no-op submission must fail; the verifier must be deterministic and outcome-based; the instruction must specify a clear final artifact; and the task must not leak the answer through filenames, helper scripts, logs, environment variables, or other accessible metadata.

The second quality gate checks AgentEye-specific visual-agentic validity. Each task must satisfy six criteria. First, success must require recovering a visual hidden state rather than using text-only files or prior knowledge. Second, the hidden state must not be redundantly exposed outside the image. Third, the recovered state must drive a terminal-native action such as implementing code, running a numerical computation, generating a structured file, mutating a file tree, or routing records. Fourth, the verifier must check enough of the final artifact to distinguish genuine state recovery from a superficial guess. Fifth, the image inputs must be reproducible and license-compatible. Sixth, visual ambiguity must be controlled so that there is a single verifier-compatible interpretation, or an explicitly documented tolerance when exact agreement is unreasonable.

These rubrics are used only to audit benchmark construction. They do not provide partial credit and they are not used as a subjective judge of model outputs.

3.3 Dataset Construction and Composition

AgentEye-Bench contains 45 tasks across three categories and 15 sub-task types, with one instance per difficulty level (Easy, Medium, Hard) per sub-type. All tasks were designed and implemented by the authors to ensure consistency in format, difficulty calibration, and ground-truth reliability. Tasks cover three cognitively distinct forms of image-grounded reasoning: Code Execution from Image, Math Reasoning from Image, and Data Reasoning from Im-

age, each requiring the agent to programmatically recover visual hidden state from a different class of visual input.

AgentEye-Bench tasks vary in their visual content and required reasoning. A code execution task presents a pseudocode snippet describing a two-layer neural network with swish and sigmoid activations; the agent must read the pseudocode from the image, implement and execute it using provided weight files, and write the resulting float to the output file. A harder code task shows rollout-planning rules, constraints, and decision logic across multiple screenshots; the agent must reconstruct the visual state-space optimization problem and write the selected rollout plan. A math reasoning task presents a physics formula for kinetic energy with a drag correction factor as an image, along with four parameter values; the agent must read the formula and parameters, compute the result, and write it to the output file rounded to four decimal places. Harder math tasks require recovering filtering, thermal-dose, sequence-decoding, or triangulation protocols from images, then producing CSV and summary artifacts. A data reasoning task provides a personal expense report as an image and asks the agent to extract transaction records and compute aggregated totals across specified categories and months. Other Data tasks require recovering values from dot-encoded tables, charts, diagrams, and forms, then producing CSV or JSON artifacts.

Table 2 summarizes the benchmark composition; the full sub-task taxonomy and all Easy/Medium/Hard instances are provided in Appendix 5 and Table 6. Difficulty is calibrated along three axes: the complexity of the visual input, the reasoning depth required after recovery, and the precision demanded by the verifier. Easy tasks usually involve a single image and one main transformation; Medium tasks add more fields, multi-step computation, or stricter schema requirements; Hard tasks often combine multiple images, composed logic, or longer state-reconstruction chains.

Category	Types	Tasks	E/M/H
Code Execution from Image	5	15	5/5/5
Math Reasoning from Image	5	15	5/5/5
Data Reasoning from Image	5	15	5/5/5

Table 2: AgentEye-Bench composition by category. Each category contains five sub-task types, with one Easy, one Medium, and one Hard task per type.

4 Experiments

4.1 Experimental Setup

Figure 3 illustrates the AgentEye-Bench evaluation pipeline. All experiments are conducted using Harbor (Team, 2026a), an open-source framework for evaluating agents in containerized environments. The experiments evaluate three questions: whether AgentEye-Bench separates current frontier model routes under a fixed scaffold, which task categories and difficulty levels expose the largest bottlenecks, and where failed trials break down. This mirrors the evaluation philosophy of Terminal-Bench (Team, 2026b): tasks are executed in isolated terminal environments, final outputs are checked by deterministic tests, and failures are analyzed to identify practical limits of agent execution.

For each trial, Harbor provisions an isolated Docker container containing the task images and supporting files under `/app/`. The agent receives a plain-text instruction that specifies the objective and the image file paths. The images are not passed directly to the language model as API inputs. After the agent terminates, Harbor uploads the task's unit tests to the container and executes them to produce a binary Pass@1 verdict.

We use Terminus-2 as a fixed scaffold for all models. Terminus-2 interacts with the language model through text-only messages, extracts shell commands from model responses, executes those commands inside the Docker container, and returns terminal output to the model. As a result, any image understanding must be mediated by tools that the agent runs in the container, such as OCR, image inspection, Python scripts, or hand-written parsers. We do not tune prompts, tools, timeouts, or task instructions per model. This fixed-scaffold setup makes the comparison about how well different language-model backends can drive the same terminal agent.

We evaluate eleven model routes from Anthropic, Google, and OpenAI. Each model is run once on each of the 45 tasks. The primary metric is Pass@1, computed as the percentage of tasks whose final unit tests pass. Agent timeouts are counted as failures, with a fixed per-task agent timeout of 1200 seconds for all model configurations.

Table 3: Pass@1 (%) on AgentEye-Bench under the fixed Terminus-2 scaffold. Higher Pass@1 is better. Timeout counts are `AgentTimeoutError` exceptions and are included as failures.

Model	Code Pass@1	Math Pass@1	Data Pass@1	Overall Pass@1	Timeouts
Claude Haiku 4.5	60.0	40.0	20.0	40.0	0
Claude Sonnet 4.6	53.3	46.7	53.3	51.1	2
Claude Opus 4.7	**80.0**	**53.3**	40.0	**57.8**	1
Gemini 3.1 Flash Lite	26.7	20.0	6.7	17.8	0
Gemini 3 Flash	33.3	46.7	33.3	37.8	1
Gemini 3.5 Flash	73.3	40.0	53.3	55.6	0
Gemini 3.1 Pro	60.0	**53.3**	**60.0**	**57.8**	0
GPT-5.4-nano	40.0	6.7	6.7	17.8	9
GPT-5.4-mini	20.0	20.0	13.3	17.8	15
GPT-5.4	60.0	**53.3**	13.3	42.2	3
GPT-5.5	66.7	46.7	53.3	55.6	1
Mean over 11 models	52.1	38.8	32.1	41.0	–

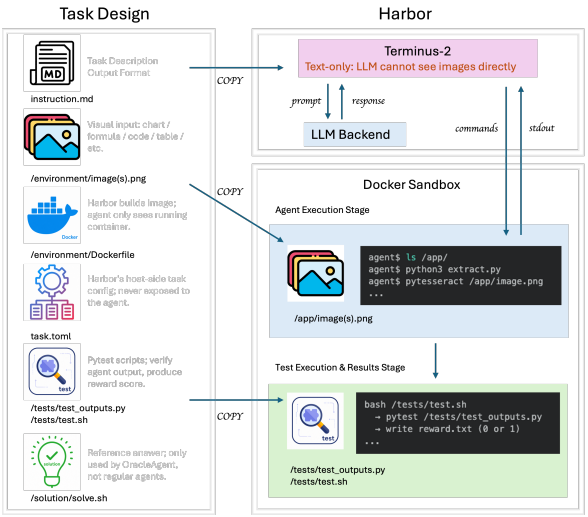

Figure 3: AgentEye-Bench evaluation pipeline. Harbor provisions an isolated Docker container per task; Terminus-2 exchanges text-only messages with the LLM and forwards bash commands to the container. The model must programmatically extract image content rather than perceiving images through a vision API. Harbor's unit tests determine the final Pass@1 score.

4.2 Main Results

Table 3 shows that no evaluated model solves AgentEye-Bench under the fixed Terminus-2 scaffold. Claude Opus 4.7 and Gemini 3.1 Pro are tied for the strongest overall result, each passing 26 of 45 tasks (57.8%). Gemini 3.5 Flash and GPT-5.5 follow at 55.6%, while Claude Sonnet 4.6 reaches 51.1%. The weakest routes, Gemini 3.1 Flash Lite, GPT-5.4-mini, and GPT-5.4-nano, each solve 17.8% overall. The 40.0 percentage-point spread between the strongest and weakest settings indicates that the benchmark separates model routes even under an identical scaffold, while the absolute scores show that the designed task set remains far from saturated.

4.3 Category and Difficulty Effects

The category breakdown reveals that Code tasks are easier than the other two categories, but all categories remain challenging. Averaged across the eleven model routes, Code tasks achieve 52.1% Pass@1, Math tasks achieve 38.8%, and Data tasks achieve 32.1%. The strongest category-specific score is Claude Opus 4.7 on Code tasks at 80.0%, whereas the best Data score is Gemini 3.1 Pro at 60.0%. This pattern suggests that recovering executable logic from images is more tractable for current terminal agents than preserving semi-structured visual records or mathematical state through downstream computation.

The Easy/Medium/Hard split provides an independent check that the benchmark has a meaningful difficulty gradient. Table 4 aggregates Pass@1 over all 495 model-task trials. Easy tasks average 66.1%, Medium tasks average 33.9%, and Hard tasks average 23.0%. The large Easy-to-Medium drop reflects the shift from localized visual recovery to cross-image state recovery and procedural reconstruction. Hard tasks further combine multiple images, longer state-reconstruction chains, stricter decision rules, or optimization-style reasoning after visual recovery.

4.4 Cost and Model Performance

Cost and timeout behavior show that practical agent performance is not a simple function of model size or price. Figure 4 shows that the best-performing region is occupied by several mid-to-high cost routes rather than by a single dominant model.

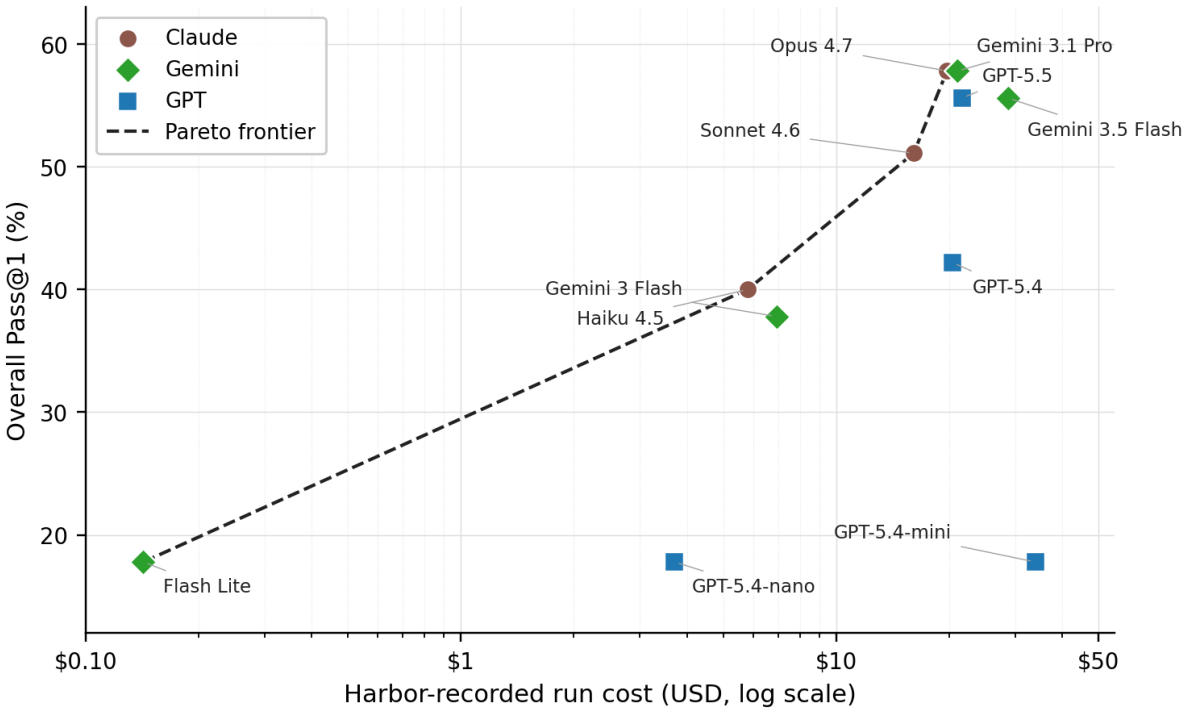

Figure 4: Cost-performance tradeoff on AgentEye-Bench under the fixed Terminus-2 scaffold. The x-axis reports Harbor-recorded API cost for each 45-task run on a log scale, and the y-axis reports overall Pass@1. The dashed line shows the Pareto frontier.

Table 4: Pass@1 (%) by task difficulty, averaged over all 495 model-task trials. Higher is better.

Difficulty	Pass@1
Easy	66.1
Medium	33.9
Hard	23.0

Claude Opus 4.7 and Gemini 3.1 Pro tie at 57.8% overall with recorded costs of $19.72 and $21.00, respectively. GPT-5.5 is close behind at 55.6% with $21.62 recorded cost, while Gemini 3.5 Flash matches GPT-5.5's Pass@1 at a higher recorded cost of $28.63. GPT-5.4-mini is a notable counterexample to a monotonic cost-performance relationship: it records the highest cost ($33.89) and 15 timeouts, but solves only 17.8% of tasks. This pattern reflects a timeout-amplification failure mode. Our scaffold enforces a wall-clock timeout, but not a strict interaction-turn, token, or cost budget; on tasks where the agent fails to converge, repeated unsuccessful interactions can accumulate large context and API costs before the wall-clock limit terminates the trial. This result should not be read as a claim about raw model competence in isolation; it shows that fixed-scaffold terminal-agent evaluation also measures execution efficiency, trajectory length, and the ability to stop or recover within a realistic budget.

4.5 Failure Analysis

Terminal-Bench reports failure taxonomies to connect aggregate scores with the practical causes of failed terminal-agent trajectories. We follow the same diagnostic goal, but report a verifier-grounded failure analysis rather than a human command-level annotation study. For each failed trial, we assign one outcome label from the final run status and one AgentEye root-cause label from the task category, subtype, exception type, and verifier outcome. The labels are therefore reproducible from benchmark logs, and can be interpreted as task-level failure surfaces.

The outcome labels separate *incorrect solutions*, where the agent finishes but the unit tests fail, from *timeouts*, where the agent does not complete within the run budget. The AgentEye root-cause labels describe the image-grounded stage most directly implicated by the failure: *visual recovery + reasoning/rule application* covers cases where a misread visual value or symbol propagates into an incorrect computation or decision; *visual recovery + state reconstruction* covers failures to preserve relations

Table 5: Failure breakdown over 292 strict failures from the final 45-task baseline matrix. Shares are percentages of all failed trials.

View	Failure group	Count	Share (%)
Task category	Data	112	38.4
	Math	101	34.6
	Code	79	27.1
Outcome	Incorrect solution	260	89.0
	Timeout	32	11.0
AgentEye root cause	Visual recovery + reasoning/rule application	138	47.3
	Visual recovery + state reconstruction	51	17.5
	Visual recovery + tool execution	36	12.3
	State reconstruction + reasoning/rule application	35	12.0
	Incomplete execution	32	11.0

such as table rows, chart series, graph edges, form states, or multi-image correspondences; *visual recovery + tool execution* covers failures where the agent's OCR, image-processing, parsing, or file-handling strategy prevents usable recovery; *state reconstruction + reasoning/rule application* covers cases where local elements are available but the global state or rule application is wrong; and *incomplete execution* covers timeouts and missing final artifacts.

Table 5 shows that most failures are completed but wrong attempts, not missing runs. Incorrect solutions account for 260 of 292 failures (89.0%), while timeouts account for 32 failures (11.0%). Data tasks contribute the largest share of failures (112 failures, 38.4%), followed by Math (101 failures, 34.6%) and Code (79 failures, 27.1%). This mirrors the category scores in Table 3: the central challenge is not merely invoking tools, but recovering image-grounded state accurately enough for downstream reasoning and deterministic verification.

The AgentEye-specific labels show that visual recovery errors usually become state or reasoning errors rather than isolated OCR mistakes. The largest group, visual recovery plus reasoning or rule application, contains 138 failures (47.3%). The second-largest group, visual recovery plus state reconstruction, contains 51 failures (17.5%). Together, these two groups account for 64.7% of all failures, showing that a single missed value, wrong symbol, omitted row, or broken correspondence can cascade into an incorrect final artifact. Purely execution-side visual failures account for 36 failures (12.3%), and incomplete execution accounts for the 32 timeout cases.

5 Conclusion

We presented AgentEye-Bench, a benchmark for evaluating language model agents on tasks that require programmatic extraction of image content and tool-augmented reasoning in sandboxed Docker environments. AgentEye-Bench comprises 45 tasks across three categories, all verified by deterministic unit tests, with images accessible to the agent only as container files rather than as API inputs. Experiments with eleven frontier model routes from Google, OpenAI, and Anthropic show that the best routes reach only 57.8% Pass@1, indicating that tool-mediated image perception remains a critical bottleneck for current language model agents. We release AgentEye-Bench as an open benchmark and plan to extend it with additional task categories, including PDF document understanding and video inputs, to support reproducible evaluation of future image-grounded agentic systems.

Limitations

Scale. AgentEye-Bench currently contains 45 tasks spanning 15 sub-task types across three categories: Code Execution from Image, Math Reasoning from Image, and Data Reasoning from Image. Each task requires a human-authored reference solution and verified ground truth, making manual construction the bottleneck. As a result, the current scale limits statistical power, and findings should be interpreted as initial evidence of model behavior rather than a comprehensive evaluation.

Tool-mediated image perception. In AgentEye-Bench, images are placed in the Docker container and the large language model receives only a plain-text instruction with file paths; to access image content, the agent must write and execute code

(e.g., using `pytesseract` or `Pillow`) within the container. This design tests tool-augmented image processing rather than native visual perception, and findings should not be directly generalized to benchmarks that pass images as multimodal API inputs.

Model coverage. We evaluate several models from Google, OpenAI, and Anthropic, selected for API availability and coverage of flagship-to-lightweight tiers. Open-source multimodal models (Qwen3.6, DeepSeek-V4, Kimi K2.6, etc.) were not included, and performance trends may differ for models with different training regimes.

Budget controls. Our current protocol uses a wall-clock timeout but does not impose a separate maximum number of interaction turns, total tokens, or per-task API cost. This choice exposes execution-efficiency failures, but it can also amplify the cost of non-convergent trajectories. Future versions should add turn, token, and cost budgets calibrated from successful-run distributions, while applying the same limits uniformly across all model routes.

References

Anthropic. 2026. Introducing Claude Opus 4.7.

Google DeepMind. 2026. Gemini 3.5 Flash.

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, and 2 others. 2026a. The Tool Decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *Preprint*, arXiv:2510.25726.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, and 22 others. 2026b. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *Preprint*, arXiv:2602.12670.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, Jiajun Bao, Yuanli Wang, Weixiang Yan, Yiyuan Li, and Han chung Lee. 2026c. ClawsBench: Evaluating capability and safety of llm productivity agents in simulated workspaces. *Preprint*, arXiv:2604.05172.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, and 1 others. 2024. MMBench: Is your multi-modal model an all-around player? In *European conference on computer vision*, pages 216–233. Springer.

OpenAI. 2022. Introducing ChatGPT.

OpenAI. 2026a. Codex | AI coding partner from OpenAI.

OpenAI. 2026b. Introducing GPT-5.5.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, Michael Bendersky, Matei Zaharia, and Xing Chen. 2026. OfficeQA Pro: An enterprise benchmark for end-to-end grounded reasoning. *Preprint*, arXiv:2603.08655.

Harbor Framework Team. 2026a. Harbor: A framework for evaluating and optimizing agents and models in container environments.

The Terminal-Bench Team. 2025. Terminal-Bench: A benchmark for ai agents in terminal environments.

The Terminal-Bench Team. 2026b. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *Preprint*, arXiv:2601.11868.

John Yang, Carlos E Jimenez, Alex L Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R Narasimhan, Diyi Yang, Sida Wang, and Ofir Press. 2025. SWE-bench Multimodal: Do ai systems generalize to visual software domains? In *International Conference on Learning Representations (ICLR)*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. 2026. Claw-Eval: Towards trustworthy evaluation of autonomous agents. *Preprint*, arXiv:2604.06132.

Xiang Yue, Tianyu Zheng, Yuansheng Ni, Yubo Wang, Kai Zhang, Shengbang Tong, Yuxuan Sun, Botao Yu, Ge Zhang, Huan Sun, Yu Su, Wenhu Chen, and Graham Neubig. 2025. MMMU-Pro: A more robust multi-discipline multimodal understanding benchmark. *Preprint*, arXiv:2409.02813.

Appendix A. Full Task Taxonomy

Sub-task Type	Easy	Medium	Hard
Code Execution from Image			
C1 *Code Extraction & Execution*	Neural net forward pass from pseudocode	Incident replay program from terminal screenshots	Multi-part Python risk-audit program from code screenshots
C2 *Algorithm from Flowchart*	Image-processing pipeline from flowchart	Constrained rollout selection from flowchart images	Rollout-planning flowchart with state-space optimization
C3 *Pseudocode Implementation*	Position-aware string encoding from pseudocode	Policy compiler from pseudocode and item constraints	Constrained rollout planner from visual pseudocode
C4 *Bug Identification & Fixing*	Bug in 2D prefix sum code	Patch a buggy risk reconciler using visual debugging evidence	Repair a transaction reconciler from visual debugging evidence
C5 *Config & Script Extraction*	Shell command from terminal screenshot	Log-processing shell pipeline from screenshots	Stateful incident replay pipeline from terminal screenshots
Math Reasoning from Image			
M1 *Formula Evaluation*	Kinetic energy formula with drag correction	Portfolio formulas and account rows to audit CSV	State-filter formulas and observations to trace CSV
M2 *Multi-step Calculation*	Multi-condition word problem from image	Thermal formulas and run rows to risk summaries	Thermal validation protocol and dose rollups
M3 *Sequence Pattern*	Two recurrence terms from image	Hidden-state sequence decoder from images	Noisy telemetry repair and decoded audit CSV
M4 *Statistics Calculation*	Seven descriptive statistics from exam scores	QC weighting and calibration statistics	Weighted statistics audit with calibration overrides
M5 *Numerical Reasoning*	Expression with factorial, logarithm, and square root	Routing cost rules and least-cost path checksum	Triangulation audit with corrected survey observations
Data Reasoning from Image			
D1 *Receipt / Invoice*	Amount bars from invoice image to JSON	Expense-report category and monthly summaries	Three-invoice billing, payment, and approval audit
D2 *Table*	Dot-encoded store table totals	Visual unit table joined with channel rates	Inventory and demand tables for reorder planning
D3 *Chart*	Bar-chart values and summary	Stacked-bar segment and quarterly summaries	Two line-chart series with trend diagnostics
D4 *Diagram*	Shortest route from colored graph diagram	Service-health dependency diagram restart plan	Critical path from project dependency diagram
D5 *Form*	Checked boxes and ratings to JSON	Inspection-form routing decisions	Multi-page claim forms and escalation artifacts

Table 6: Full AgentEye-Bench task taxonomy. Category headers group the 15 sub-task types; each row lists the Easy, Medium, and Hard instance for that sub-task.