RUNNING THE GAUNTLET: HARD AGENTIC TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

As agentic systems continue to evolve and are widely deployed in real-world scenarios, there is a growing demand to faithfully evaluate their capabilities. However, current benchmarks are typically built on popular applications with relatively simple tasks and focus on a narrow set of capabilities while overlooking broader dimensions, resulting in saturated performance on modern agents and failing to probe their limitations. To this end, we introduce *GauntletBench*, a web-based benchmark for evaluating agent generalisation in challenging scenarios, focusing on three underexplored capabilities (temporal perception, graphical understanding, and 3D reasoning), across five less-covered professional applications (`Video Editor`, `Workflow Builder`, `3D Modeller`, `Flight Analyser`, and `Circuit Designer`), each with 27 vision-intensive tasks (135 in total). Our benchmark provides a modular pipeline that comprises an environment compatible with both open- and closed-source agent frameworks, a controlled web-based application, a well-structured task suite, and an automated evaluation engine with diverse metrics. Contrary to widespread expectations, our empirical results reveal that frontier agentic systems remain far from achieving human-level performance. Even the state-of-the-art agent achieves only a 28.2% success rate on our GauntletBench, highlighting the limitations in these overlooked capabilities and generalisation. By comparison, non-expert human annotators achieve over 80% success on our challenging yet feasible tasks, revealing the substantial gap between current agent capabilities and those required for complex real-world scenarios[1].

1 INTRODUCTION

The rapid advancement of multimodal large language models (MLLMs) has significantly improved their cross-modal perception and reasoning capabilities (Anthropic, 2025; Google, 2025; OpenAI, 2025), establishing a strong foundation for the transition from passive assistants to intelligent agents (Yao et al., 2022b; Park et al., 2023; Shen et al., 2023). Agentic systems built on these models are now widely deployed for autonomous interactions and execution across diverse real-world domains, spanning from consumer-facing (Liu et al., 2025; Yao et al., 2022a; Zhou et al., 2024) to enterprise applications (Saxena et al., 2025; Mo et al., 2026). As modern agents evolve, they are expected to handle increasingly complex scenarios, necessitating more challenging evaluations to reflect their capabilities in such settings (Bommasani et al., 2021; Park et al., 2023).

While existing benchmarks provide valuable assessments, they often operate within highly homogeneous settings with relatively simplistic tasks, leading to saturated performance on modern agents and failing to reflect real-world complexity (Bommasani et al., 2021; Liu et al., 2025). Specifically, current benchmarks face two fundamental limitations: (i) they are built on popular applications or near-identical replicas (e.g., Amazon, Booking, or CRM systems (Garg et al., 2025; Mo et al., 2026)), with the tasks built upon them exhibiting high similarity and simplicity, making them insufficient to probe the limits of current agentic systems; and (ii) the task design typically targets a narrow set of capabilities, such as UI understanding, tool use, and long-term reasoning (Zhou et al., 2024; Wang et al., 2025), while failing to capture broader yet critical aspects of agentic performance. Moreover, these benchmarks often struggle to continuously support new agentic frameworks due to intricate pipelines, limited compatibility, and rigid configurations (Liu et al., 2025). These limitations lead to a mismatch between the evaluations provided by current benchmarks and the challenges of complex real-world scenarios, resulting in an inaccurate assessment of agentic capabilities.

[1]The homepage, code, and tasks of GauntletBench are available anonymously.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

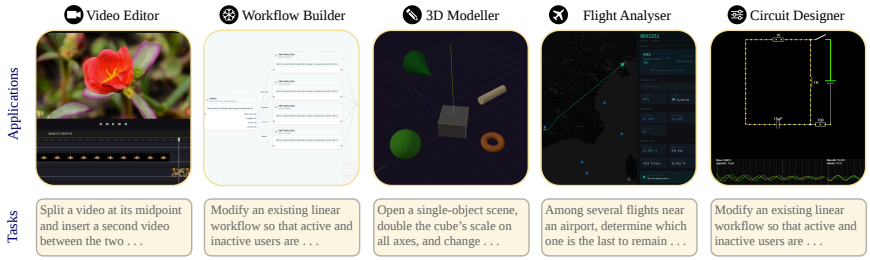


Figure 1. **Overview of GauntletBench.** Our benchmark contains five controlled web-based applications with 135 vision-intensive tasks. Detailed application descriptions are provided in Table 1, and full application screenshots are shown in Appendix A.

To bridge this gap, we propose GauntletBench, a web-based benchmark targeting underrepresented capabilities in less-covered applications via visually grounded tasks to evaluate agents' generalisation in challenging scenarios. Specifically, our benchmark focuses on three previously overlooked capabilities, namely temporal perception, graphical understanding, and 3D reasoning, which are increasingly important as agentic systems operate in more complex settings and expand across broader modalities. In our benchmark, we build five professional applications, including `Video Editor`, `Workflow Builder`, `3D Modeller`, `Flight Analyser`, and `Circuit Designer`, which are well-suited for evaluating our target capabilities and provide less familiar interfaces for assessing generalisation. Each application includes 20 challenging tasks across well-calibrated difficulty levels, covering a spectrum of capabilities from vision-based UI understanding to the targeted aspects of our benchmark.

To support compatibility, controllability, and reproducibility, our platform adopts a modular design consisting of four components: environments, applications, tasks, and the evaluation engine. The environments provide high-level observation and action interfaces, supporting both open-source and API-based MLLM agents, as well as closed-source agent frameworks. Built on these environments, we develop our applications using modern web technologies to support natural interaction patterns and flexible configuration. We provide tasks across three difficulty levels: easy, medium, and hard, following a standardised structure comprising application background, task description, and result format. Finally, we provide an automated evaluation engine that integrates tailored objective evaluators to assess success rate, MLLM-judged progress rate, and efficiency measures, supporting automated evaluation even for closed-source agent frameworks.

Our evaluations indicate that most open-source MLLM agents, such as Qwen-3-VL (Bai et al., 2025) and Llama-4-Maverick (Meta, 2026), achieve near-zero performance, and no API-based MLLM agents achieve success rates above 13.2% on GauntletBench. Even the state-of-the-art (SOTA) closed-source agentic frameworks like GPT-5.4 Computer Use (OpenAI, 2026a), Gemini Enterprise (OpenAI, 2025), and Claude Opus Computer Use (Anthropic, 2024) achieve success rates of only 7.4%, 25.7%, and 28.2%, respectively. In contrast, human annotators without specific domain knowledge of these applications achieve a success rate of over 80% on our benchmark while requiring 30% fewer steps than the frontier agentic system. This substantial gap in our challenging yet feasible benchmark indicates that current agentic systems still have critical limitations in these previously overlooked capabilities and generalisation, which are not faithfully reflected by existing benchmarks. We believe our benchmark provides a new perspective on evaluating agent generalisation in complex real-world scenarios, enabling a more accurate assessment to meet the growing demands of such settings. Our major contributions are summarised as follows:

- We introduce the GauntletBench, a challenging benchmark designed to evaluate agents' generalisation by targeting three underexplored capabilities: temporal perception, graphical understanding, and 3D reasoning.
- Our benchmark provides a web-based environment, comprising five less-covered professional applications, 135 challenging yet human-feasible tasks, and an automated evaluation engine.
- Empirical results show that SOTA agents achieve only a 28.2% success rate on GauntletBench, indicating that they remain far from human-level performance in complex real-world scenarios.

2 GauntletBench

We introduce GauntletBench, a challenging web-based benchmark targeting agent generalisation in complex real-world scenarios. Our platform follows a modular pipeline: a unified web-based

Table 1. **Applications in GauntletBench.** We provide anonymous links for these applications, along with their corresponding descriptions and targeted capability categories.

Application	Description	Target Capabilities
Video Editor	A timeline-based media editor for previewing clips and performing editing operations: trimming, audio overlay, colour adjustment, transitions, and text insertion.	Temporal perception; Graphical understanding
Workflow Builder	A node-based workflow builder for composing automation pipelines by adding, connecting, and configuring nodes, including trigger, branching, data, API, system, and AI nodes.	Graphical Understanding
3D Modeller	An interactive 3D scene workspace for navigation, object inspection, and spatial manipulation, including translation, rotation, scaling, and foreground property adjustment.	Graphical understanding; 3D reasoning
Flight Analyser	A flight radar map for searching aircraft, replaying historical traffic, inspecting trajectories, and extracting telemetry from detailed flight panels.	Temporal perception; Graphical understanding
Circuit Designer	A browser-based circuit environment, adapted from CircuitJS1 (Sharp, 2015), for composing and editing circuits consisting of logical and electrical components.	Graphical understanding

environment (Section 2.1), where agents interact with controlled applications (Section 2.2) to execute well-structured tasks (Section 2.3), with outcomes evaluated by an automated engine (Section 2.4).

2.1 Environments

Following standard formulations (Mo et al., 2026; Garg et al., 2025), we define a unified environment as $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ and $\mathcal{O}$ indicates the action and observation space, $\mathcal{T}$ is the transition function, and $\mathcal{R}$ refers to the reward function. At each time step t, the agent receives an observation $o_t \in \mathcal{O}$, executes an action $a_t \in \mathcal{A}$, transitions to the next state according to $s_{t+1} = \mathcal{T}(s_t, a_t)$, and use a binary reward $r_t \in \{0, 1\}$ to evaluate task success.

Observation Space GauntletBench exposes agents to all information naturally available through standard web interaction, without artificially restricting the observation space. In `3D Modeller`, `Flight Analyser`, and `Circuit Designer`, advanced functionality is implemented through canvas-based rendering, whose underlying implementations inherently do not expose structured representations through browser interfaces. As a result, in these three applications, agents can only use *Screenshots* to perceive and reason about task-relevant content. On the other hand, for the `Video Editor` and `Workflow Builder`, we additionally provide the *Accessibility Tree* to expose available semantic information about UI elements. We would like to emphasise that all tasks in GauntletBench can be completed using visual observations alone, and that this visual dependence reflects a common characteristic of real-world professional web applications, whose core interactive workspaces are often not directly exposed through structured browser interfaces.

Action Space Agents act by selecting $a_t \in \mathcal{A}$ from a set of high-level commands that emulate standard user interactions, such as `click`, `type`, `select`, and `scroll`, as well as navigation operations. These actions are executed via a browser automation layer (e.g., Playwright) and, when available, are optionally guided by elements identified from the accessibility tree.

Agent Interface We provide a unified interface that supports open-source and API-based MLLM agents, as well as closed-source agent frameworks. For MLLM agents, we leverage components of the REAL framework (Garg et al., 2025) to instantiate agents via MLLM-webpage interactions. For closed-source agent frameworks, the standardised interface we provide enables their execution without requiring access to the internal invocation process. This design decouples agent implementation from environment execution, enabling consistent evaluation across different systems.

2.2 Applications

To faithfully evaluate agent generalisation in complex real-world scenarios, our benchmark is designed to assess previously underexplored capabilities while introducing less familiar interfaces. To achieve these objectives, we identify five less-covered professional applications spanning diverse domains: `Video Editor`, `Workflow Builder`, `3D Modeller`, `Flight Analyser`, and `Circuit Design`.

As shown in Table 1, these applications not only support the evaluation of basic capabilities, such as UI understanding, tool use, and long-term reasoning, but are also well-suited for assessing our

targeted capabilities: temporal perception, graphical understanding, and 3D reasoning. Moreover, the interfaces of our applications differ substantially from those of popular applications, further supporting the evaluation of generalisation, as illustrated in Figure 1. These characteristics make our applications an effective testbed for assessing the limitations of modern agentic systems.

Website Technology Stack All applications in our benchmark are implemented using modern web technologies to ensure realistic interaction patterns. Specifically, the websites are implemented using JavaScript frameworks (e.g., React or Next.js), while their visual interfaces are rendered using canvas-based techniques. Due to the lack of structured information exposed by canvas-based interfaces, structured semantic observations are available only in two of the five applications. This implementation not only ensures consistent interaction logic, but also enables a genuine evaluation of the targeted visual perception capabilities, as the structured observations heavily relied upon by previous LLM-based agents (Garg et al., 2025) are not directly available.

2.3 Tasks

Our benchmark provides 27 challenging yet human-feasible tasks for each application, totalling 135 tasks. Each task is designed to evaluate one or more agent capabilities, either basic ones or those targeted by GauntletBench. We further support customised task design by enabling users to define tasks and upload ground truth, allowing seamless integration with our automated evaluation engine.

Task Complexity We categorise tasks into three levels: *easy*, *medium*, and *hard*. For each application, we construct 9 easy, 9 medium, and 9 hard tasks. Easy tasks require agents to follow explicit and straightforward instructions, whereas medium tasks involve composing multiple features or functionalities within the environment, requiring agents to coordinate several operations to achieve the goal. In contrast, hard tasks demand multi-step reasoning, fine-grained understanding, and the combination of multiple capabilities. All 135 tasks are detailed in Appendix G.

Task Structure Each task follows a standardised three-level structure: *application background*, *task description*, and *result format*. The application background consists of three components: an application overview (describing the application's purpose), features & capabilities (describing the available functionalities), and environment architecture & interaction paradigms (defining agent interaction with the interface). The task description specifies the objective through a clearly defined goal (describing the high-level objective) and steps (enumerating the conditions required for successful task completion). The result format defines the expected output format and evaluation target. We consider two output formats: information retrieval and action-based. For information retrieval tasks, it specifies the exact output format and required content. For action-based tasks, agents are required to export the application state as a JSON file or return a completion signal (e.g., a `done` JSON object) upon task completion. The full structure and per-application background are in Appendix F.

2.4 Evaluation Engine and Metrics

Our evaluation engine supports fully automated evaluation for both open- and closed-source agent frameworks with diverse evaluation metrics. To achieve this, we embed a tailored evaluator for each application, enabling the system to automatically execute the corresponding evaluation once an agent signals task completion. This design not only improves compatibility by enabling evaluation even for closed-source agent frameworks, but also enables more flexible and sophisticated evaluation metrics. As shown in Figure 2, the evaluator for `Video Editor` allows a 100 ms tolerance when matching start times and video durations between agent outputs and ground truth, as even human annotators cannot perform such edits with frame-level precision. Our evaluator also supports fuzzy matching for order-invariant tasks and colour differences. More importantly, this tailored evaluator provides more fine-grained localisation and detection of actual errors and their variants, such as property mismatches in `Flight Analyser`. This design makes success rate assessment more reasonable and human-aligned without compromising rigour, and similarly extends to progress rate and efficiency metrics. More case analyses of tailored evaluators on different applications can be found in Appendix E.

Success Rate We define the success rate (SR) as the fraction of tasks in which the agent successfully completes the task according to the tailored objective evaluator. Formally, let $\mathcal{D} = \{d_1, \ldots, d_N\}$ denote the set of N tasks and let $\mathbb{1}[\cdot]$ be the indicator function. For each task d_i, the objective evaluator returns a binary label $s_i \in \{0, 1\}$, where $s_i = 1$ if and only if the agent output matches the ground truth across all checkpoints. The SR is then given by $\mathrm{SR} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}[s_i = 1]$. This metric is a strict all-or-nothing criterion that reflects the agent's ability to fully satisfy all task requirements.

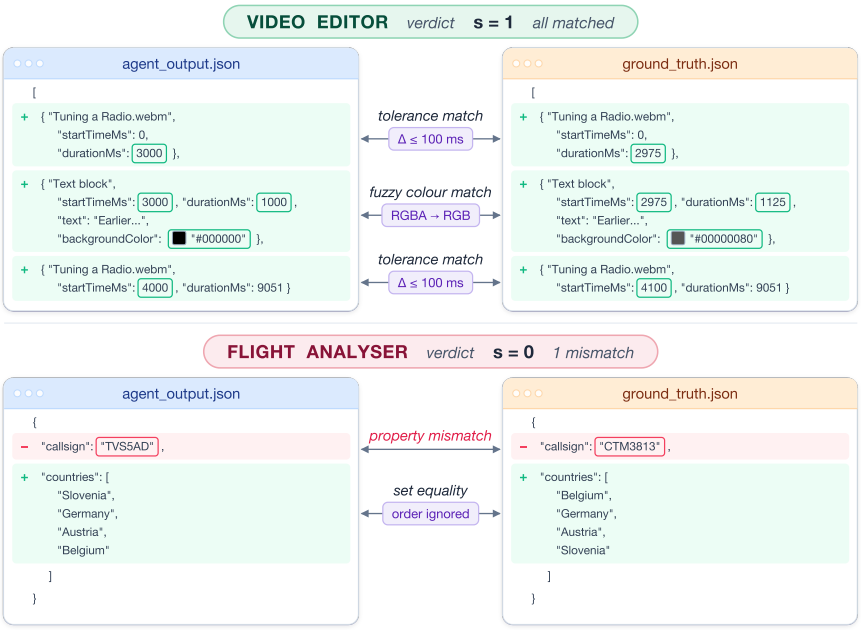


Figure 2. **Tailored Evaluator.** Each application uses a comparator that aligns agent output with ground truth via task-aware rules rather than strict string equality: `Flight Analyser` (top) combines exact property matching with order-invariant set equality on list fields, while `Video Editor` (bottom) applies a $\Delta \leq 100$ ms timing tolerance and format-normalised fuzzy colour matching.

Progress Rate To capture intermediate progress on partially completed tasks, we additionally report the progress rate (PR). For each task d_i, an MLLM-based judge assigns a score $p_i \in \{1, 2, 3, 4, 5\}$ based on visible final state and trajectory evidence, where 1 denotes no meaningful progress and 5 denotes fully successful completion. The detailed scoring criteria are described in Appendix C. The PR is computed as the mean judge score $\mathrm{PR} = \frac{1}{N}\sum_{i=1}^{N} p_i$. Unlike the SR, the PR rewards substantial progress toward the goal, providing a more fine-grained view of agent capability.

Efficiency Metrics Efficiency is another important aspect of agent performance. For each task d_i, let c_i be the total number of tokens consumed by the agent and a_i the number of interaction steps. We report consumed tokens (CT) and consumed steps (CS), defined as $\mathrm{CT} = \frac{1}{N}\sum_{i=1}^{N} c_i$ and $\mathrm{CS} = \frac{1}{N}\sum_{i=1}^{N} a_i$, respectively.

3 Experiment

3.1 Evaluation Setup

We evaluate GauntletBench using 14 frontier agents, spanning open-source and API-based MLLM agents as well as closed-source agent frameworks, assessing both task performance and efficiency. The details of human annotation are provided in Appendix A.

Open-Source MLLM Agents We evaluate widely used open-source MLLMs, including Gemma-3 (Team et al., 2025), Mistral-Large-3 (Mistral AI, 2025), Qwen3-VL (Bai et al., 2025), and Llama-4-Maverick (Meta, 2026). To verify the vision-intensive nature of our tasks, we also evaluate text-only LLM agents, Qwen3 (Yang et al., 2025), DeepSeek-V3.2 (DeepSeek-AI et al., 2025), and the proprietary Qwen-Max (Team, 2025), on the `Video Editor` and `Workflow Builder`, where minimal structured information is exposed.

API-Based MLLM Agents We also evaluate proprietary MLLMs, including GPT-5.4 (OpenAI, 2026b), Gemini-3.1-Pro (Google DeepMind, 2026), and Claude-Opus-4.6 (Anthropic, 2026).

Closed-Source Agent Frameworks Finally, we evaluate closed-source agent frameworks to demonstrate the challenge and compatibility of our benchmark, including GPT-5.4 Computer Use (GPT-5.4 CU) (OpenAI, 2026a), Gemini Enterprise (Google, 2026), and Claude-Opus-4.6 Computer Use (Claude-Opus-4.6 CU) (Anthropic, 2024).

Table 2. **Performance of Agents on GauntletBench.** Success rate (%, left) and progress rate (1-5, right). "-" marks screenshot-only applications, which are unsuitable for LLMs or vision-disabled MLLM agents. Results for each application are computed over 27 tasks and averaged over 3 independent runs. **Bold** and underlined entries denote the best and second-best results.

Agent	Success Rate (↑)						Progress Rate (↑)					
	Video Editor	Workflow Builder	3D Modeller	Flight Analyser	Circuit Designer	Avg.	Video Editor	Workflow Builder	3D Modeller	Flight Analyser	Circuit Designer	Avg.
Open-source MLLM Agent												
Gemma-3-27B	0.0	0.0	0.0	2.5	0.0	0.5	1.9	1.8	2.2	1.9	1.0	1.8
Mistral-Large-3	0.0	1.2	1.2	2.5	0.0	1.0	1.9	1.6	1.6	1.8	1.2	1.6
Qwen3-32B (LLM)	0.0	0.0	-	-	-	0.0	1.7	1.4	-	-	-	1.6
Qwen3-VL-32B	0.0	0.0	1.2	1.2	0.0	0.5	2.0	1.9	1.7	1.9	1.0	1.7
DeepSeek-V3.2 (LLM)	0.0	2.5	-	-	-	1.2	2.0	2.0	-	-	-	2.0
Llama-4-Maverick	0.0	0.0	0.0	3.7	0.0	0.7	2.0	1.8	1.8	2.0	1.0	1.7
API-based MLLM Agent												
Qwen-Max (LLM)	0.0	4.9	-	-	-	2.5	2.0	2.6	-	-	-	2.3
GPT-5.4 (w/o vision)	1.2	4.9	-	-	-	3.1	2.1	2.3	-	-	-	2.2
GPT-5.4	8.6	2.5	1.2	6.2	0.0	3.7	2.8	2.3	2.0	2.1	1.0	2.0
Claude-Opus-4.6	24.7	**24.7**	14.8	22.2	0.0	17.3	3.1	3.2	2.5	2.9	1.0	2.6
Gemini-3.1-Pro	4.9	9.9	30.9	21.0	**22.2**	17.8	2.5	3.0	3.3	3.0	3.0	3.0
Closed-source Agent Framework												
GPT-5.4 CU	12.3	0.0	1.2	16.0	7.4	7.4	2.9	2.3	2.5	2.6	1.2	2.3
Gemini Enterprise	37.0	12.3	**59.3**	13.6	6.2	25.7	3.6	3.4	3.9	2.5	2.5	3.2
Claude-Opus-4.6 CU	**54.3**	19.8	21.0	**30.9**	14.8	**28.2**	3.9	3.2	3.3	3.3	2.7	3.3
Human Annotation												
Human	83.9	82.7	77.8	76.5	79.6	80.1	4.3	4.0	3.8	4.3	4.2	4.1

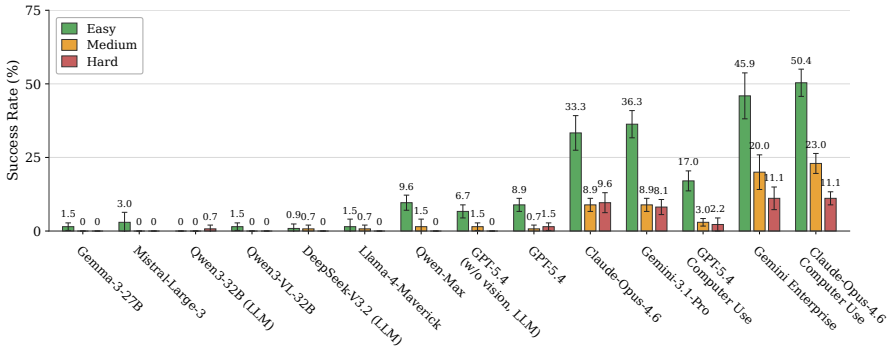


Figure 3. **Agent Success Rate Across Task Difficulty Levels.** Our benchmark contains 45, 45, and 45 tasks in the easy, medium, and hard difficulty levels, respectively. Results are averaged over three independent runs.

3.2 Performance Evaluation

Frontier Agentic Systems Remain Far from Human-Level Performance As shown in Table 2 (left panel), all open-source MLLM agents fail to surpass a 1.7% success rate on any single application, while most evaluated systems achieve 0.0% across all applications. The strongest agentic system, Claude-Opus-4.6 Computer Use, achieves only a 28.2% average success rate, followed by Gemini Enterprise at 25.7% and Gemini-3.1-Pro at 17.8%. In contrast, non-expert human participants achieve between 75% and 85% across all applications, indicating that the benchmark is challenging yet feasible. This substantial gap revealed by GauntletBench not only more faithfully reflects agent capabilities in complex real-world scenarios, but also exposes the limitations of existing benchmarks in evaluating the challenges and capabilities required for modern agents.

How Far Current Agents Remain from Reliable Performance? From Table 2 (right panel), we can observe that nearly all agents exhibit meaningful intermediate progress toward task completion, as reflected by greater than 1 progress rates. Claude-Opus-4.6 Computer Use even achieves an average progress rate exceeding 50% across all tasks, suggesting that failures often stem not from an inability to make progress, but from difficulty reliably handling the accumulating complexity of long-horizon task execution. This observation is further supported by Figure 3, where agents perform substantially better on easy tasks than on medium and hard ones, highlighting the limited robustness of current agents under increasing task complexity. These results provide an optimistic perspective: current agents are not entirely incapable of handling our targeted capabilities and challenging tasks, but their existing abilities remain insufficient for reliable performance.

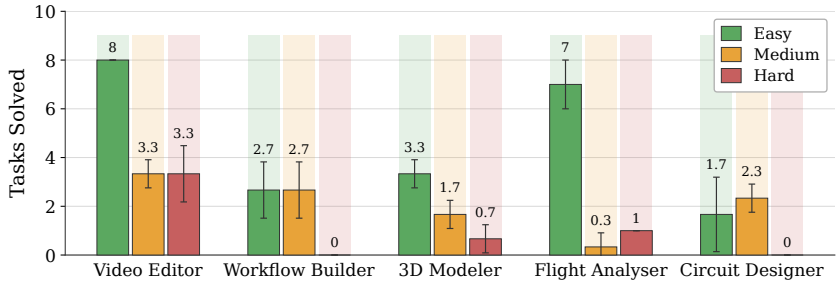


Figure 4. **Number of Solved Tasks Across Difficulty Levels.** Our benchmark contains 9, 9, and 9 tasks for the easy, medium, and hard difficulty levels of each application, respectively. Results are generated by Claude-Opus-4.6 Computer Use, and averaged over three independent runs.

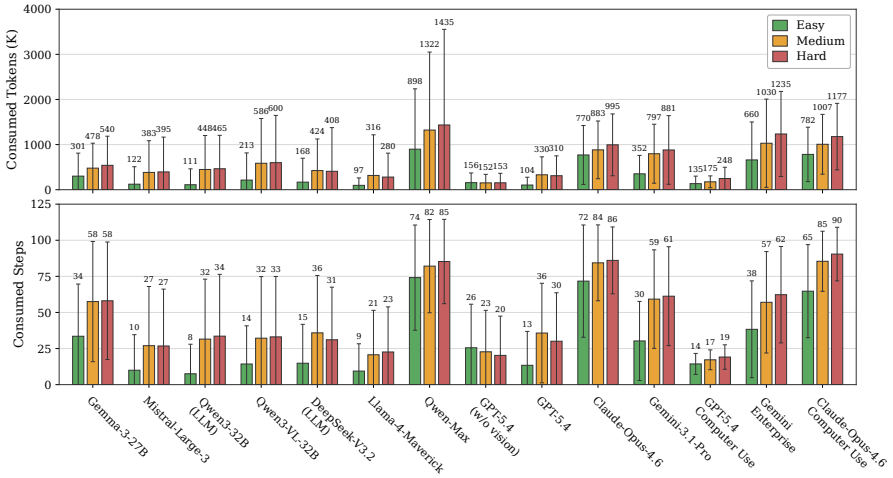


Figure 5. **Efficiency of Agents on GauntletBench.** The consumed tokens and consumed steps are reported in the top and bottom panels, respectively. Results for each application are computed over 27 tasks and averaged over three independent runs.

What Capabilities Do Current Agents Lack Most? As illustrated in Figure 4, the `3D Modeler` and `Circuit Designer` are the most challenging applications, with success rates at 19.2% and 14.8%, respectively. This suggests that current agents may already recognise 3D elements and graphical structures, yet still lack a deeper understanding of their relationships, interactions, and behaviours.

Does the Vision Modality Help Agents Complete Tasks? To understand whether MLLMs genuinely rely on vision to solve our intentionally vision-intensive tasks, we compare text-only LLM agents on the `Video Editor` and `Workflow Builder` applications, where structured information is accessible. From Table 2 (right panel), we can observe that relying solely on structured information enables agents to complete only a very small subset of tasks. In contrast, enabling visual inputs consistently improves agent performance, increasing average progress rates on these two applications by 22.5% and 13.7% for the Qwen and GPT families, respectively.

Which Reasoning Strategies Are Most Efficient? As shown in Figure 5, Qwen-Max incurs the highest token consumption, yet does not achieve a correspondingly strong success rate. In contrast, Claude Opus 4.6 Computer Use achieves the best overall performance with surprisingly low inference cost. Among token-similar models, Claude Opus 4.6 Computer Use consumes significantly more steps than Gemini Enterprise. While we cannot fully exclude the influence of model scale, intrinsic capabilities, and closed-source framework design, our results suggest that taking more fine-grained and incremental steps, a divide-and-conquer strategy, may be more effective and efficient for challenging tasks. Per-agent monetary cost and passes per dollar are reported in Appendix B.1.

3.3 Ablation Studies

The Impact of Model Scale Table 3 shows that increasing model scale consistently improves both success rate and progress rate. Notably, the largest model in each family also achieves the lowest token consumption (despite the higher cost of individual inference steps), while maintaining a comparable

Table 3. **The Impact of Model Size.** Two model families that share an architecture but differ in scale: GPT-5.4 (Nano, Mini, full) and Claude-4.6 (Haiku, Sonnet, Opus). Success rate (%) and progress rate (1-5) are computed over all 135 tasks; consumed tokens (in thousands, K) and steps over the 45 medium-difficulty tasks. Mean ± standard deviation over three independent runs. **Bold** and underlined entries denote the best and second-best results. Per-difficulty results are in Appendix B.

Model	Success Rate (↑)	Progress Rate (↑)	Consumed Tokens (↓)	Consumed Steps (↓)
GPT-5.4-Nano	0.0 ± 0.0	1.50 ± 0.04	521 ± 819	36.0 ± 40.7
GPT-5.4-Mini	0.0 ± 0.0	1.35 ± 0.16	449 ± 807	$\mathbf{27.3 \pm 42.7}$
GPT-5.4	2.3 ± 2.4	2.14 ± 0.84	$\mathbf{330 \pm 396}$	35.7 ± 34.3
Claude-Haiku-4.5	1.7 ± 2.4	2.01 ± 0.08	1204 ± 1317	74.3 ± 38.6
Claude-Sonnet-4.6	8.7 ± 1.2	2.21 ± 0.10	1496 ± 3448	84.8 ± 31.1
Claude-Opus-4.6	$\mathbf{12.3 \pm 3.3}$	$\mathbf{2.44 \pm 1.17}$	888 ± 633	85.1 ± 25.2

Table 4. **The Role of Reasoning.** Pairs of models sharing the same base model but differing in reasoning effort: GPT-5.4 with and without extended reasoning, Gemini-3.1-Pro at low versus high reasoning, and Claude-Opus-4.6 with and without thinking. Success rate (%) and progress rate (1-5) are reported over all 135 tasks; consumed tokens (in thousands, K) and steps over the 45 medium-difficulty tasks. Mean ± standard deviation over three independent runs. **Bold** and underlined entries denote the best and second-best results. Per-difficulty results are in Appendix B.

Model	Success Rate (↑)	Progress Rate (↑)	Consumed Tokens (↓)	Steps (↓)
GPT-5.4 w/o reasoning	0.7 ± 0.3	1.50 ± 0.68	$\mathbf{132 \pm 226}$	$\mathbf{10.3 \pm 19.6}$
GPT-5.4	2.3 ± 2.4	2.14 ± 0.90	330 ± 396	35.7 ± 34.3
Gemini-3.1-Pro Low-reasoning	6.3 ± 1.2	2.38 ± 0.93	326 ± 485	28.7 ± 29.0
Gemini-3.1-Pro High-reasoning	$\mathbf{13.2 \pm 0.7}$	$\mathbf{2.84 \pm 1.00}$	815 ± 652	59.7 ± 33.6
Claude-Opus-4.6	12.3 ± 3.3	2.44 ± 1.17	888 ± 633	85.1 ± 25.2
Claude-Opus-4.6 Thinking	10.3 ± 1.5	2.37 ± 1.12	947 ± 603	92.9 ± 21.2

number of interaction steps; consumed tokens and steps are reported on the medium-difficulty subset, where the comparison across scales is most informative. These results suggest that performance gains cannot come from additional steps alone, but require fundamentally stronger models.

The Role of Reasoning As shown in Table 4, enabling extended reasoning generally improves success rates for sufficiently capable base models, but also introduces a trade-off in token consumption and interaction steps, reported here on the medium-difficulty subset. We additionally evaluate Qwen-3-VL with and without extended reasoning, but observe no improvement. This suggests that extended reasoning is most effective when the underlying model has sufficient capacity to benefit from additional deliberation; for weaker models, the extra computation yields only limited gains.

Accuracy of MLLM-Judged Progress Rate To validate the accuracy of the MLLM-judged Progress Rate, we collect human annotations on the 1–5 rubric for five uniformly sampled trajectories per application, and compare them with the stage-2 (final-outcome) judge. As shown in Table 5, all evaluated MLLM judges produce results highly consistent with human annotations, suggesting that they provide reliable assessments. Moreover, GPT-5.1 achieves the strongest overall agreement and is therefore used for large-scale evaluation.

3.4 Error Analysis

We manually inspected a sample of failed trajectories and identified four recurring error categories, detailed in Appendix D: (1) task-completion misjudgment, where agents either corrupt an otherwise correct result by continuing to act or terminate prematurely; (2) grounding failures, where actions are applied to incorrect UI elements without being detected by the agent; (3) protocol violations, ranging from outputting an entire plan without executing actions to batching multiple steps into a single response; and (4) systematic failures in open-source models, including syntactically invalid outputs and infinite action loops, likely due to a lack of agentic browser-control fine-tuning data.

How Far Do Failures Get? To quantify these categories, we cross-tabulate the progress rate against the binary success label over all 4,500 evaluated trajectories (Appendix D.2). Of the 4,299 failures, 97.2% fall in the PR = 1-3 band, indicating that most failed runs break down before achieving substantial progress. The remaining tail is concentrated in the strongest systems: within its own

Table 5. **Human Agreement with MLLM-Judged Progress Rate.** Agreement on five uniformly sampled trajectories per application, reporting the mean PR, linearly weighted Cohen's κ, and macro F1, averaged over three independent runs. ⋆Human self-agreement is 1.00 by definition and shown as an upper bound.

Judge	Video Editor			Workflow Builder			3D Modeller			Flight Analyser			Circuit Designer			Avg.		
	PR	κ	F1	PR	κ	F1	PR	κ	F1	PR	κ	F1	PR	κ	F1	PR	κ	F1
GPT-5.1 (w/ stage-1)	**2.87**	**0.94**	**0.71**	**2.73**	**0.65**	**0.41**	**1.87**	0.43	0.24	**2.40**	**0.44**	0.20	**1.40**	**1.00**	**0.67**	**2.25**	**0.73**	**0.58**
GPT-5.1 (w/o stage-1)	2.53	0.70	0.48	2.47	0.56	0.29	**1.87**	**0.56**	0.33	2.47	0.56	0.29	**1.40**	**1.00**	**0.67**	2.05	0.62	0.48
Claude Opus 4.6	2.13	0.53	0.21	2.07	0.36	0.04	1.33	0.46	**0.35**	1.87	0.43	**0.31**	1.27	0.74	0.44	1.73	0.54	0.35
Claude Sonnet 4.6	2.00	0.41	0.08	2.13	0.40	0.21	1.40	0.09	0.17	1.80	0.38	0.29	1.20	0.62	0.33	1.71	0.44	0.25
Human Evaluation	2.80	1.00⋆	1.00⋆	3.00	1.00⋆	1.00⋆	1.80	1.00⋆	1.00⋆	2.60	1.00⋆	1.00⋆	1.40	1.00⋆	1.00⋆	2.32	1.00⋆	1.00⋆

failures, Claude-Opus-4.6 CU scores $\mathrm{PR} = 1$ on only 2 of 244 cases (0.8%) while 60.7% reach $\mathrm{PR} \geq 3$, whereas weaker agents such as Mistral-Large-3 collapse immediately on 51.3% of theirs. This separates two failure regimes: weaker agents fail for perceptual and comprehension reasons before making progress, while stronger agents fail at the execution level after substantial interaction. Consistently, protocol violations are confined to weaker agents — action-free trajectories account for 15.1% of Qwen3-32B, but never occur for Claude-Opus-4.6, Claude-Opus-4.6 CU, GPT-5.4 CU or Gemini Enterprise (Appendix D.3).

4 Related Work

Prior work on computer-use agents has developed benchmarks for web, mobile, and desktop interaction, ranging from synthetic browser tasks to realistic websites, operating-system environments, and enterprise workflows (Shi et al., 2017; Liu et al., 2018; Zhou et al., 2024; Koh et al., 2024; He et al., 2024; Deng et al., 2023; Xie et al., 2024; Rawles et al., 2025; Bonatti et al., 2024; Drouin et al., 2024; Garg et al., 2025). These benchmarks have been central to measuring progress, but much of the current evaluation landscape still emphasizes consumer-style information seeking, navigation, and form-based workflows. As agents improve, such settings increasingly risk saturating or underrepresenting the perceptual and spatial reasoning demands of professional software.

GauntletBench is complementary to this line of work. We keep the reproducibility advantages of browser-based environments while shifting the task distribution toward visually dense, dynamic, and specialist interfaces, including graphical editing, circuit design, flight tracking, and 3D manipulation. We also follow recent work on richer agentic assessment by combining structured checks with model-based partial-credit judging and efficiency metrics (Chiang et al., 2024; Zheng et al., 2023).

Several multimodal benchmarks probe the perceptual competences we target, including temporal video understanding (Li et al., 2024; 2025a), flowchart reading (Singh et al., 2024), geospatial reasoning (Dihan et al., 2025), and circuit knowledge (Zhao et al., 2025), but they query a model with questions rather than requiring an agent to act on a stateful interface. Closer to our setting, ScreenSpot-Pro (Li et al., 2025b), BlenderGym (Gu et al., 2025), and Spider2-V (Cao et al., 2024) evaluate single-step grounding on professional screenshots, 3D editing through a scripting API, and code-centric data workflows, respectively. GauntletBench instead embeds temporal, graphical, and 3D perception inside closed-loop, multi-step GUI interaction; the targeted capabilities are therefore underexplored not in multimodal understanding benchmarks at large, but in agentic benchmarks in particular. An extended discussion of prior work is provided in Appendix H.

5 Conclusion and Discussion

We introduce *GauntletBench*, a challenging web-based benchmark for evaluating agent generalisation in complex real-world scenarios. It targets previously overlooked capabilities, including temporal perception, graphical understanding, and 3D reasoning, through diverse professional applications and vision-intensive tasks. Empirical evaluations across frontier agentic systems reveal that current agents remain far from reliable real-world performance, highlighting substantial gaps in their ability to perceive, reason, and interact robustly across diverse environments. By providing a challenging and comprehensive evaluation of these capabilities, we hope GauntletBench can serve as a useful testbed for measuring progress and facilitating future research on robust perception, reasoning, planning, and interaction toward more generalisable and reliable agentic systems. Limitations, future work, and broader impacts are discussed in Appendix I.

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing methods, specifically in writing code for the benchmark applications and the evaluation harness. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on the research methodology or experiments, perform qualitative analysis, or interpret results; the remaining tasks with required disclosure, namely formulating or proving mathematical claims and translation, are not applicable to this work. Additionally, we used generative AI tools to edit the manuscript for readability and to polish the wording of the task descriptions and application background text shown to agents and human participants (Appendix F); we did not use them to identify or summarise related literature. We have reviewed all AI-assisted work: all AI-assisted code was reviewed and tested by the authors, every task description was manually verified against the corresponding application and ground truth, and all edited text was reviewed by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves a small-scale human evaluation (Appendix A) in which adult participants with a computer-science background completed the same tasks as the evaluated agents. Participants were briefed on the procedure before starting, and screenshots were recorded only within the tested application window and only with their consent; no personally identifiable or sensitive information was collected. All five applications are self-hosted, sandboxed web environments built or adapted by the authors, so neither agents nor participants interact with real users, real accounts, or third-party services, and agent actions have no effect outside the benchmark. The `Circuit Designer` application is adapted from the open-source CircuitJS1 simulator, which we credit accordingly, and all tasks and ground-truth annotations were authored by us. We release the benchmark publicly to support rigorous and transparent evaluation of agentic systems.

REPRODUCIBILITY STATEMENT

We did several steps to make GauntletBench and all reported results reproducible. The five benchmark applications are self-hosted, deterministic web applications that we make publicly accessible through the anonymous links in Table 1, so that every environment an agent or human interacts with can be revisited in exactly the state used in our experiments. The source code of our evaluation harness, including the unified agent interface, the browser automation layer, the tailored evaluators, and the MLLM-as-a-judge pipeline, together with the full specification and ground truth of all 135 tasks, is released at `https://anonymous.4open.science/r/gauntletbench-evaluation-harness/` and `https://anonymous.4open.science/r/gauntletbench-tasks/`, with a project homepage collecting these resources. The environment formulation, observation and action spaces, and agent interface are described in Section 2.1; the exact task prompt template and the per-application background blocks sent to every agent are reproduced verbatim in Appendix F, and every task is illustrated in Appendix G. The success-rate, progress-rate, and efficiency metrics are defined in Section 2.4; the application-specific matching rules of the tailored evaluators are documented in Appendix E, and the two-stage MLLM judge, including the models used at each stage, the screenshot-filtering thresholds, and the complete system and user prompts, is given in Appendix C. The full list of evaluated agents and their configurations is provided in Section 3.1, and all quantitative results are averaged over three independent runs. The human-evaluation protocol, including the annotation harness, participant briefing, per-task time limit, and workspace layout, is described in Appendix A.

REFERENCES

Anthropic. Developing a computer use model. `https://www.anthropic.com/research/developing-computer-use`, 2024.

Anthropic. System Card: Claude Opus 4.6. `https://www-cdn.anthropic.com/14e4fb01875d2a69f646fa5e574dea2b1c0ff7b5.pdf`, February 2026.

AI Anthropic. System card: Claude opus 4 & claude sonnet 4. *Claude-4 Model Card*, 2025.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui,

Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024. URL https://arxiv.org/abs/2409.08264.

Ruisheng Cao, Fangyu Lei, Haoyuan Wu, Jixuan Chen, Yeqiao Fu, Hongcheng Gao, Xinzhuang Xiong, Hanchong Zhang, Yuchen Mao, Wenjing Hu, Tianbao Xie, Hongshen Xu, Danyang Zhang, Sida Wang, Ruoxi Sun, Pengcheng Yin, Caiming Xiong, Ansong Ni, Qian Liu, Victor Zhong, Lu Chen, Kai Yu, and Tao Yu. Spider2-v: How far are multimodal agents from automating data science and engineering workflows?, 2024. URL https://arxiv.org/abs/2407.10956.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024. URL https://arxiv.org/abs/2403.04132.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M. S. Di, M. Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y. Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao, Bei Feng, Hui Li, J. L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R. J. Chen, R. L. Jin, S. S. Li, Shuang Zhou, Tianyu Sun, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Mingming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T. Wang, W. L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, and Zihua Qu. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023. URL https://arxiv.org/abs/2306.06070.

Mahir Labib Dihan, Md Tanvir Hassan, Md Tanvir Parvez, Md Hasebul Hasan, Md Almash Alam, Muhammad Aamir Cheema, Mohammed Eunus Ali, and Md Rizwan Parvez. Mapeval: A map-based evaluation of geo-spatial reasoning in foundation models, 2025. URL https://arxiv.org/abs/2501.00316.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

Divyansh Garg, Shaun VanWeelden, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Tomas Abraham, Michael Lara, Federico Lopez, James Liu, Atharva Gundawar, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Motwani. Real: Benchmarking autonomous agents on deterministic simulations of real websites, 2025. URL https://arxiv.org/abs/2504.11543.

Google. A new era of intelligence with Gemini 3. Google Blog, https://blog.google/products-and-platforms/products/gemini/gemini-3/, 2025. Accessed 16 January 2026.

Google. Computer Use. https://ai.google.dev/gemini-api/docs/computer-use, 2026. Google AI for Developers.

Google DeepMind. Gemini 3.1 Pro Model Card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-1-Pro-Model-Card.pdf, February 2026.

Yunqi Gu, Ian Huang, Jihyeon Je, Guandao Yang, and Leonidas Guibas. Blendergym: Benchmarking foundational model systems for graphics editing, 2025. URL https://arxiv.org/abs/2504.01786.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024. URL https://arxiv.org/abs/2401.13919.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 39648–39677. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/7cc1005ec73cfbaac9fa21192b622507-Paper-Conference.pdf.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks, 2024. URL https://arxiv.org/abs/2401.13649.

Bozheng Li, Yongliang Wu, Yi Lu, Jiashuo Yu, Licheng Tang, Jiawang Cao, Wenqing Zhu, Yuyang Sun, Jay Wu, and Wenbo Zhu. Veu-bench: Towards comprehensive understanding of video editing, 2025a. URL https://arxiv.org/abs/2504.17828.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use, 2025b. URL https://arxiv.org/abs/2504.07981.

Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, Limin Wang, and Yu Qiao. Mvbench: A comprehensive multi-modal video understanding benchmark, 2024. URL https://arxiv.org/abs/2311.17005.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *CoRR*, abs/1802.08802, 2018. URL http://arxiv.org/abs/1802.08802.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents, 2025. URL https://arxiv.org/abs/2308.03688.

Meta. Llama 4. https://www.llama.com/models/llama-4/, 2026.

Mistral AI. Introducing Mistral 3. https://mistral.ai/news/mistral-3, 2025.

Ying Mo, Yu Bai, Dapeng Sun, Yuqian Shi, Yukai Miao, Li Chen, and Dan Li. Entworld: A holistic environment and benchmark for verifiable enterprise gui agents. *arXiv preprint arXiv:2601.17722*, 2026.

OpenAI. Chatgpt atlas - release notes. https://help.openai.com/en/articles/12591856-chatgpt-atlas-release-notes, 2025. Accessed: 2026-05-06.

OpenAI. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

OpenAI. Computer Use. https://developers.openai.com/api/docs/guides/tools-computer-use, 2026a.

OpenAI. GPT-5.4 Thinking System Card. https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf, March 2026b.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents, 2025. URL https://arxiv.org/abs/2405.14573.

Divyanshu Saxena, Rishikesh Maurya, Xiaoxuan Ou, Gagan Somashekar, Shachee Mishra Gupta, Arun Iyer, Yu Kang, Chetan Bansal, Aditya Akella, and Saravan Rajmohan. Continuous benchmark generation for evaluating enterprise-scale llm agents. *arXiv preprint arXiv:2511.10049*, 2025.

Iain Sharp. CircuitJS1: Electronic circuit simulator in the browser. https://github.com/sharpie7/circuitjs1, 2015. GWT port of Paul Falstad's original Java applet. Open-source, GPL v2.0.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36: 38154–38180, 2023.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3135–3144. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/shi17a.html.

Shubhankar Singh, Purvi Chaurasia, Yerram Varun, Pranshu Pandya, Vatsal Gupta, Vivek Gupta, and Dan Roth. Flowvqa: Mapping multimodal logic in visual question answering with flowcharts, 2024. URL https://arxiv.org/abs/2406.19237.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Qwen Team. Qwen3-max: Just scale it, September 2025.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android, 2021. URL https://arxiv.org/abs/2105.13231.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. Odyssey-bench: Evaluating llm agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*, 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, 2024. URL https://arxiv.org/abs/2404.07972.

Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, Anjie Yang, Zhaoxuan Jin, Jianbo Deng, Philip Torr, Bernard Ghanem, and Guohao Li. Crab: Cross-environment agent benchmark for multimodal language model agents, 2025. URL https://arxiv.org/abs/2407.01511.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35: 20744–20757, 2022a.

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

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks?, 2024. URL https://arxiv.org/abs/2407.15711.

Chenchen Zhao, Zhengyuan Shi, Xiangyu Wen, Chengjie Liu, Yi Liu, Yunhao Zhou, Yuxiang Zhao, Hefei Feng, Yinan Zhu, Gwok-Waa Wan, Xin Cheng, Weiyu Chen, Yongqi Fu, Chujie Chen, Chenhao Xue, Guangyu Sun, Ying Wang, Yibo Lin, Jun Yang, Ning Xu, Xi Wang, and Qiang Xu. Mmcircuiteval: A comprehensive multimodal circuit-focused benchmark for evaluating llms, 2025. URL https://arxiv.org/abs/2507.19525.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024. URL https://arxiv.org/abs/2307.13854.

APPENDIX: TABLE OF CONTENTS

A Human Evaluation **17**

B Additional Ablation Studies on Efficiency **20**

B.1 Success Rate versus Monetary Cost 20

C Progress Rate Criteria **22**

C.1 LLM-as-a-Judge 22

C.1.1 Details 22

C.2 LLM-as-a-Judge Prompts 22

C.2.1 Stage 1 System Prompt 22

C.2.2 Stage 1 Task Prompt 23

C.2.3 Stage 2 System Prompt 24

C.2.4 Stage 2 Task Prompt 25

D Error Analysis **26**

D.1 Failure Taxonomy 26

D.2 How Far Do Failures Get? 26

D.3 How Often Do Protocol Failures Occur? 27

D.4 Representative Failure Cases 28

E Tailored Evaluation **29**

F Task Format **31**

F.1 Prompt Template 31

F.2 Per-Application Background Blocks 31

F.2.1 Video Editor 31

F.2.2 Workflow Builder 32

F.2.3 3D Modeller 33

F.2.4 Flight Analyser 34

F.2.5 Circuit Designer 34

G GauntletBench Task Illustrations **36**

G.1 Video Editor 36

G.2 Workflow Builder 48

G.3 3D Modeller 61

G.4 Flight Analyser 86

G.5 Circuit Designer 106

H Extended Related Work **125**

I Discussion **126**

A HUMAN EVALUATION

Harness Overview Our web-based **Human Evaluation Harness** mirrors the agent's interaction surface: the participant works directly inside the same web application an agent would control. It only adds an instrumentation layer around the application — task delivery, screenshot capture, and answer collection — so human trajectories are structurally identical to agent trajectories.

The participant flow has three stages:

- **Authentication** Each participant enters a single-session, application-specific access code.
- **Pre-session screen** Before the first task, the participant sees written instructions and a short demo video of the application (Figure 6). Both stay available throughout via the *Recall Instructions* and *Recall Demo Video* buttons; recording pauses while either is open.
- **Session workspace** During each task, the participant sees the application in the center, a header with the timer, task index, and recording status, and a sidebar with the task prompt and answer fields (Figure 7). They fill in the answer field(s) and click **Finish** to proceed.

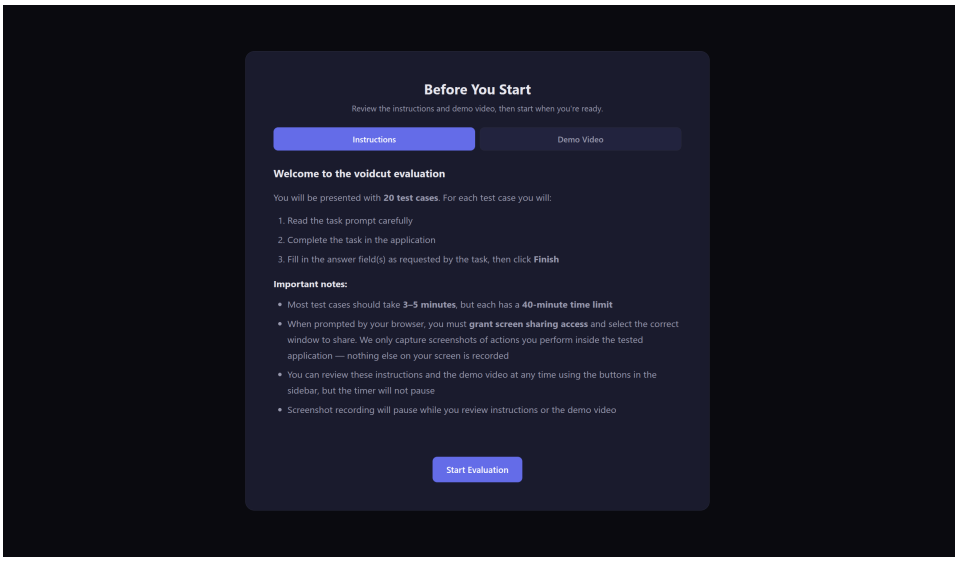
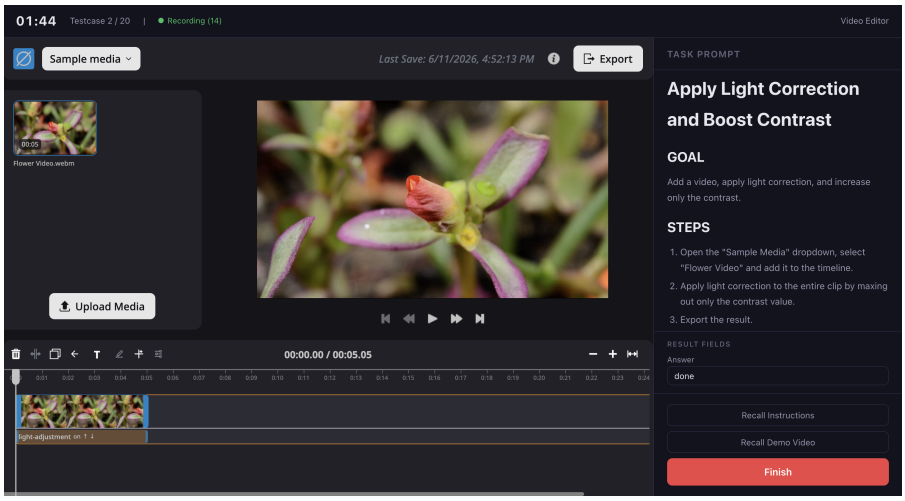
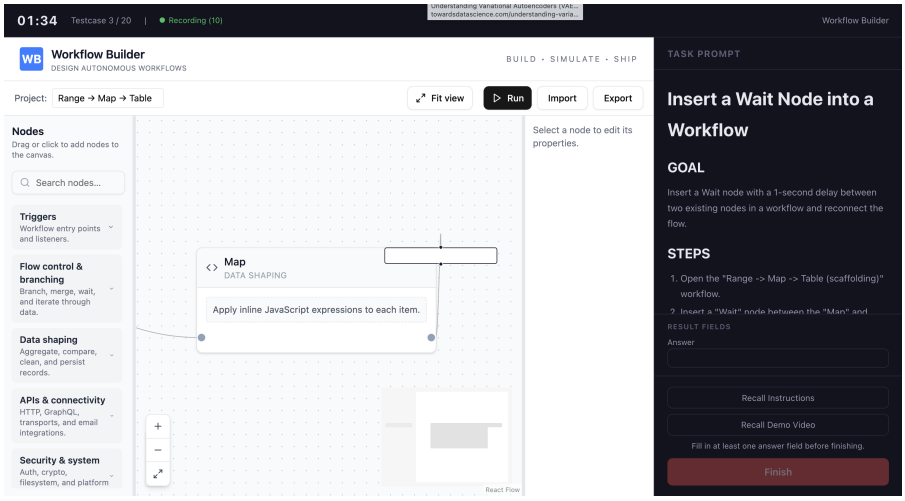


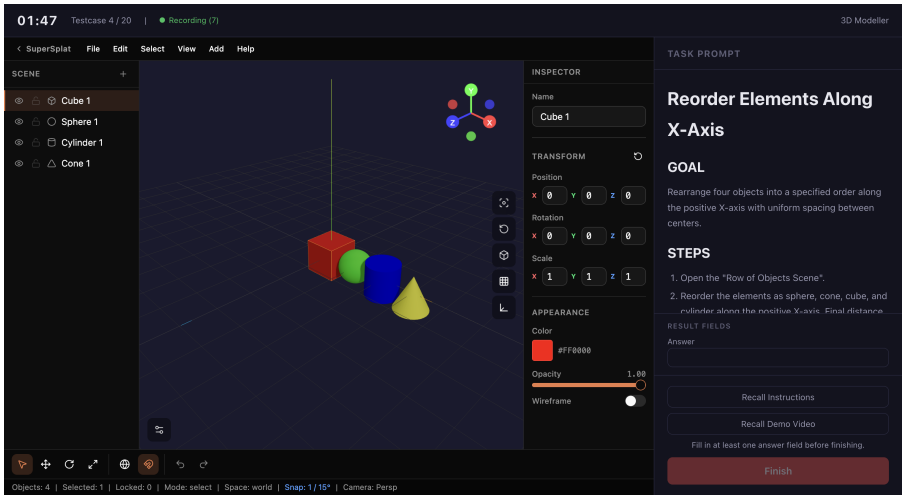
Figure 6. **Human Evaluation Harness: Pre-Session Briefing.** Before starting the first task, each participant is shown written instructions describing the application and the evaluation procedure, alongside an embedded demo video that illustrates basic interactions with the application. Both resources remain accessible during every task via the *Recall Instructions* and *Recall Demo Video* buttons; screenshot recording is paused while either is open to avoid leaking instructional content into the captured trajectory.



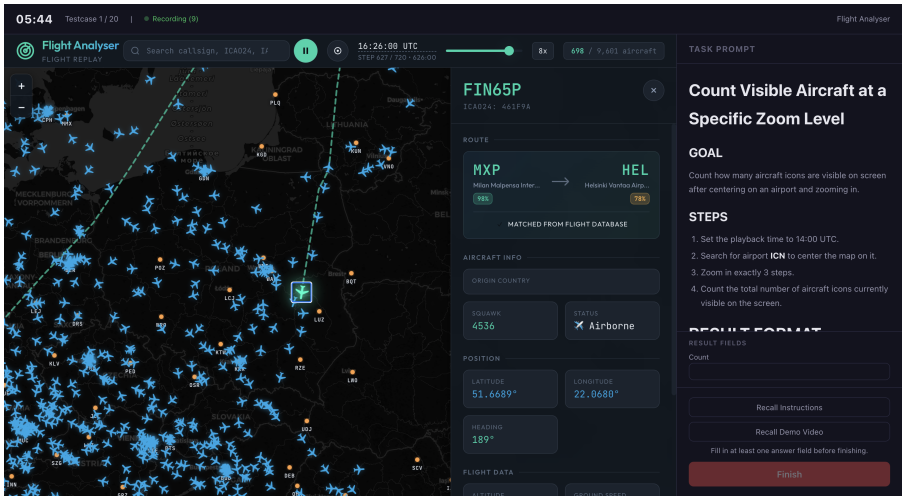
(a) Video Editor



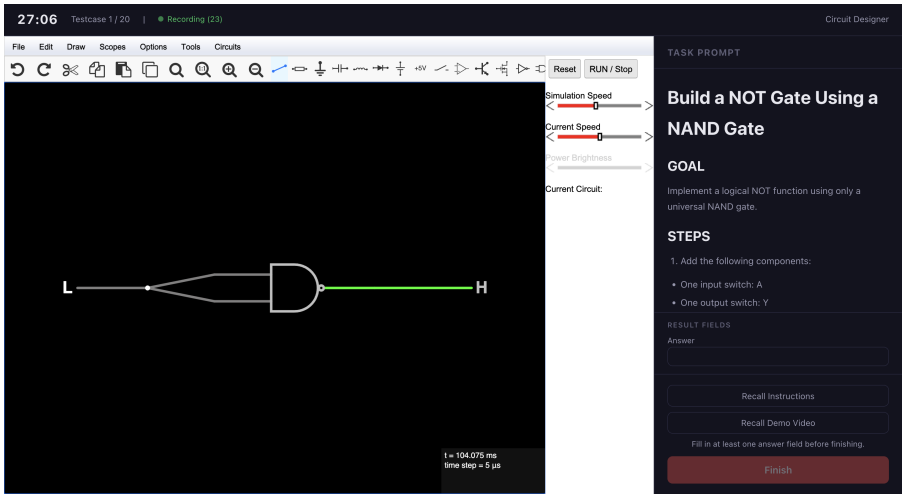
(b) Workflow Builder



(c) 3D Modeller



(d) Flight Analyser



(e) Circuit Designer

Figure 7. **Human Evaluation Harness: Per-Task Workspace Across All Five Applications.** Each panel shows the workspace for one application; the layout is identical across applications and mirrors the interaction surface presented to agents. The *center panel* embeds the live application that the participant directly operates. The *right sidebar* shows the task prompt (using the same unified template sent to agents, see Appendix F.1) and the answer field(s) the participant must fill in before clicking **Finish** to submit and advance to the next task. The *header* displays the countdown timer (40-minute hard limit), the current task index within the session, and the screenshot recording status.

Recruitment For this round, we have 3 participants per application (15 participants total, 300 tasks completed in total). Each participant has a CS background and prior hands-on experience with the type of tool their assigned application represents. The harness is mainly built to scale to multiple annotators per application.

Task Presentation Participants see the same prompt sent to agents, formatted with the unified template from Appendix F.1 (without application background part). No paraphrasing or extra hints are given.

Time Limit Each task has a hard **40-minute** limit. The timer keeps running while instructions or the demo video are reviewed mid-task.

No Retries Once **Finish** is clicked, the answer is final and the session goes to next task.

Recording With participant consent the harness captures screenshots of actions performed inside the tested application window only.

Scoring Human outputs are scored through the **same pipeline** as agent outputs, with no human-specific adjustments.

B Additional Ablation Studies on Efficiency

In this section, we report the consumed tokens and consumed steps for the two ablation studies of the main paper (the impact of model scale and the role of reasoning) broken down by difficulty bucket (easy, medium, hard) as well as the pooled overall value. These per-difficulty efficiency numbers are the source from which the medium-difficulty columns in Tables 3 and 4 of the main paper are drawn. We additionally report the monetary cost of evaluating each agent in Appendix B.1.

Table 6. **The Impact of Model Size: Consumed Tokens by Difficulty.** Consumed tokens (in thousands, K) for the two model families compared in Table 3: GPT-5.4 (Nano, Mini, full) and Claude-4.6 (Haiku, Sonnet, Opus). Values are reported per difficulty bucket (Easy: 45 tasks, Medium: 45 tasks, Hard: 45 tasks) and pooled overall (135 tasks), as mean ± standard deviation over three independent runs. Best results in each column are highlighted in bold, second-best are underlined.

Model	Easy	Medium	Hard	Overall
GPT-5.4-Nano	$\underline{34 \pm 76}$	521 ± 819	516 ± 796	504 ± 802
GPT-5.4-Mini	$\mathbf{23 \pm 79}$	$\underline{449 \pm 807}$	$\underline{415 \pm 757}$	$\underline{425 \pm 778}$
GPT-5.4	104 ± 173	$\mathbf{330 \pm 396}$	$\mathbf{310 \pm 438}$	$\mathbf{307 \pm 406}$
Claude-Haiku-4.5	697 ± 528	1204 ± 1317	1196 ± 1312	1150 ± 1267
Claude-Sonnet-4.6	974 ± 613	1496 ± 3448	2322 ± 5786	1815 ± 4548
Claude-Opus-4.6	770 ± 652	888 ± 633	995 ± 683	915 ± 659

Table 7. **The Impact of Model Size: Consumed Steps by Difficulty.** Consumed steps for the same models as Table 6, reported per difficulty bucket and pooled overall, as mean ± standard deviation over three independent runs. Best results in each column are highlighted in bold, second-best are underlined.

Model	Easy	Medium	Hard	Overall
GPT-5.4-Nano	$\underline{6.0 \pm 20.6}$	36.0 ± 40.7	41.9 ± 42.9	37.7 ± 41.8
GPT-5.4-Mini	$\mathbf{4.5 \pm 20.7}$	$\mathbf{27.3 \pm 42.7}$	$\mathbf{26.4 \pm 42.0}$	$\mathbf{26.2 \pm 42.1}$
GPT-5.4	13.4 ± 23.5	$\underline{35.7 \pm 34.3}$	$\underline{30.1 \pm 33.4}$	$\underline{31.8 \pm 33.4}$
Claude-Haiku-4.5	72.8 ± 40.9	74.3 ± 38.6	73.2 ± 40.7	73.7 ± 39.8
Claude-Sonnet-4.6	85.0 ± 29.8	84.8 ± 31.1	84.6 ± 30.6	84.7 ± 30.7
Claude-Opus-4.6	71.7 ± 38.9	85.1 ± 25.2	86.1 ± 23.1	83.0 ± 27.3

Table 8. **The Role of Reasoning: Consumed Tokens by Difficulty.** Consumed tokens (in thousands, K) for the three reasoning pairs compared in Table 4: GPT-5.4 with and without extended reasoning, Gemini-3.1-Pro at low versus high reasoning, and Claude-Opus-4.6 with and without thinking. Values are reported per difficulty bucket (Easy: 45 tasks, Medium: 45 tasks, Hard: 45 tasks) and pooled overall (135 tasks), as mean ± standard deviation over three independent runs. Best results in each column are highlighted in bold, second-best are underlined.

Model	Easy	Medium	Hard	Overall
GPT-5.4 w/o reasoning	$\underline{173 \pm 196}$	$\mathbf{132 \pm 226}$	$\mathbf{158 \pm 280}$	$\mathbf{157 \pm 233}$
GPT-5.4	$\mathbf{104 \pm 173}$	330 ± 396	$\underline{310 \pm 438}$	$\underline{248 \pm 370}$
Gemini-3.1-Pro Low-reasoning	238 ± 442	$\underline{326 \pm 485}$	396 ± 539	321 ± 494
Gemini-3.1-Pro High-reasoning	352 ± 409	815 ± 652	870 ± 731	677 ± 665
Claude-Opus-4.6	770 ± 652	888 ± 633	995 ± 683	883 ± 664
Claude-Opus-4.6 Thinking	776 ± 608	947 ± 603	985 ± 685	940 ± 642

B.1 Success Rate versus Monetary Cost

Consumed tokens and steps measure efficiency independently of pricing. To complement them, Table 10 reports the monetary cost of running each agent over the full benchmark and ranks agents

Table 9. **The Role of Reasoning: Consumed Steps by Difficulty.** Consumed steps for the same models as Table 8, reported per difficulty bucket and pooled overall, as mean ± standard deviation over three independent runs. Best results in each column are highlighted in bold, second-best are underlined.

Model	Easy	Medium	Hard	Overall
GPT-5.4 w/o reasoning	$\underline{23.5 \pm 25.7}$	$\mathbf{10.3 \pm 19.6}$	$\mathbf{10.5 \pm 20.8}$	$\mathbf{18.0 \pm 24.2}$
GPT-5.4	$\mathbf{13.4 \pm 23.5}$	35.7 ± 34.3	$\underline{30.1 \pm 33.4}$	$\underline{26.4 \pm 32.3}$
Gemini-3.1-Pro Low-reasoning	24.7 ± 23.6	$\underline{28.7 \pm 29.0}$	32.8 ± 29.1	28.8 ± 27.5
Gemini-3.1-Pro High-reasoning	30.3 ± 27.4	59.7 ± 33.6	61.9 ± 34.1	50.3 ± 35.0
Claude-Opus-4.6	71.7 ± 38.9	85.1 ± 25.2	86.1 ± 23.1	80.7 ± 30.8
Claude-Opus-4.6 Thinking	73.2 ± 37.1	92.9 ± 21.2	89.0 ± 24.8	88.6 ± 26.5

by the number of passed tasks per dollar. Agents with high success rates are not necessarily the most cost-efficient: Gemini Enterprise passes the most tasks per dollar, whereas Claude-Opus-4.6 and Gemini-3.1-Pro reach comparable success rates at roughly an order of magnitude higher cost.

Table 10. **Success Rate versus Monetary Cost.** Agents are ranked by passed tasks per dollar. Success rate (%) is averaged over three independent runs. Input and output tokens (in millions, M) and cost are summed over all three runs. Cost is computed from OpenRouter list prices as of September 2026. **Bold** and underlined entries denote the best and second-best passes per dollar.

Agent	Success Rate (%)	Input Tokens (M)	Output Tokens (M)	Cost ($)	Cost / Task ($)	Passes / $ (↑)
Gemini Enterprise	25.7	407.9	2.8	212.40	0.524	**0.490**
GPT-5.4 CU	7.4	76.1	0.3	195.37	0.482	<u>0.154</u>
Llama-4-Maverick	0.7	89.5	4.2	19.54	0.048	<u>0.154</u>
Gemma-3-27B	0.5	175.4	2.7	15.26	0.038	0.131
DeepSeek-V3.2 (LLM)	1.2	70.3	2.0	19.68	0.121	0.102
GPT-5.4 (w/o vision)	3.1	24.0	0.0	60.74	0.375	0.082
Gemini-3.1-Pro	17.8	223.0	51.0	1,058.66	2.614	0.068
Claude-Opus-4.6 CU	28.1	312.6	4.9	1,686.55	4.164	0.068
Mistral-Large-3	1.0	118.6	3.0	63.74	0.157	0.063
GPT-5.4	3.7	100.4	0.2	253.17	0.625	0.059
Qwen3-VL-32B	0.5	184.0	5.8	21.54	0.053	0.046
Claude-Opus-4.6	17.3	348.6	8.4	1,952.76	4.822	0.036
Qwen-Max (LLM)	2.5	306.9	3.1	251.37	1.552	0.016
Qwen3-32B (LLM)	0.0	25.8	7.5	4.16	0.026	0.000

C Progress Rate Criteria

C.1 LLM-as-a-Judge

For clarity, in this appendix we refer to the two prompted LLM calls as **Stage 1** and **Stage 2**. In our implementation, Stage 1 uses `gpt-4o-mini` and Stage 2 uses `gpt-5.1`.

C.1.1 Details

Given a trajectory of screenshots and actions, we first construct consecutive screenshot pairs. Before invoking the judge model, we remove pairs with no meaningful visual change using a lightweight image-difference check. Each screenshot is resized to 320×200 and converted to grayscale, and we compute three metrics between adjacent frames: RMSE, perceptual hash distance, and the fraction of changed pixels. A pair is forwarded to the LLM judge if at least one threshold is exceeded (RMSE ≥ 0.03, pHash ≥ 5.0, or changed-pixel fraction ≥ 0.01). This filtering step reduces unnecessary model calls and focuses the judge on visually informative transitions.

Stage 1: Pairwise Trajectory Parsing For each retained screenshot pair, the judge receives the task instruction, the success condition when available, the action executed between the two frames, the previous screenshot, the current screenshot, and a compact running summary of the visible state accumulated from earlier judged steps. We use `gpt-4o-mini` for this step because the task is local: it only requires identifying visible changes between adjacent screenshots and updating a concise state summary. The model is instructed to rely on visual evidence first and to use the action text only as supporting context. It outputs a structured record containing: (i) the visible action, (ii) a short summary of the visible change, (iii) a change type, (iv) task relevance, (v) whether the step represents positive, neutral, or negative progress, (vi) a confidence label, (vii) an uncertainty note, and (viii) an updated visible-state summary.

Stage 2: Final Outcome Judgment After processing the trajectory, we compress the Stage 1 outputs into a sequence of trajectory events and combine them with the final screenshot. The final judge also receives the task instruction, the success condition, the agent's final textual answer, and, when available, auxiliary reference signals such as `llm_judge_gt` and a binary objective evaluation result. We use `gpt-5.1` for this step because it requires integrating evidence across the trajectory and assigning a calibrated final score. The model outputs a score from 1 to 5, where 1 denotes no meaningful visible progress and 5 denotes clearly successful completion with the required details present and no important visible errors or redundant artifacts. The judge is instructed to be conservative: the score must be grounded primarily in the visible final state, while intermediate trajectory events and textual signals serve only as supporting evidence.

C.2 LLM-as-a-Judge Prompts

C.2.1 Stage 1 System Prompt

```
You are a careful visual trajectory parser for computer-use agent benchmarks.

Your task is to compare TWO consecutive screenshots from an agent trajectory and
↪  determine what visibly changed in the interface.

You must be conservative and evidence-based:
- Only report changes that are visually supported by the screenshots.
- Do not assume hidden state changes.
- Do not infer success from the action alone.
- If the change is ambiguous, say so.
- Prefer "unknown" or "no clear change" over guessing.

You are given:
1. The task instruction
2. The action taken between the screenshots
3. The previous screenshot
4. The current screenshot
5. Optionally, a compact prior visible-state summary from earlier confirmed
↪  steps
```

```
Your goals:
- Identify the visible UI change between the screenshots
- Judge whether that change is relevant to task completion
- Judge whether the step made positive progress, no progress, or negative
↪  progress
- Update the visible-state summary only when the evidence is sufficient
- Keep the visible-state summary compact and cumulative, focused only on durable
↪  facts that are still visible
- If the evidence is too weak to update the summary, carry the prior summary
↪  forward unchanged

Important:
- The prior visible-state summary is only contextual memory, not ground truth.
- If the screenshots contradict the prior summary, trust the screenshots.
- Never force consistency with prior results.
- Count something as completed only if it is visibly supported.

Return valid JSON only.
```

C.2.2 Stage 1 Task Prompt

```
Task instruction:
{TASK_PROMPT}

Action performed between screenshots:
{ACTION_TEXT}

Prior visible-state summary:
{PRIOR_STATE_SUMMARY_JSON}

Please compare the screenshots and return a strict JSON object with this schema:

{
  "visible_action": "<short textual description of the actual action if visually
  ↪  supported or consistent, otherwise 'unknown'>",
  "visible_change_type": "navigation | ui_interaction | content_update |
  ↪  object_added | object_removed | selection_changed | text_changed |
  ↪  dialog_opened | dialog_closed | error | no_clear_change | other",
  "task_relevance": "high | medium | low",
  "progress": "positive | neutral | negative",
  "confidence": "high | medium | low",
  "uncertainty_note": "<brief note if ambiguous, otherwise empty string>",
  "updated_visible_state_summary": "<compact visible-state summary to carry into
  ↪  later steps; if no reliable update is possible, repeat the prior summary>"
}

Rules:
- Base the answer on visual evidence from the screenshots first, action text
↪  second.
- If the action suggests something happened but the screenshots do not confirm
↪  it, say "no_clear_change" or mark confidence low.
- If the screenshots show a likely change but the exact object is unclear,
↪  describe it conservatively.
- Do not claim task success.
- Keep the updated visible-state summary short and durable. Include only facts
↪  still visible after the current screenshot.
- Do not output markdown.
```

C.2.3 STAGE 2 SYSTEM PROMPT

```
You are a careful final-outcome judge for computer-use agent benchmarks.

Your task is to judge how much of the task was visibly completed by the agent at
↪  the end of the trajectory.

You must be conservative, thorough, and evidence-based:
- Judge only from the provided screenshot-based events, the final screenshot,
↪  and the final agent answer.
- Do not assume hidden state changes unless they are supported by visible
↪  evidence.
- Do not treat the final answer as ground truth.
- If the final answer claims success but the screenshots do not support it,
↪  score based on the screenshots.
- If the evidence is ambiguous, prefer the lower score.
- Inspect fine-grained visible details that matter to task completion.
- Penalize clearly wrong, extra, duplicated, or redundant visible elements when
↪  the task requires a precise final state.
- Penalize missing required details even when the overall page or app looks
↪  close.

You are given:
1. The task instruction
2. The success condition
4. An objective evaluation result (`OBJECTIVE EVALUATION RESULT`) with binary
↪  value `0` or `1`
5. A sequence of screenshot-based events extracted from the trajectory
6. The final screenshot
7. The final agent answer

Scoring rubric:
- 5 = Full success clearly visible, including essential details; no meaningful
↪  visible mistakes or redundant artifacts remain
- 4 = Near success; most required state is achieved, but a minor non-essential
↪  issue remains
- 3 = Important partial progress, but at least one major requirement is missing,
↪  wrong, duplicated, or visibly inconsistent
- 2 = Limited progress; only a small portion of the task is visibly completed,
↪  or the final state contains major visible problems
- 1 = No meaningful visible progress at all

Important:
- Score visible task completion, not effort.
- Use the trajectory events as supporting context, but trust direct visual
↪  evidence from the screenshots most.
- Treat `OBJECTIVE EVALUATION RESULT` as supporting context. A value of `0` does
↪  not erase clearly visible success.
- If you cannot verify something - lean towards a `OBJECTIVE EVALUATION RESULT`.
↪  If it is `0`, be skeptical of claimed success. If it is `1`, assume success.
- However, if `OBJECTIVE EVALUATION RESULT` is provided and equals `0`, you must
↪  not assign score `5`.
- A correct-looking final answer without visible support should not receive a
↪  high score.
- A visibly successful final state can still score highly even if the final
↪  answer is weak or incomplete.
- High scores require correct details, not just approximate intent.
- If extra uploaded items, duplicate objects, wrong selections, leftover
↪  dialogs, incorrect quantities, or other redundant/wrong visible artifacts
↪  remain, lower the score accordingly.

Return valid JSON only.
```

C.2.4 STAGE 2 TASK PROMPT

```
Task instruction:
{TASK_PROMPT}

An OBJECTIVE EVALUATION RESULT (binary 0 or 1):
{OBJECTIVE_EVALUATION_RESULT}

Sequence of screenshot-based events extracted from the trajectory:
{TRAJECTORY_EVENTS_JSON}

Final agent answer:
{FINAL_AGENT_ANSWER}

Please inspect the final screenshot and judge partial success.

Return a strict JSON object with this schema:

{
  "score": <k>,
  "reason": "<brief explanation>"
}

Rules:
- Base the judgment on visible evidence from the trajectory events and the final
↪  screenshot first.
- Be strict about required details in the final visible state.
- Penalize clearly redundant, duplicated, extra, or wrong visible elements when
↪  they conflict with the intended final result.
- Use `OBJECTIVE EVALUATION RESULT` only as supporting context, not as proof by
↪  itself.
- If `OBJECTIVE EVALUATION RESULT` is provided and equals `0`, do not assign
↪  score `5`.
- Use the final agent answer only as supporting context, not as proof of
↪  success.
- If the evidence supports only partial completion, use the lower partial score
↪  that best matches the rubric.
- If the trajectory shows useful progress but the final screenshot does not
↪  preserve it, score the visible final state plus clearly supported trajectory
↪  progress conservatively.
- In the reason, mention the most important satisfied requirement and the most
↪  important missing, wrong, or redundant detail when applicable.
- Keep the reason short, concrete, and tied to the visible evidence.
- Do not output markdown.
```

D Error Analysis

Beyond the aggregate numbers, we also went through a sample of failed trajectories by hand to see what was actually going wrong. A few patterns kept showing up across almost all agents, and we group them into four categories in Appendix D.1. We then quantify how far failures get before breaking down (Appendix D.2), how often the protocol-level failures actually occur (Appendix D.3), and give verbatim examples of each (Appendix D.4).

D.1 Failure Taxonomy

Agents Do Not Know When To Stop A common failure mode we observed is that agents struggle to correctly judge when a task is actually done. On one side, we saw cases where the agent completes the task correctly, but then keeps going and ends up corrupting its own result. For example, on a 3D Editor task, an agent might set an object's position to the requested coordinates and at that point the task is essentially done — but instead of stopping, it continues to "check" the change by clicking around, dragging the object, or modifying some unrelated property, and the final state no longer matches the ground truth. On the other side, we also saw the opposite problem of early stopping, where the agent declares the task finished after only completing a part of the required steps, leaving the final state clearly incomplete. Both behaviours happen across most models we tested, but the issue is especially noticeable with GPT-based agents: they often declare in their textual output that the task is finished, while the actual final state is still far from satisfying the requirements. So there is a real mismatch between what the agent thinks it did and what actually ended up on the screen. This mismatch is measurable: among failed trajectories, an agent's own decision to stop carries almost no information about how much it achieved (Appendix D.2).

Grounding Failures A second big source of errors is grounding, i.e. mapping a high-level intent like "click the Map node" to the correct low-level action on the page. We observed agents clicking close to the right element but missing it, selecting a visually similar but functionally different element (like a neighbouring toolbar button), or typing the correct value into the wrong input field. What makes grounding errors especially bad is that the agent itself usually does not realise anything went wrong — the action executes fine from the browser's point of view, so the agent assumes everything is on track and keeps building on a state that is already broken.

Instruction-following Issues A third pattern is that agents mishandle the execution loop itself rather than the task. We distinguish three sub-modes. (i) *Plan-only responses*: the agent writes out its entire plan as one block of prose, often with placeholder element identifiers it never resolved, and then ends the trajectory without executing a single browser action. (ii) *Premature result format*: the agent emits the task's completion JSON on its first turn, before acting at all; because that object is the completion signal, the episode ends immediately with an empty final state. (iii) *Blind batching*: the agent packs many actions into a single turn. Our action space does permit multiple actions per turn, so this is not an illegal response, but it discards the intermediate observations the prompt asks the agent to reflect on, so any mis-grounded action in the middle of the batch goes unnoticed. These are not perception or reasoning failures — they are protocol failures — but they still cap how well an agent can do, no matter how strong the underlying model is. Importantly, they are concentrated in the weaker agents and are essentially absent from the strongest ones (Appendix D.3).

Open-source Models Failure Finally, the near-zero success rate of all open-source models is not just about model size. Most of them produced trajectories that were syntactically broken, looped on the same action over and over, or gave up after only a few steps. Our guess is that this is mostly because these models were probably never trained on agentic, multi-step browser-control data — their fine-tuning seems to be focused on single-turn instruction following and tool use in static settings, rather than long-horizon interaction with a stateful environment. Closing this gap likely needs not only bigger models but also targeted post-training on agent trajectories.

D.2 How Far Do Failures Get?

To quantify the relative frequency of these failure modes, we use the progress rate as a measure of how far a trajectory got before breaking down, and cross-tabulate it against the binary success label over the full evaluation corpus of 15 agents $\times$ 5 applications $\times$ 20 tasks $\times$ 3 seeds $=$ 4,500 trajectories.

Table 11 reports the joint distribution. Of the 4,299 failed trajectories, 97.2% fall in the PR = 1–3 band, so the large majority of failures do not make substantial progress before breaking down. A smaller tail of failures nonetheless reaches high progress: 120 trajectories (2.8%) are scored PR $\geq$ 4, i.e. near-successful runs that still fail the objective checker on a final detail.

Table 11. **Joint Distribution of Success Rate and Progress Rate.** Counts over all 4,500 evaluated trajectories (15 agents $\times$ 5 applications $\times$ 20 tasks $\times$ 3 seeds). Percentages in the bottom row are taken within the failed (SR = 0) population. Progress rates come from the stage-2 judge described in Appendix C. For 317 of the 4,500 cells (7.0%) the judged score was unavailable and was filled in from the same agent-application-task cell in a parallel seed, or floored at PR = 1 when no parallel observation existed; restricting the table to the 4,183 directly observed cells changes the PR = 1–3 share of failures from 97.2% to 97.1%.

	PR = 1	PR = 2	PR = 3	PR = 4	PR = 5	Total
SR = 1 (success)	5	1	33	26	136	201
SR = 0 (failure)	1240	2002	937	82	38	4299
% of failures	28.8	46.6	21.8	1.9	0.9	100.0

Two Distinct Failure Regimes This tail is not spread evenly across agents. Within each agent's own failures, the share that collapses immediately (PR = 1) falls monotonically with agent capability: 51.3% for Mistral-Large-3, 46.3% for Qwen3-32B and 35.5% for Llama-4-Maverick, against 28.1% for Claude-Opus-4.6, 5.0% for Gemini-3.1-Pro, 4.6% for Gemini Enterprise and 0.8% for Claude-Opus-4.6 CU. The share reaching meaningful progress (PR $\geq$ 3) moves the opposite way, from 2.3% for Qwen3-VL-32B and 5.0% for Qwen3-32B up to 52.7% for Gemini-3.1-Pro, 60.7% for Claude-Opus-4.6 CU and 68.3% for Gemini Enterprise. Concretely, Claude-Opus-4.6 CU scores PR = 1 on only 2 of its 244 failures, while 148 of them (60.7%) reach PR $\geq$ 3.

This indicates two qualitatively different failure regimes. For weaker agents, failures are predominantly perceptual or comprehension-related: the agent never establishes a workable understanding of the interface and the trajectory collapses at the outset. For stronger agents, failures overwhelmingly occur *after* substantial interaction progress, which points to an execution-level rather than an understanding-level limitation — consistent with the grounding and stopping failures described above, whose cost compounds over a long horizon.

Stopping Decisions Carry Little Information We can also separate failures by *why* the episode ended. Restricting to the 3,199 failed trajectories run through our own harness, where the termination flags are recorded, 24.7% ended because the agent itself declared the task complete and 71.6% because the step budget was exhausted. The two groups are almost indistinguishable in how much they actually achieved, with mean progress rates of 1.94 and 1.90 respectively. In other words, when a failing agent chooses to stop, that choice is essentially uncorrelated with whether it has accomplished anything — a direct quantitative counterpart to the stopping failures described in Appendix D.1.

D.3 How Often Do Protocol Failures Occur?

The clearest protocol failure to measure is a trajectory in which the agent produced a response but never executed a single browser action. Across all 5,132 recorded trajectories, 160 (3.1%) are of this kind, after excluding runs that failed for infrastructure reasons, so every case counted is a model behaviour. They are strongly concentrated in the weaker agents: Qwen3-32B (53 of 352 trajectories, 15.1%) and OpenAI o3-pro (41 of 299, 13.7%) together account for over half of all such cases, followed by Qwen-Max (4.8%), DeepSeek-V3.2 (4.2%), Llama-4-Maverick (3.2%), GPT-5.4 (3.0%), GPT-5.4 without vision (2.0%), Mistral-Large-3 (1.3%) and Qwen3-VL-32B (0.8%).

The two sub-modes separate cleanly by model family. OpenAI o3-pro, GPT-5.4 and GPT-5.4 without vision fail exclusively through *premature result format*, emitting the completion JSON on the first turn (41, 12 and 8 trajectories respectively). Llama-4-Maverick, DeepSeek-V3.2, Qwen-Max and Mistral-Large-3 fail exclusively through *plan-only responses*, writing an unexecuted multi-step plan (13, 13, 15 and 2 trajectories respectively); Qwen3-32B is the only agent that exhibits both (40 premature, 13 plan-only).

We emphasise that this failure mode is *absent* from the strongest agents: Claude-Opus-4.6, Claude-Opus-4.6 CU, GPT-5.4 CU, Gemini Enterprise and Gemma-3-27B never produced an action-free trajectory across 404, 404, 299, 405 and 298 recorded trajectories respectively, and Gemini-3.1-Pro

did so once in 397 (0.3%). We therefore do not claim that frontier models such as Claude-Opus-4.6 fail to follow instructions; the instruction-following category describes the weaker agents in our evaluation, which typically break down before making any meaningful progress.

D.4 Representative Failure Cases

We give one verbatim example of each protocol sub-mode, taken directly from the recorded trajectories. Elisions are marked `[...]`; nothing else is edited.

Case 1: Plan-Only Response Llama-4-Maverick on 3D Modeller, *Put the Cone Directly Above the Sphere*. The agent answers with a single 2,664-character response containing five code blocks. It grounds only the first step against the accessibility tree; for the remaining steps it writes placeholder identifiers it never resolved and symbolic values it never read from the scene, states outright that the steps are conceptual, and closes with the completion JSON. The episode ends at step 0 with zero browser actions executed and an empty final state.

````
To accomplish the task "Put the Cone Directly Above the Sphere," we need to
↪  follow the
steps outlined: Open the "Four Objects Scene," read the position of "Sphere 1,"
↪  and then
move "Cone 1" so that its center is directly above the center of "Sphere 1"
↪  [...]

1. **Open the "Four Objects Scene":** [...] The "Open" button is at [25], so its
↪  bid is
'25'. We will click on this button.

```python
click('25')
```

[...]

3. **Move "Cone 1" so that its center is directly above the center of "Sphere
↪  1":**
First, we need to select "Cone 1" to modify its position. Again, assuming we can
↪  access
"Cone 1" in the Scene Hierarchy Panel.

```python
click('bid_for_Cone_1')
```

[...]

```python
fill('bid_for_Cone_1_X_position', str(X))
fill('bid_for_Cone_1_Y_position', str(Y+3))
fill('bid_for_Cone_1_Z_position', str(Z))
```

Since the exact bids for the position fields of "Cone 1" and the values of X, Y,
↪  and Z
for "Sphere 1" are not provided, the above steps are conceptual.

After completing these steps, we will have repositioned "Cone 1" directly above
"Sphere 1" as required.

The final answer is:
```json
{"answer": "done"}
````
````

```
```

Case 2: Premature Result Format OpenAI o3-pro on 3D Modeller, *Reorder Elements Along X-Axis*. The complete trajectory is reproduced below: the agent's first and only response is the completion object, emitted before it has opened the scene or taken any action. Because that object is how an agent signals completion on action-based tasks, the episode terminates immediately with 0 steps. This exact shape accounts for all 41 action-free o3-pro trajectories (13.7% of its runs).

```
```json
{"answer": "done"}
```
```

Case 3: Blind Batching Gemini-3.1-Pro on Circuit Designer, *RLC DC Steady-State*. Here the agent does act, but 62 of its 80 turns issue more than one action at once, averaging 5.0 and peaking at 21 actions per turn. Its final turn, shown below, chains ten coordinate-level actions — component placement, keyboard mode switches and drags — with no observation between them, so a single misplaced drag corrupts everything that follows without the agent ever seeing it. Multiple actions per turn are permitted by our action space, so this response is legal; what it forfeits is the per-step feedback the prompt asks the agent to reflect on. The run exhausted its 80-step budget without ever emitting the required result object.

```
```python
mouse_click(500, 500)
keyboard_press('Escape')
keyboard_press('l')
mouse_drag_and_drop(200, 200, 200, 300)
keyboard_press('r')
mouse_drag_and_drop(200, 300, 200, 400)
keyboard_press('g')
mouse_drag_and_drop(200, 400, 200, 450)
keyboard_press(' ')
mouse_dblclick(200, 250)
```
```

E TAILORED EVALUATION

Each application domain uses a dedicated objective evaluator. A task solution is scored 1 (pass) only when *all* checked properties match; partial credit is not awarded.

Video Editor The evaluator decomposes both the ground-truth and predicted timelines into individual media blocks and matches them by content: media name, type, and timing. Matching is order-independent—blocks are sorted and paired globally rather than by position in the track list. Each matched block must have the correct start time, duration, and trim offsets within ± 1 s. Track placement is only enforced for text blocks that are layered on top of a video clip; standalone text is considered track-agnostic. For scenarios that additionally require visual effects (light adjustment or fade-out), the effect type, timing, and parameters are verified as extra constraints.

Workflow Builder The evaluator compares the node-and-edge structure of the exported workflow. Because agents assign their own node identifiers, nodes are matched by type and configuration payload using a greedy minimum-cost assignment. List-valued fields such as switch `cases` are compared order-independently. Edges are then verified using the resulting ground-truth-to-predicted ID mapping: an edge passes if the correct (source, target, output handle) triple exists in the predicted graph. Any extra or missing nodes or edges cause the test to fail.

3D Modeller Objects are matched by content rather than by name, again using greedy minimum-cost assignment. For each matched pair, object type and wireframe flag are checked exactly; position, rotation, and scale must agree within ± 0.15; color and opacity are compared exactly. A key special

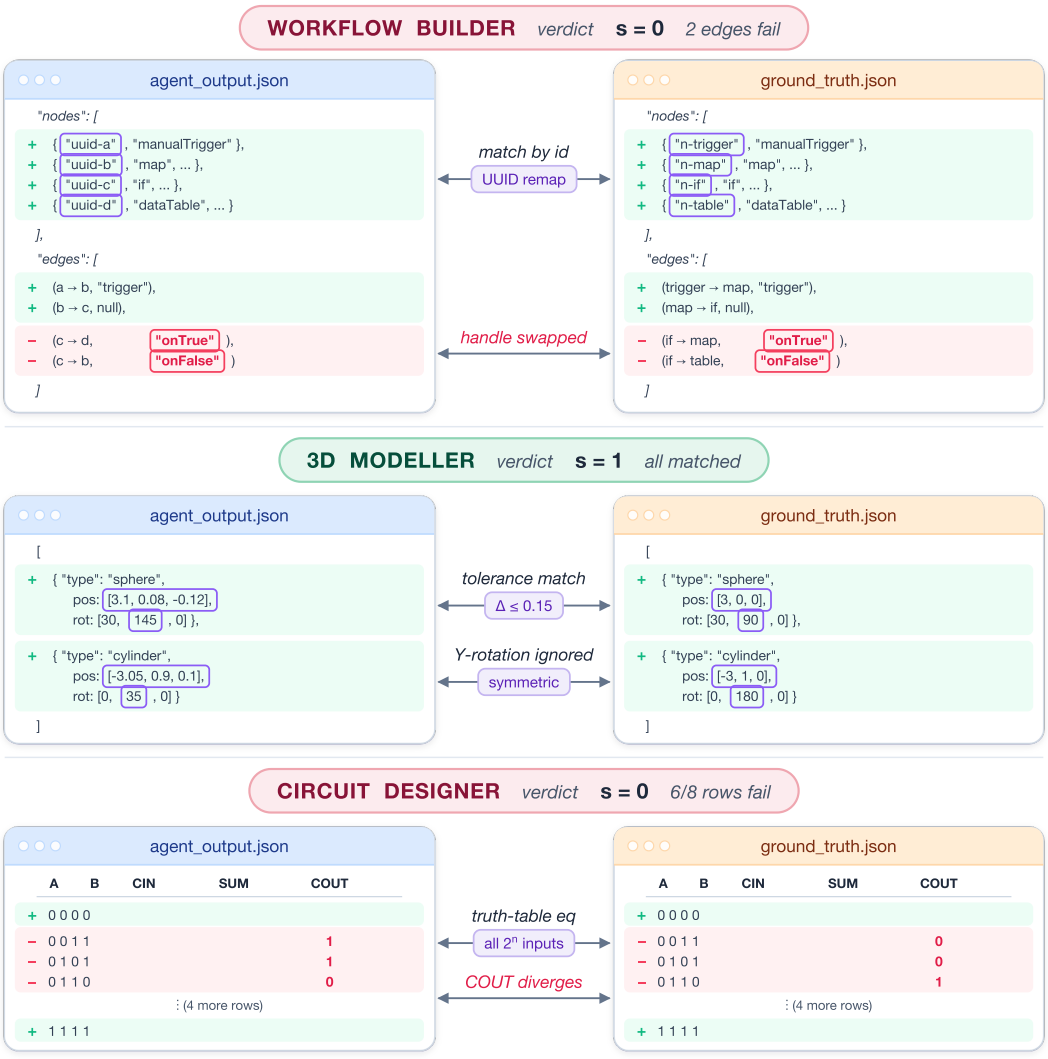


Figure 8. **Examples of tailored evaluators across the five environments.** The evaluator compares agent outputs against ground-truth JSON using application-specific matching rules: set equality and property matching for Flight Analysis, tolerance-based timestamp and colour matching for Video Editor, UUID remapping and edge matching for Workflow Builder, tolerance and symmetry-aware matching for 3D Modeller, and truth-table equivalence for Circuit Designer.

case: Y-axis rotation is *not* checked for rotationally symmetric shapes (sphere, cylinder, cone), since any rotation about their symmetry axis produces an identical visual result. Scene-level settings—background colour, field of view, and grid/axes visibility—are verified separately.

Flight Analyser The agent's answer is compared field-by-field against the ground truth. For tasks involving multiple aircraft, the aircraft objects are matched order-independently (all permutations are tried). All fields must match exactly (case-insensitive); time-of-day values are given a ± 1 minute tolerance to account for minor display rounding.

Circuit Designer For tasks that require building a logic circuit, correctness is determined by truth-table equivalence: both the ground-truth and predicted circuits are simulated over all input combinations, and the test passes only if their outputs agree on every row. For tasks that require reading analogue measurements, the agent's reported values are compared against the ground truth with a 5 % relative tolerance.

F TASK FORMAT

F.1 PROMPT TEMPLATE

To enable standardised, fair comparisons across various applications, we provide a unified prompt template for our tasks.

```
You are interacting with a web-based application.

# APPLICATION BACKGROUND

## Application Overview

[APPLICATION_NAME] is a web application used for [PRIMARY PURPOSE].
The main workspace/interface displays [MAIN UI SURFACE], where users can [CORE
↪  USER ACTIONS].

## Features & Capabilities

- [FEATURE 1]: [DESCRIPTION OF FEATURE 1]
- [FEATURE 2]: [DESCRIPTION OF FEATURE 2]
- [FEATURE 3]: [DESCRIPTION OF FEATURE 3]
- [CONTINUE AS NEEDED]
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms

To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- [UI MECHANIC 1]: [DESCRIPTION OF UI MECHANIC 1]
- [UI MECHANIC 2]: [DESCRIPTION OF UI MECHANIC 2]
- [UI MECHANIC 3]: [DESCRIPTION OF UI MECHANIC 3]
- [CONTINUE AS NEEDED]


# TASK
# [TASK_TITLE]

## GOAL
[HIGH-LEVEL OBJECTIVE]

## STEPS
1. [STEP 1]
2. [STEP 2]
3. [STEP 3]
4. [CONTINUE AS NEEDED]

# RESULT FORMAT

```json
{"[OUTPUT_KEY_1]": "[FINAL_VALUE_1]", "[OUTPUT_KEY_2]": "[FINAL_VALUE_2]",
↪  "..."}
```
```

F.2 PER-APPLICATION BACKGROUND BLOCKS

F.2.1 VIDEO EDITOR

```
# APPLICATION BACKGROUND
```

```
## Application Overview

Video Editor is a web application used for arranging and editing video, audio,
↪  and text.
The main workspace/interface displays a media panel on the left, a preview
↪  player in the center, and a timeline at the bottom, where You can drag,
↪  drop, cut, and enhance media clips to assemble a finished project.

## Features & Capabilities

- **Media Management**: You can select pre-loaded media from the "Sample media"
↪  dropdown.
- **Timeline Editing**: You can drag media to the timeline and use tools to
↪  Delete, Split, and Duplicate clips. A snapping feature helps align items
↪  seamlessly.
- **Text & Effects**: You can insert text blocks using the "T" tool or add and
↪  adjust visual effects (like Light Adjustment) using the "Add Effects" ("+"
↪  icon) and "Tweak Selected Effect" features.
- **Playback & View Control**: You can preview their edits, step through frames,
↪  and manipulate their view of the timeline using Zoom In, Zoom Out, and Fit
↪  To Screen tools (button on the right side above the timeline).
- **Exporting**: Once a project is finished, You can render and download the
↪  final video file.
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms

To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- **Drag-and-Drop Workflow**: Media items must first be loaded into the left
↪  panel and then physically dragged onto the bottom timeline area to be
↪  included in the edit.
- **Playhead Dependency**: Editing actions, specifically the "Split" tool,
↪  execute precisely at the current position of the playhead indicator on the
↪  timeline. Ensure the playhead is scrubbed to the correct timestamp before
↪  making a cut.
- **Selection State**: Clips and effect layers on the timeline must be
↪  explicitly clicked and highlighted before applying contextual actions like
↪  "Delete" or "Tweak Selected Effect".
- **Modal Interactions**: Adjusting effects opens a secondary modal window
↪  ("Tweak effect") containing sliders and save/cancel actions that must be
↪  completed before returning to the main workspace.
```

F.2.2 WORKFLOW BUILDER

```
# APPLICATION BACKGROUND

## Application Overview

Workflow Builder is a web application used for designing automation workflows
↪  and API-driven agents.
The main workspace/interface displays a visual node-based canvas, where users
↪  can drag, connect, and configure various functional nodes to build data
↪  pipelines and resilient agent behaviors.

## Features & Capabilities
```

```
- **Workflow Hub**: A central dashboard allowing users to create new agent
↪  workflows from scratch, import/paste JSON recipes, or utilize pre-built
↪  quick start templates (e.g., "API Data Pipeline", "Enterprise Data
↪  Pipeline").
- **Node Library**: A searchable, categorized left-hand sidebar containing
↪  functional blocks such as Triggers, Flow control & branching, Data shaping,
↪  APIs & connectivity, and Security.
- **Node Properties Editor**: A contextual right-hand sidebar that dynamically
↪  populates with a selected node's specific settings, allowing for parameter
↪  adjustments and advanced JSON configuration.
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms

To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- **Click-based Instantiation**: New workflow steps are added by searching for
↪  the desired node in the left-hand "Nodes" panel and clicking on it - this
↪  makes node appear directly on the canvas.
- **Edge Routing (Connections)**: The flow of execution is dictated by edges
↪  (lines). You can connect nodes by clicking and dragging from the output port
↪  (right side) of a preceding node to the input port (left side) of a
↪  subsequent node.
- **Edge Deletion**: To break a connection, you must select the existing edge on
↪  the canvas to open its menu in the right-hand panel, then click the "Delete
↪  edge" button.
```

F.2.3 3D MODELLER

```
# APPLICATION BACKGROUND

## Application Overview

3D Modeller is a web application used for viewing and manipulating 3D scenes,
↪  specifically designed for completing objective-based tasks.
The main workspace/interface displays a central 3D viewport, a scene hierarchy
↪  panel, an inspector panel, and a task prompt sidebar, where users can select
↪  3D objects, modify their transformations and appearance, and follow specific
↪  instructional steps.

## Features & Capabilities

- **Scene Management Dashboard:** A home screen allowing users to browse, open,
↪  and manage various pre-configured 3D environments (e.g., "Four Objects
↪  Scene", "Basic Shapes").
- **Scene Hierarchy Panel:** A left-sided menu that lists all active objects
↪  within the current 3D space, providing a straightforward way to locate and
↪  select specific items.
- **Inspector Panel:** A detailed right-sided menu that displays the properties
↪  of the currently selected object. Users can modify the object's `Name`,
↪  `Transform` attributes (Position, Rotation, Scale across X, Y, and Z axes),
↪  and `Appearance` (Color, Opacity, Wireframe toggle).
- **Precise Color Picker:** An appearance tool allowing users to alter object
↪  colors using either precise HEX codes or specific RGB (Red, Green, Blue)
↪  integer values ranging from 0-255.
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms
```

```
To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- **Selection-Dependent Editing:** The Inspector panel remains blank ("No object
↪  selected") until an object is actively clicked either in the 3D viewport or
↪  from the Scene list. Selecting a new object immediately updates the
↪  Inspector to reflect that specific object's current state.
- **Direct Input Manipulation:** While objects can visually be seen in the
↪  viewport, structural and aesthetic changes (like moving an object to an
↪  exact coordinate or setting a specific color) are achieved by typing
↪  specific numeric values directly into the input fields within the Inspector
↪  panel.
```

F.2.4 FLIGHT ANALYSER

```
# APPLICATION BACKGROUND

## Application Overview

Flight Analyser is a web application used for tracking, visualizing, and
↪  replaying aircraft flight data.
The main workspace/interface displays a map populated with aircraft icons
↪  alongside a task panel, where users can search for specific flights, adjust
↪  historical playback times, view flight trajectories, and extract detailed
↪  telemetry.

## Features & Capabilities

- **Temporal Navigation**: Adjust the temporal state of the map using the
↪  playback controls and timeline slider located at the top center of the
↪  screen to set exact UTC times.
- **Flight Search**: Locate specific aircraft by typing their callsign into the
↪  Search bar situated in the top left corner.
- **Flight Details View**: Selecting an aircraft highlights its flight path and
↪  opens a detailed panel overlaying the map, detailing its route,
↪  origin/destination, current status, and specific Flight Data (Altitude, Geo
↪  Altitude, Ground Speed, Vertical Rate).
- **Radius Tool**: Draw or clear dashed boundaries around target airports using
↪  the "Draw Radius Circle" tool (accessed via the target icon next to the
↪  pause button).
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms

To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- **Sequential Data Retrieval**: The interface displays detailed data for one
↪  flight at a time.
- **Panel Scrolling**: Specific telemetry metrics are located under the "Flight
↪  Data" section of the details panel. You may need to scroll down within the
↪  panel to locate fields like Geo Altitude or Ground Speed.
```

F.2.5 CIRCUIT DESIGNER

The Circuit application is adapted from CircuitJS1 (Sharp, 2015), an open-source browser-based circuit simulator ported from Paul Falstad's original Java applet by Iain Sharp.

```
# APPLICATION BACKGROUND
```

```
## Application Overview

Circuit Designer is a web application used for designing and simulating analog
↪  and digital circuits.
The main workspace displays a circuit canvas, where users can place components,
↪  wire them together, and observe live simulation behavior.

## Features & Capabilities

- **Component Placement:** Add parts from the **Draw** menu, top search button,
↪  or by right-clicking empty canvas space.
- **Circuit Wiring:** Connect terminals by clicking and dragging between
↪  component endpoints. Crossing wires do not connect unless joined at
↪  endpoints.
- **Component Editing:** Right-click a placed component to edit values,
↪  duplicate, or delete it.
- **Measurement:** Hover over a component to view live values in the
↪  bottom-right information panel.
- **General Rule:** Follow the task statement exactly. Only perform actions
↪  necessary to complete the requested task. Do not make unrelated changes.

## Environment Architecture & Interaction Paradigms

To successfully complete tasks in this environment, adhere to the following UI
↪  behaviors and state mechanics:

- **Right-Click Menus:** Most components are added through nested right-click
↪  menus on empty canvas space.
- **Property Dialogs:** Values such as resistance or voltage are changed through
↪  popup dialogs after selecting **Edit**.
- **Drag Wiring:** Connections are made by dragging from one terminal to
↪  another.
- **Logic Input Placement:** Draw logic inputs from right to left for proper
↪  orientation.
- **Live Updates:** Simulation runs continuously, so allow time for values to
↪  stabilize before measuring.
```

G GauntletBench Task Illustrations

This appendix lists all 135 tasks in GauntletBench across our five environments. Each table contains **No.** indicating the task identifier, **Task**, **Ground Truth**, and **Initial State**, where applicable.

Difficulty: indicates easy testcases; indicates medium-difficulty testcases; indicates hard testcases.

G.1 Video Editor

<table>
<tr><th>No.</th><th>Task</th></tr>
<tr><td>1</td><td>

```
# Trim a Video to First 5 Seconds

## GOAL
Add a video and trim it so that only the first 5 seconds remain.

## STEPS
1.  Open the "Sample Media" dropdown and select "White Candle" and add it to timeline.
2.  Trim the clip so that only the first 5 seconds remain.
3.  Export the result.

# RESULT FORMAT

‘‘‘json
{"answer":  "done"}
‘‘‘
```

Ground truth



</td></tr>
<tr><td>2</td><td>

```
# Apply Light Correction and Boost Contrast

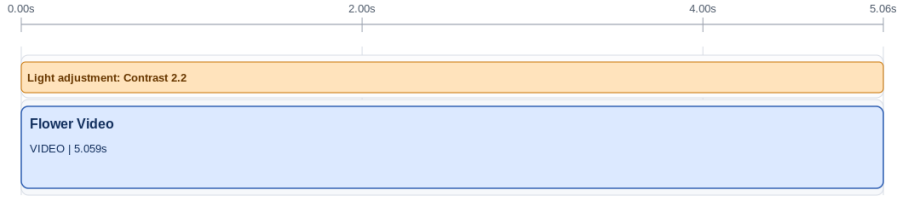
## GOAL
Add a video, apply light correction, and increase only the contrast.

## STEPS
1.  Open the "Sample Media" dropdown, select "Flower Video" and add it to the timeline.
2.  Apply light correction to the entire clip by maxing out only the contrast value.
3.  Export the result.

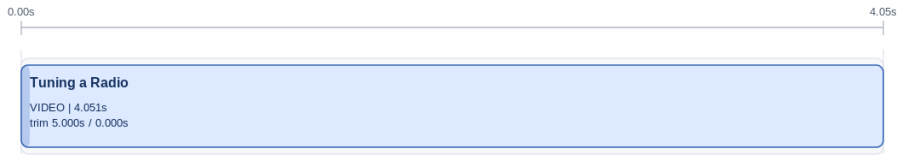
# RESULT FORMAT

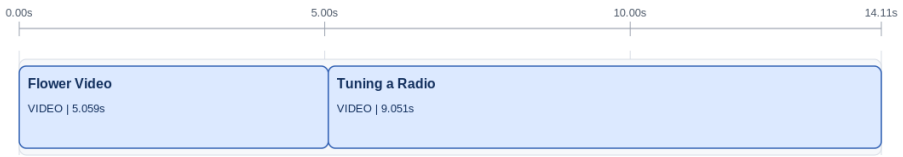
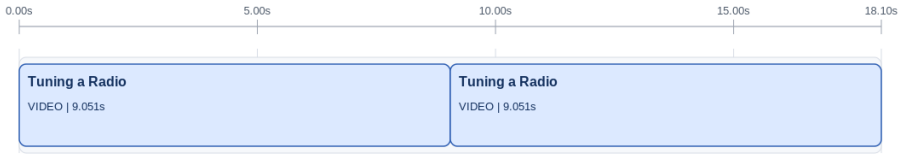
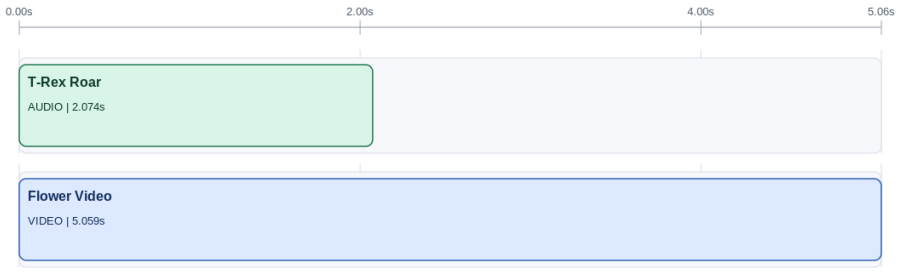
‘‘‘json
{"answer":  "done"}
‘‘‘
```

Ground truth



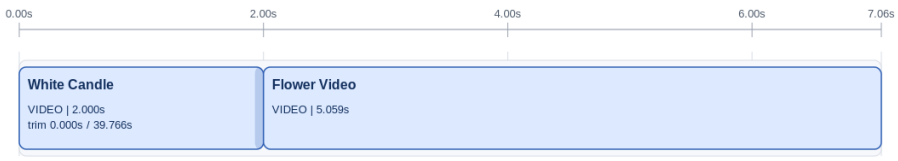
</td></tr>
</table>

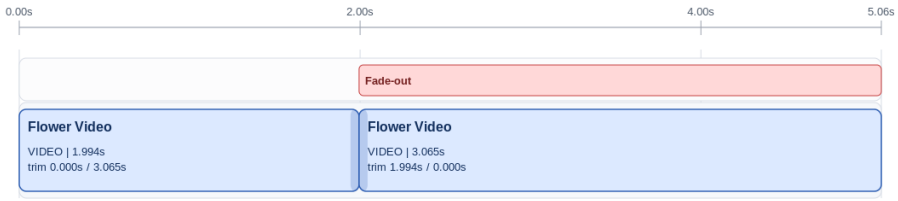
No.	Task
3	# A Single Title Card ## GOAL Build a timeline that contains one styled text block and nothing else. ## STEPS 1. Add a text block with the text "Welcome", placed at the very beginning of the timeline. 2. Set the font color to black. 3. Set the background color to white. 4. Set the text block duration to 3 seconds. 5. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
4	# Keep Only the Last 4 Seconds ## GOAL Add a video to the timeline and trim it from its start so only the ending remains. ## STEPS 1. Open the "Sample Media" dropdown, select "Tuning a Radio" and add it to the timeline. 2. Trim the start of the clip so that only the last 4 seconds remain. 3. Move the trimmed clip to the very beginning of the timeline. 4. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
5	# Two Clips Back to Back ## GOAL Place two full-length videos on the timeline one after the other in a given order. ## STEPS 1. Open the "Sample Media" dropdown and add "Flower Video" and "Tuning a Radio". 2. Drag "Flower Video" to the timeline, starting at the beginning. 3. Drag "Tuning a Radio" to the timeline, placing it immediately after "Flower Video" with no gap. 4. Do not trim either clip. 5. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

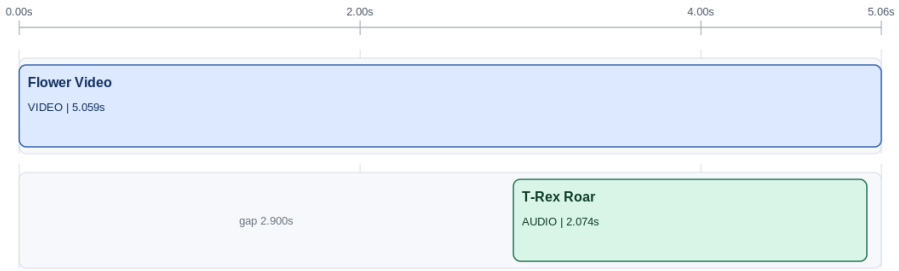
No.	Task
	
6	# Play a Clip Twice ## GOAL Place a clip on the timeline and repeat it once directly after itself. ## STEPS 1. Open the "Sample Media" dropdown, select "Tuning a Radio" and add it to the timeline, starting at the beginning. 2. Duplicate the clip once and place the copy immediately after the original with no gap. 3. Do not trim either copy. 4. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
7	# Audio at the Start of a Video ## GOAL Add a video and an audio clip so the audio plays over the beginning of the video. ## STEPS 1. Open the "Sample Media" dropdown and add "Flower Video" and the audio clip. 2. Drag "Flower Video" to the timeline, starting at the beginning, and do not trim it. 3. Drag the audio clip to a separate track so that it starts at the very beginning of the timeline, in parallel with the video. 4. Do not trim the audio clip. 5. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

No.	Task

8	`# Title Card With Background Audio`
	
	`## GOAL`
	`Build a timeline with just a text block and an audio clip playing together, with no video.`
	
	`## STEPS`
	`1. Add a text block with the text "Now Playing", placed at the very beginning of the timeline.`
	`2. Set the font color to white.`
	`3. Set the background color to black.`
	`4. Set the text block duration to 5 seconds.`
	`5. Open the "Sample Media" dropdown and add the audio clip.`
	`6. Drag the audio clip to a separate track so that it starts at the very beginning of the timeline, in parallel with the text block.`
	`7. Do not trim the audio clip.`
	`8. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````
	Ground truth
	

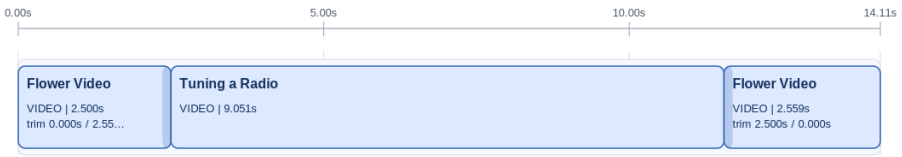
9	`# Short Opener Before the Main Clip`
	
	`## GOAL`
	`Use a short trimmed clip as an opener in front of a second, untrimmed clip.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "White Candle" and "Flower Video".`
	`2. Drag "White Candle" to the timeline, starting at the beginning, and trim it so that only the first 2 seconds remain.`
	`3. Drag "Flower Video" to the timeline, placing it immediately after "White Candle" with no gap, and do not trim it.`
	`4. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````
	Ground truth
	

No.	Task
10	# Apply Fade-Out Transition ## GOAL Add a video and apply a fade-out transition to its last few seconds. ## STEPS 1. Open the "Sample Media" dropdown, select "Flower Video" and add it to the timeline. 2. Apply a fade-out transition targeting the last 3 seconds of the clip. 3. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
11	# Trim a Video to a Specific Portion ## GOAL Add a video and trim it to keep only the segment from 4s to 11s. ## STEPS 1. Open the "Sample Media" dropdown, add "White Candle" video and drag to the timeline. 2. Trim the video so that only the 4.00-11.00 seconds chunk remain. 3. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
12	# Add Background Audio to End of Video ## GOAL Add a video and an audio clip, positioning the audio as background for the end of the video. ## STEPS 1. Open the "Sample Media" dropdown, add "Flower Video" and audio clip to the timeline. 2. Position the audio clip so that it plays as background audio during the end portion of the video clip. 3. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

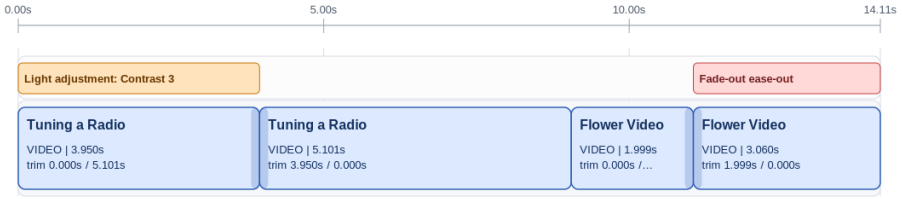
No.	Task
	
13	# Combine Three Videos in Sequence ## GOAL Place three videos on the timeline in a specific order and export. ## STEPS 1. Open the "Sample Media" dropdown and add "Tuning a Radio", "White Candle" and "Flower Video". 2. Drag all the videos to timeline and place them in same order. 3. Trim ends of all videos so length of each clip matches length of "Flower Video" clip. 4. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
14	# The Quote ## GOAL Lift a middle segment out of a video, play it on its own first, then play the full original video. ## STEPS 1. Open the "Sample Media" dropdown and add "Tuning a Radio". 2. On the timeline, first play only seconds 3 through 6 of the clip, then immediately after play the full original clip from start to end. 3. Add a 1-second text block reading "Earlier..." with white text on black background, placed immediately before the full playthrough begins. 4. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

No.	Task

15	`# Split Video and Insert Another in the Middle`
	
	`## GOAL`
	`Split a video at its midpoint and insert a second video between the two halves.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and select "Flower Video".`
	`2. Drag "Flower Video" to the timeline.`
	`3. Split the clip at its midpoint.`
	`4. Open the "Sample Media" dropdown and select "Tuning a Radio".`
	`5. Drag "Tuning a Radio" to the timeline.`
	`6. Insert "Tuning a Radio" between the two halves of "Flower Video".`
	`7. Export the result.`
	
	`# RESULT FORMAT`
	
	``json
	{"answer": "done"}
	``

	Ground truth
	

16	`# Reverse Order, Light Correction, and Fade-Out`
	
	`## GOAL`
	`Arrange two videos in reverse order, apply light correction to a portion of the first, and apply a fade-out to the second.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video".`
	`2. Open the "Sample Media" dropdown and add "Tuning a Radio".`
	`3. Drag "Tuning a Radio" to the timeline first.`
	`4. Drag "Flower Video" to the timeline, placing it right after "Tuning a Radio".`
	`5. Select "Tuning a Radio" and apply light correction to the first 4 seconds of the clip: max out contrast only.`
	`6. Select "Flower Video" and apply a fade-out effect with ease-out easing to the last 3 seconds of the clip.`
	`7. Export the result.`
	
	`# RESULT FORMAT`
	
	``json
	{"answer": "done"}
	``

	Ground truth
	

No.	Task
17	# Symmetric Audio Bridge ## GOAL Place an audio clip so its midpoint aligns with the transition between two videos, then trim both videos so the audio has equal silent runway on both sides. ## STEPS 1. Open the "Sample Media" dropdown and add "Flower Video", "Tuning a Radio" and audio clip. 2. Drag "Flower Video" and "Tuning a Radio" to the timeline back-to-back in that order. 3. Place the audio clip on a separate track so that the middle of the audio clip aligns exactly with the cut between the two videos. 4. Trim "Flower Video" from its start and "Tuning a Radio" from its end so that the audio clip begins exactly 1 second after the video timeline starts and ends exactly 1 second before the timeline ends. 5. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
18	# Split a Video, Swap Halves, and Insert a Midpoint Title ## GOAL Split a video at its midpoint, swap the two halves, and insert a styled text block between them. ## STEPS 1. Open the "Sample Media" dropdown and select "Tuning a Radio". 2. Drag "Tuning a Radio" to the timeline. 3. Split the clip at its midpoint. 4. Move the second half to the beginning of the timeline. 5. Add a text block with the text "Midpoint". 6. Set the font color to white. 7. Set the background color to black. 8. Set the text block duration to 1 second. 9. Place the text block immediately after the second half. 10. Move the first half to immediately after the text block. 11. Verify the final timeline order is: second half of "Tuning a Radio" -> "Midpoint" text block -> first half of "Tuning a Radio". 12. Export the result. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

No.	Task

19	`# Trim and Duplicate a Video`
	
	`## GOAL`
	`Trim a video, then duplicate several times.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown, add "Flower Video" and drag it to timeline.`
	`2. Trim the clip so that 4 first seconds remain, duplicate the video 4 times, but trim even numbered video to 2 last seconds.`
	`4. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````

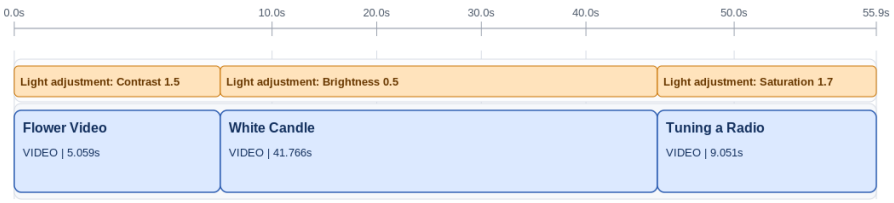
	Ground truth
	

20	`# Three Videos with Empty Gaps Between Them`
	
	`## GOAL`
	`Arrange three videos in sequence with specific empty gaps between them.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video", "White Candle" and "Tuning a Radio" videos.`
	`2. Drag the videos to timeline in the same order.`
	`3. Place the videos with empty gaps of 1 second and 2 seconds between them.`
	`4. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````

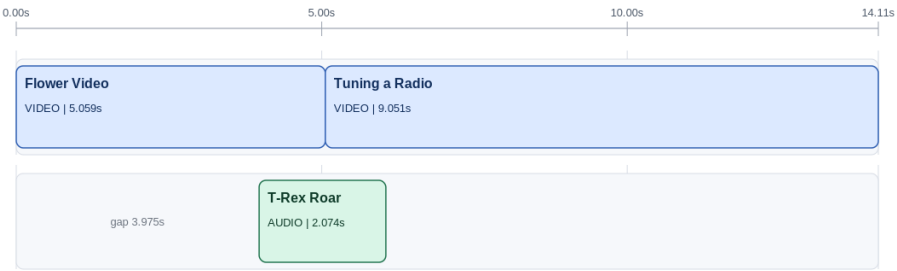
	Ground truth
	

21	`# Different Light Corrections on Three Videos`
	
	`## GOAL`
	`Arrange three videos in sequence and apply a different light correction to each.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video", "White Candle" and "Tuning a Radio".`
	`2. Drag the video to timeline in asked above order.`
	`3. Apply light correction one each video clip: max out contrast for first, brightness for second and saturation for last video respectively.`
	`4. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````

	Ground truth

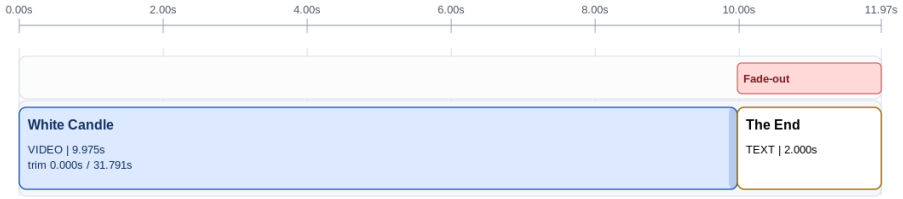
No.	Task
	

22	`# Two Videos with Audio at Transition`
	
	`## GOAL`
	`Arrange two videos in sequence and place an audio clip so it spans the transition between them.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video", "Tuning a Radio" and audio clip.`
	`2. Drag the videos to the timeline placing "Tuning a Radio" right after "Flower Video".`
	`3. Drag the audio clip to the timeline.`
	`4. Position the audio clip so that it spans the transition point between the two videos (equally overlapping the end of the first and the beginning of the second).`
	`5. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json
	{"answer": "done"}
	``` ````
	Ground truth

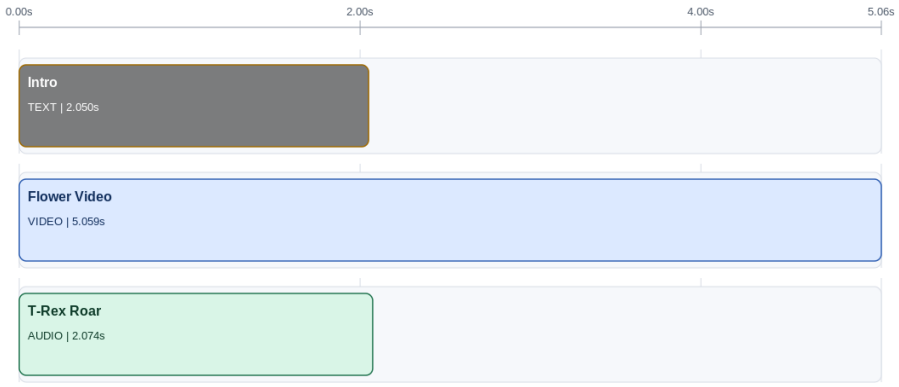
	

23	`# Three Colored Text Blocks`
	
	`## GOAL`
	`Create an intro video with different color scenes.`
	
	`## STEPS`
	`1. Add a half-second text block with white "Second!" text.`
	`2. Duplicate the text block two times, setting new blocks texts, so that the timeline reads: "Second!" -> "Third!" -> "First!", resulting in three consecutive blocks on the timeline.`
	`3. Change text blocks background colors to be "yellow", "red" and "green" accordingly to their timeline order.`
	`4. Change durations of text blocks, so that each one is 2 times longer than previous one, leaving first text block duration unchanged.`
	`5. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json
	{"answer": "done"}
	``` ````
	Ground truth

No.	Task
	

24	`# Trimmed Video with Closing Titles and Fade-Out`
	
	`## GOAL`
	`Add a video, trim it, append a closing title card, and apply a fade-out to the title.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and select "White Candle".`
	`2. Drag "White Candle" to the timeline.`
	`3. Trim the end of the clip so that only the first 10 seconds remain.`
	`4. Add a text block with the text "The End", black font color, and white background.`
	`5. Set the text block duration to 2 seconds.`
	`6. Place the text block on the timeline immediately after the trimmed video.`
	`7. Select the text block and apply a fade-out effect to it.`
	`8. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json
	{"answer": "done"}
	``` ````
	
	Ground truth
	

25	`# Overlay Text and Audio on a Video`
	
	`## GOAL`
	`Add a video to the timeline, overlay a text block on the first 2 seconds, and add an audio clip playing in parallel for the same 2 seconds.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video" and audio clip.`
	`2. Drag "Flower Video" to the timeline.`
	`3. Add a 2-second "Intro" text block with font size of 72 on a track above the video track so it overlays the first 2 seconds of "Flower Video".`
	`4. Drag the audio clip to the timeline.`
	`5. Position the audio clip so it starts at the beginning of the timeline and plays for the first 2 seconds, in parallel with the video and the text overlay.`
	`6. Export the result.`
	
	`# RESULT FORMAT`
	
	```` ```json
	{"answer": "done"}
	``` ````
	
	Ground truth

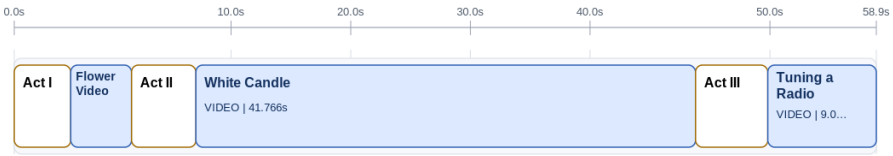
No.	Task
	

26	`# Alternating Split Segments with Per-Segment Corrections`
	
	`## GOAL`
	`Split two videos into quarters, interleave them on the timeline, and apply a different light correction to each odd-indexed segment.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Tuning a Radio" and "White Candle".`
	`2. Drag "Tuning a Radio" to the timeline and split it into 3 equal segments (T1, T2, T3).`
	`3. Drag "White Candle" to the timeline after "Flower Video" and split it into 3 equal segments (W1, W2, W3).`
	`4. Reorder the segments on the timeline so the final order is: T1 -> W1 -> T2 -> W2 -> T3 -> W3.`
	`5. Apply light correction to first two odd-positioned segment on the timeline: max out contrast for first segment, max out brightness for last segment.`
	`6. Trim each of the segments so that all segments on the timeline have the same duration, equal to the shortest segment.`
	`7. Export the result.`
	
	`# RESULT FORMAT`
	
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

No.	Task

27	`# Act Titles Before Each Video`
	
	`## GOAL`
	`Arrange three videos in sequence, each preceded by a titled text block.`
	
	`## STEPS`
	`1. Open the "Sample Media" dropdown and add "Flower Video".`
	`2. Open the "Sample Media" dropdown and add "White Candle".`
	`3. Open the "Sample Media" dropdown and add "Tuning a Radio".`
	`4. Add a text block with text "Act I", black font color, white background, and 1-second duration. Place it on the timeline.`
	`5. Drag "Flower Video" to the timeline, placing it immediately after the "Act I" text block.`
	`6. Add a text block with text "Act II", black font color, white background, and 1-second duration. Place it on the timeline immediately after "Flower Video".`
	`7. Drag "White Candle" to the timeline, placing it immediately after the "Act II" text block.`
	`8. Add a text block with text "Act III", black font color, white background, and 1-second duration. Place it on the timeline immediately after "White Candle".`
	`9. Drag "Tuning a Radio" to the timeline, placing it immediately after the "Act III" text block.`
	`10. Verify the final timeline order is: "Act I" -> "Flower Video" -> "Act II" -> "White Candle" -> "Act III" -> "Tuning a Radio".`
	`11. Export the result.`
	
	`# RESULT FORMAT`
	
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

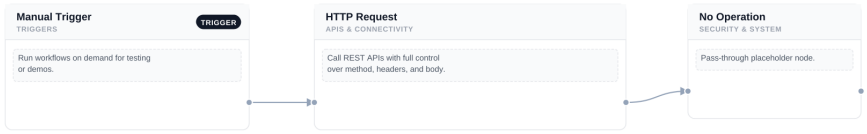
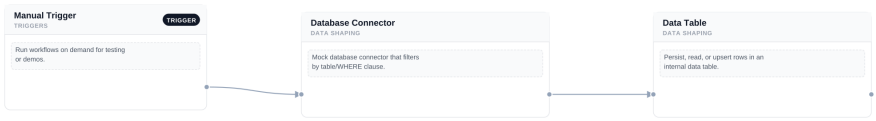
	Ground truth
	

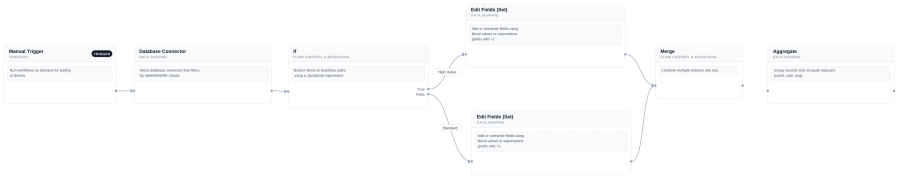
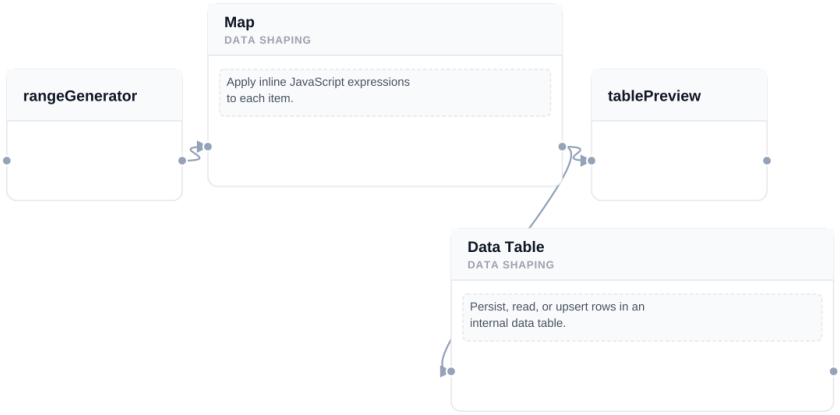
G.2 WORKFLOW BUILDER

No.	Task

1	`# Remove a Node and Reconnect in a Pipeline`
	
	`## GOAL`
	`Remove a node from a multi-step pipeline and reconnect the upstream node to the downstream node.`
	
	`## STEPS`
	`1. Open the "API Data Pipeline" workflow.`
	`2. Delete the "Sort" node.`
	`3. Reconnect the "Map" node directly to the next downstream node.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

No.	Task
2	# Create a Simple Three-Node Workflow ## GOAL Create a simple three-node workflow from scratch with a Manual Trigger, a configured HTTP Request, and a No Operation node. ## STEPS 1. Create a new workflow named "Example Todo Fetch". 2. Add these nodes and connect them in sequence: ‘Manual Trigger -> HTTP Request -> No Operation‘. 3. Configure the "HTTP Request" node: - Method: "GET" - URL: ‘https://jsonplaceholder.typicode.com/todos/1‘ # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
3	# Create an Order Snapshot Workflow ## GOAL Create a three-node workflow that reads rows from a database table and writes them into a data table. ## STEPS 1. Create a new workflow named "Order Snapshot". 2. Add these nodes and connect them in sequence: ‘Manual Trigger -> Database Connector -> Data Table‘. 3. Configure the "Database Connector" node to read the ‘orders‘ table. 4. Configure the "Data Table" node: - Table name: ‘order_snapshot‘ - Operation: ‘insert‘ # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
4	# Detach the Aggregate Step ## GOAL Remove a single connection from a pipeline, leaving every node in place. ## STEPS 1. Open the "Order Processing Pipeline" workflow. 2. Delete the connection that goes from the "Merge" node to the "Aggregate" node. 3. Do not delete any node. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

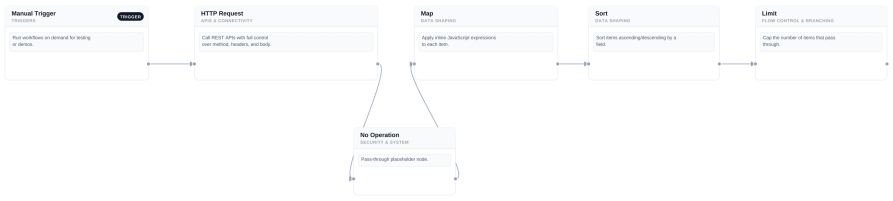
No.	Task
	
5	# Add a Table Output Alongside the Preview ## GOAL Add one node to an existing workflow and wire it as a second output of an existing node, leaving the current flow untouched. ## STEPS 1. Open the "Range -> Map -> Table (scaffolding)" workflow. 2. Add a "Data Table" node to the canvas. 3. Connect the "Map" node output to the new "Data Table" node input. 4. Keep the existing connection from "Map" to "Table Preview" in place. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
6	# Change the Sort Field and Direction ## GOAL Reconfigure an existing Sort node in a pipeline. ## STEPS 1. Open the "API Data Pipeline" workflow. 2. Open the "Sort" node settings. 3. Set the sort field to ‘city‘ and the direction to ‘desc‘. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

No.	Task

7	`# Create a Webhook Responder`
	
	`## GOAL`
	`Build a two-node workflow that receives a webhook call and answers it.`
	
	`## STEPS`
	`1. Create a new workflow named "Webhook Echo".`
	`2. Add these nodes and connect them: ‘Webhook Trigger -> Respond to Webhook‘.`
	`3. Configure the "Webhook Trigger" node:`
	`- Method: "GET"`
	`- Path: ‘/hook/echo‘`
	`4. Configure the "Respond to Webhook" node with status code 404.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

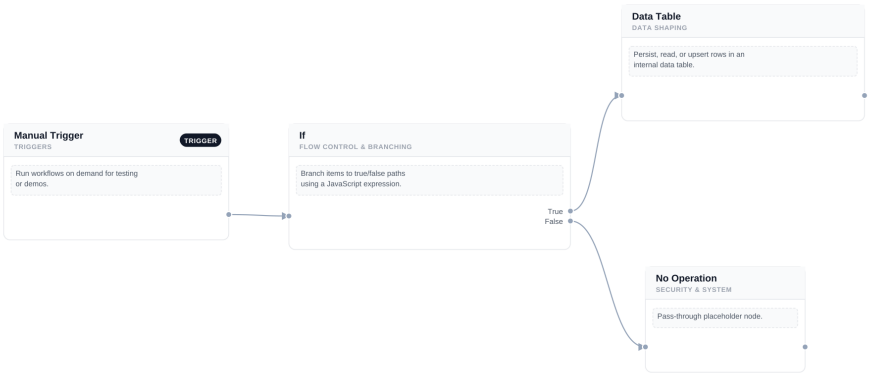
	Ground truth
	

8	`# Insert a No Operation Node into the API Pipeline`
	
	`## GOAL`
	`Insert a node between two existing nodes of a pipeline and rewire the flow through it.`
	
	`## STEPS`
	`1. Open the "API Data Pipeline" workflow.`
	`2. Add a "No Operation" node between the "HTTP Request" and "Map" nodes.`
	`3. Reconnect the workflow so the path through this section is: ‘HTTP Request -> No Operation -> Map‘.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

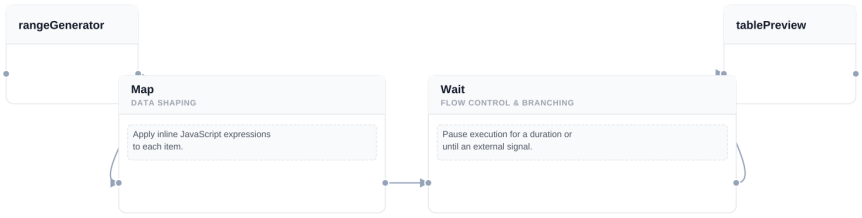
	Ground truth
	

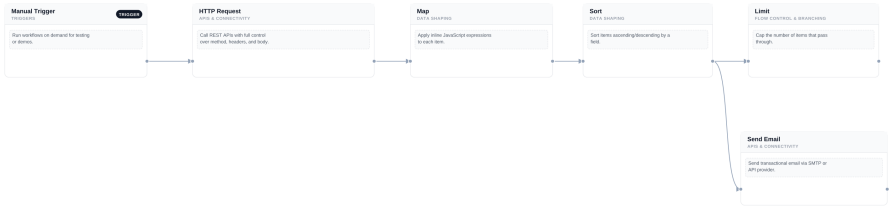
No.	Task

9	`# Create a Two-Branch Condition`
	
	`## GOAL`
	`Build a small workflow where an If node routes items into two different nodes.`
	
	`## STEPS`
	`1. Create a new workflow named "Value Router".`
	`2. Add exactly these nodes: "Manual Trigger", "If", "Data Table", "No Operation".`
	`3. Set the "If" condition to: ‘item.value > 5‘.`
	`4. Connect the workflow as follows:`
	`- ‘Manual Trigger -> If‘`
	`- The ‘true‘ output of "If" goes to the "Data Table" node.`
	`- The ‘false‘ output of "If" goes to the "No Operation" node.`
	`5. Do not add any other nodes.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

10	`# Insert a Wait Node into a Workflow`
	
	`## GOAL`
	`Insert a Wait node with a 1-second delay between two existing nodes in a workflow and reconnect the flow.`
	
	`## STEPS`
	`1. Open the "Range -> Map -> Table (scaffolding)" workflow.`
	`2. Insert a "Wait" node between the "Map" and "Table Preview" nodes.`
	`3. Configure the delay to "1 second".`
	`4. Reconnect the workflow correctly.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

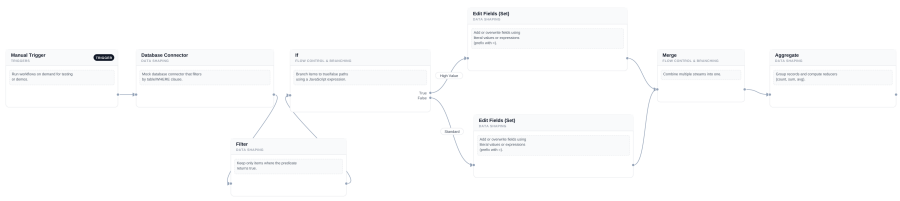
No.	Task
11	# Delete a Node and Disable Default Port in Switch ## GOAL Delete a branch target node from a Switch-based pipeline and disable the corresponding default port in the Switch settings. ## STEPS 1. Open the "Data Categorization Pipeline" workflow. 2. Delete one of the "Edit Fields (Set)" node responsible for "Default" branch of a "Switch" node. 3. Disable the "Default" port in the "Switch" node settings. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
12	# Add a Node Using Search ## GOAL Use the node search to find and add a Send Email node, configure its subject, and connect it to the end of a pipeline. ## STEPS 1. Open the "API Data Pipeline" workflow. 2. Use the node search box to find "Send Email". 3. Add the "Send Email" node to the canvas after the "Sort" node. 4. Set "Send Email" subject to "Users list". 5. Connect the "Sort" node output to the "Send Email" node input. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
13	# Insert a Formula Node into a Workflow ## GOAL Insert a Formula node into a workflow, configure it to compute a derived field, and reconnect the data flow. ## STEPS 1. Open the "Range -> Map -> Table (scaffolding)" workflow. 2. Insert a "Formula" node between the "Map" and "Table" nodes. 3. Configure the "Formula" node to name a new field ‘double‘ with value ‘item.value * 2‘. 4. Reconnect the workflow so the final path is: ‘Range -> Map -> Formula -> Table‘. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

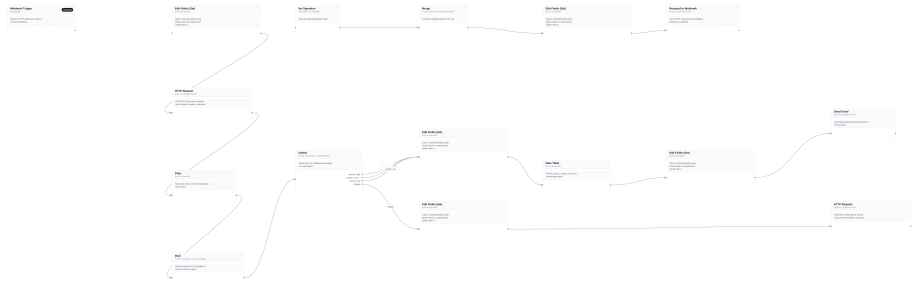
No.	Task
	

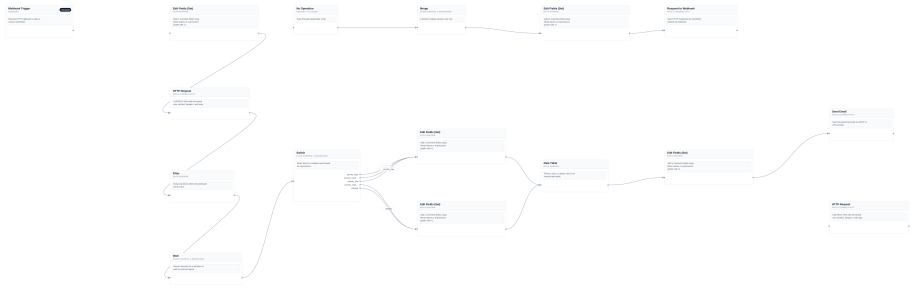
14	`# Insert and Configure a Filter Node`
	
	`## GOAL`
	`Add and configure a Filter node in an existing workflow, wiring it into the correct position in the data flow.`
	
	`## STEPS`
	`1. Open the "Range -> Map -> Table (scaffolding)" workflow.`
	`2. Delete the rightmost "Table" node.`
	`3. Add a "Filter" node and connect "Map" node output to the "Filter" node input .`
	`4. Configure the filter to pass only values where ‘item.value > 10‘.`
	`5. The final path must be: ‘Range -> Map -> Filter‘.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

15	`# Insert a Filter Node into an Existing Pipeline`
	
	`## GOAL`
	`Insert a Filter node between two existing nodes in a pipeline and configure it with a specific condition.`
	
	`## STEPS`
	`1. Open the "Order Processing Pipeline" workflow.`
	`2. Add a "Filter" node between the "Database Connector" and the "If" node.`
	`3. Configure the filter to pass only orders where ‘item.status === ’completed’‘.`
	`4. Reconnect the workflow so the path through this section is: ‘Database Connector -> Filter -> If‘.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

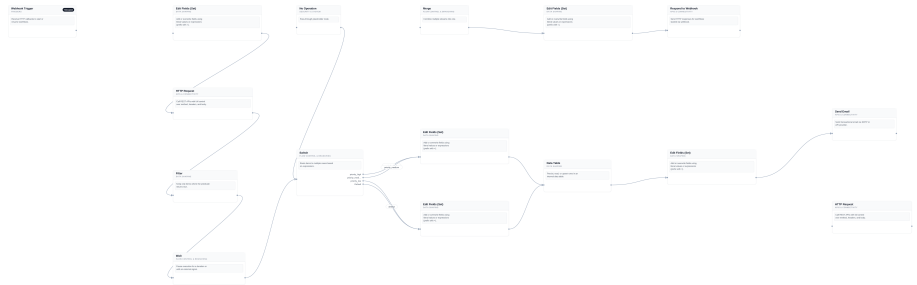
	Ground truth
	

No.	Task
16	# Rewire a Branch Input to a Different Node ## GOAL Rewire a branch input on a downstream node so it feeds into a different merge point in the pipeline. ## STEPS 1. Open the "Enterprise Data Pipeline" workflow. 2. Find the "Data Table" node that receives inputs from the priority branches. 3. Disconnect the input coming from the lower "Edit Fields (Set)" node. 4. Reconnect that lower "Edit Fields (Set)" node into the "HTTP Request" node located on the right, below "Send Email" node. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
17	# Add an Side Branch Without Breaking Existing Output ## GOAL Add a side branch that records processed records while keeping the original main pipeline output unchanged. ## STEPS 1. Open the "Order Processing Pipeline" workflow. 2. Find the node immediately before the final "Aggregate" output node. 3. Add a "No Operation" node. 4. Add a "Data Table" node after the "No Operation" node. 5. Connect the node immediately before the final output node to the new "No Operation" node, leaving all other existing connections as is. 6. Connect "No Operation" to the new "Data Table" node. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

No.	Task
18	# Split a Pipeline into Two Conditional Branches ## GOAL Modify an existing linear workflow so that active and inactive users are processed through separate branches. ## STEPS 1. Open the "API Data Pipeline" workflow. 2. Insert an "If" node between the "Map" node and the next downstream node. 3. Configure the "If" condition to check: ‘item.active === true‘ 4. Add a new "Send Email" node for the true branch. 5. Set the new "Send Email" subject to: ‘Active user report‘ 6. Connect the workflow so that: - ‘HTTP Request -> Map -> If‘ - The ‘true‘ output of "If" goes to the new "Send Email" node. - The ‘false‘ output of "If" goes to the existing downstream path that originally followed "Map". # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
19	# Add a New Output Branch to a Switch Node ## GOAL Add a new output case to a Switch node in a complex pipeline and wire it to an existing downstream node. ## STEPS 1. Open the "Enterprise Data Pipeline" workflow. 2. In the main "Switch" node, add a new case labeled ‘priority_urgent‘, do not include "expr" field. 3. Connect the new ‘priority_urgent‘ output to the same "Edit Fields (Set)" node currently connected to the "default" output. 4. Do not change any other routes or node settings. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

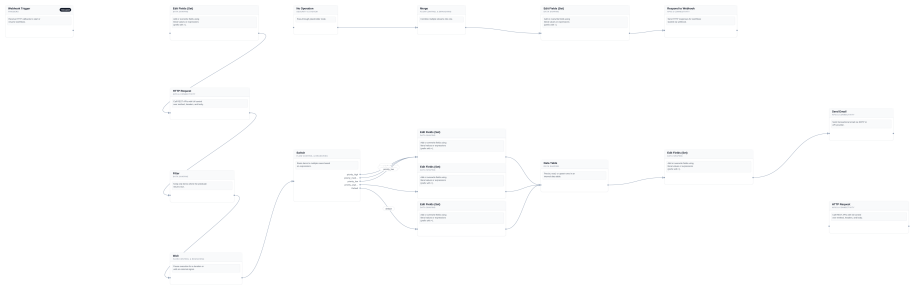
No.	Task

20	`# Rewire Edges in a Pipeline`
	
	`## GOAL`
	`Rewire existing connections in a complex pipeline without adding, deleting, or reconfiguring any nodes.`
	
	`## STEPS`
	`1. Open the "Enterprise Data Pipeline" workflow.`
	`2. Disconnect the edge from the top-row "No Operation" node to the top-row "Merge" node.`
	`3. Connect that same "No Operation" node to the "Switch" node.`
	`4. Rewire the ‘priority_low‘ output of the "Switch" to the lower "Edit Fields (Set)" node.`
	`5. Do not create, delete, move, or reconfigure any nodes.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

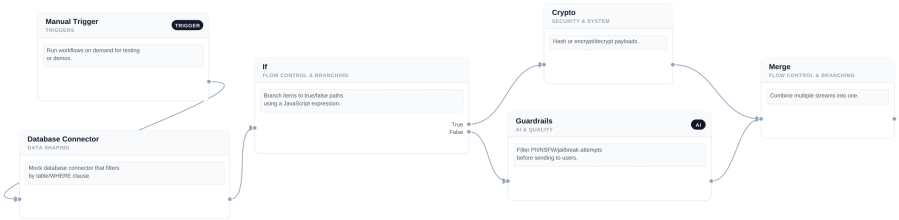
21	`# Insert a Filter and Modify Sort in a Pipeline`
	
	`## GOAL`
	`Make multiple changes to an existing pipeline: insert a Filter, reconfigure a Sort, and delete a trailing Limit node.`
	
	`## STEPS`
	`1. Open the "API Data Pipeline" workflow.`
	`2. Insert a "Filter" node between the "Map" and "Sort" nodes.`
	`3. Configure the "Filter" to pass only items where ‘item.email.includes(".org")‘.`
	`4. Change the "Sort" node to use field ‘email‘ with direction ‘desc‘.`
	`5. Delete last "Limit" node.`
	`6. The final path must be: ‘HTTP Request -> Map -> Filter -> Sort‘.`
	`7. Do not add any other nodes.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

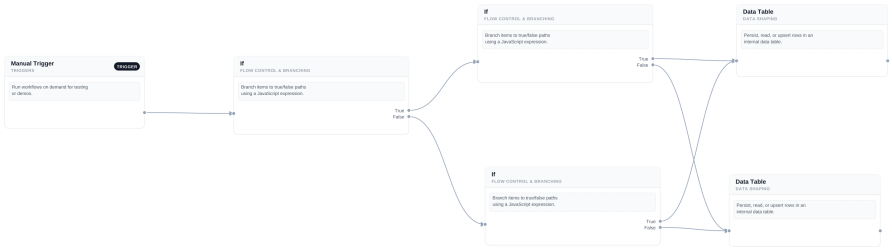
	Ground truth
	

No.	Task
22	# Add a New Priority Branch to a Switch Node ## GOAL Extend a Switch node with a new priority branch, add a dedicated processing node for it, and wire it into the main pipeline. ## STEPS 1. Open the "Enterprise Data Pipeline" workflow. 2. Find the "Switch" node that routes items by priority. 3. Add a new output branch labeled ‘priority_urgent‘, do not include "expr" field. 4. Add a new "Edit Fields (Set)" node for the urgent branch. 5. Connect the ‘priority_urgent‘ output to the new "Edit Fields (Set)" node. 6. Connect the new "Edit Fields (Set)" node back into the "Data Table" node. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
23	# Create a Simple Loop Workflow ## GOAL Create a new workflow where data passes through a loop before reaching the final output. ## STEPS 1. Create a new workflow named "Simple Loop Pipeline". 2. Add exactly these nodes: - "Manual Trigger" - "Map" - "If" - "Data Table" 3. Configure the "Map" node to add or update the field: ‘counter = (item.counter || 0) + 1‘ 4. Configure the "If" node with the condition: ‘item.counter < 3‘ 5. Connect the workflow as follows: - ‘Manual Trigger -> Map -> If‘ - The ‘true‘ output of "If" loops back to the "Map" node. - The ‘false‘ output of "If" goes to "Data Table". # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

No.	Task
	

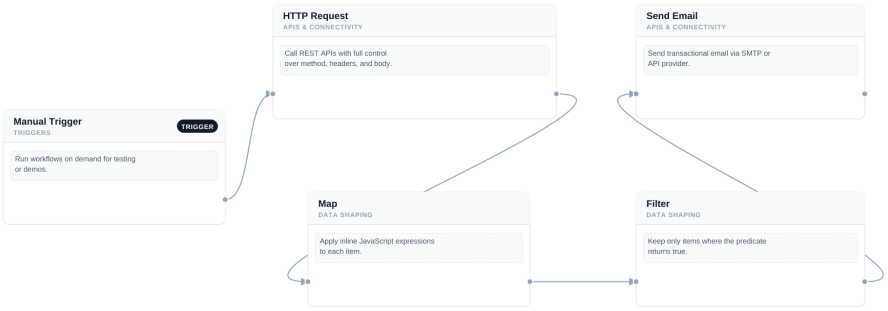
24	`# Create a Branching Workflow with Merge`
	
	`## GOAL`
	`Build a branching workflow with an If condition that routes data to two different nodes, then merges the branches back together.`
	
	`## STEPS`
	`1. Create a new workflow named "ICML Project".`
	`2. Add these nodes and connect them: ‘Manual Trigger -> Database Connector -> If‘.`
	`3. Add two more nodes: "Crypto" and "Guardrails".`
	`4. From the "If" node:`
	`- Connect the "true" output to a "Crypto" node.`
	`- Connect the "false" output to a "Guardrails" node.`
	`5. Connect both branches into a "Merge" node: ‘Crypto -> Merge‘ and ‘Guardrails -> Merge‘.`
	`6. Set the "If" condition to check: ‘item.country === "UK"‘.`
	`7. Do not add any extra nodes.`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

	Ground truth
	

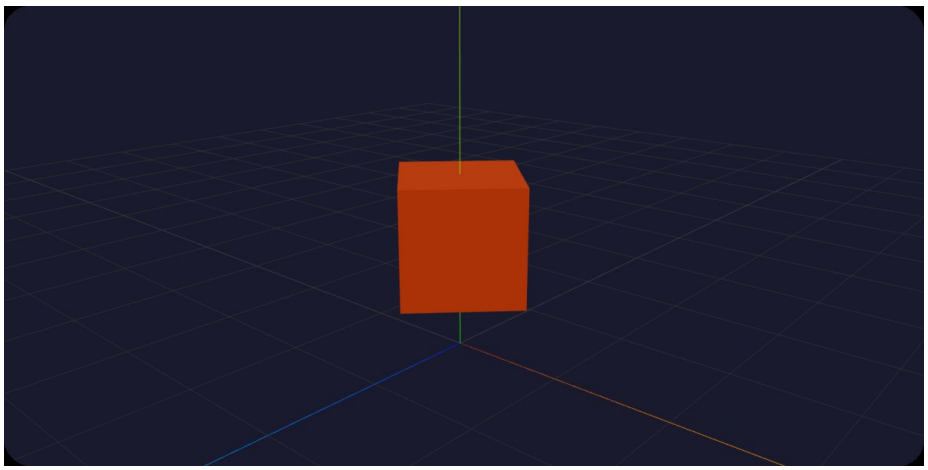
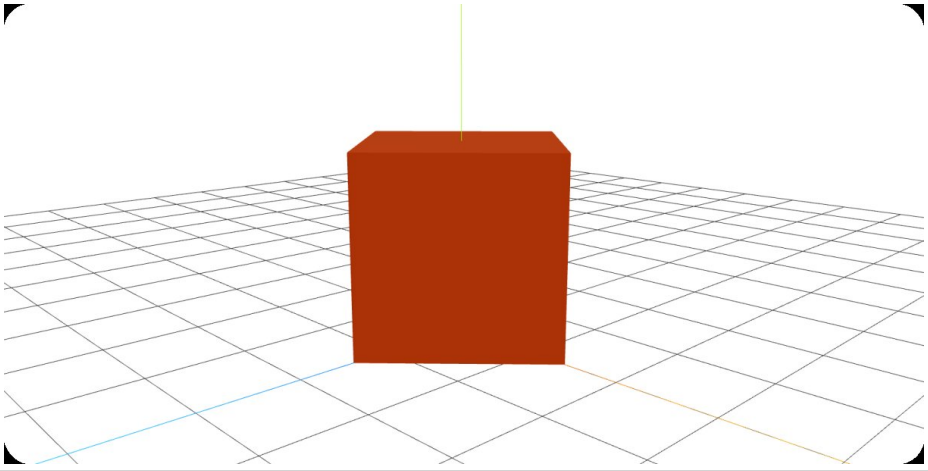
No.	Task
25	# Create a Simple Decision Tree Workflow ## GOAL Create a small tree-shaped workflow with 3 If nodes and 4 final outputs. ## STEPS 1. Create a new workflow named "Simple Decision Tree Pipeline". 2. Add: - 1 Manual Trigger - 3 If nodes - 2 Data Table nodes 3. Configure the If nodes: - Root If: ‘item.score >= 50‘ - Upper-branch If: ‘item.score >= 80‘ - Lower-branch If: ‘item.score >= 20‘ 4. Connect: - ‘Manual Trigger -> Root If‘ - Root If ‘true‘ -> Upper-branch If - Root If ‘false‘ -> Lower-branch If - Upper-branch If ‘true‘ -> first Data Table - Upper-branch If ‘false‘ -> second Data Table - Lower-branch If ‘true‘ -> first Data Table - Lower-branch If ‘false‘ -> second Data Table # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
26	# Create a Conditional Email Workflow ## GOAL Create a conditional email workflow where the If node routes to two separate Send Email nodes with different subjects. ## STEPS 1. Create a new workflow named "ICML Project". 2. Add these nodes and connect them in sequence: ‘Chat Trigger -> Database Connector -> If‘. 3. Add two "Send Email" nodes. 4. Connect first "Send Email" node only to the "false" output of the "If" node. 5. Connect second "Send Email" node only to the "true" output of the "If" node. 6. Set the "If" condition to check: ‘item.country === "FR"‘. 7. Set the first "Send Email" subject to: ‘ICML Project notification‘. 8. Set the second "Send Email" subject to: ‘Notification de projet ICML‘. 9. Do not add extra nodes. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

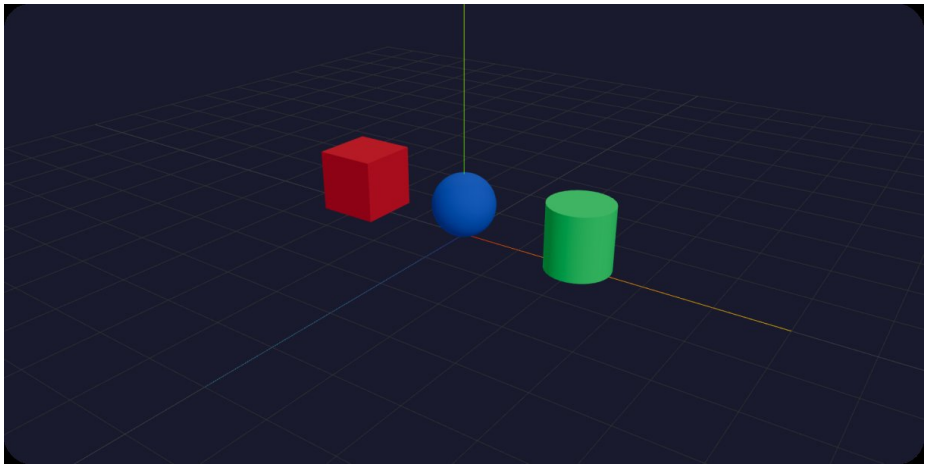
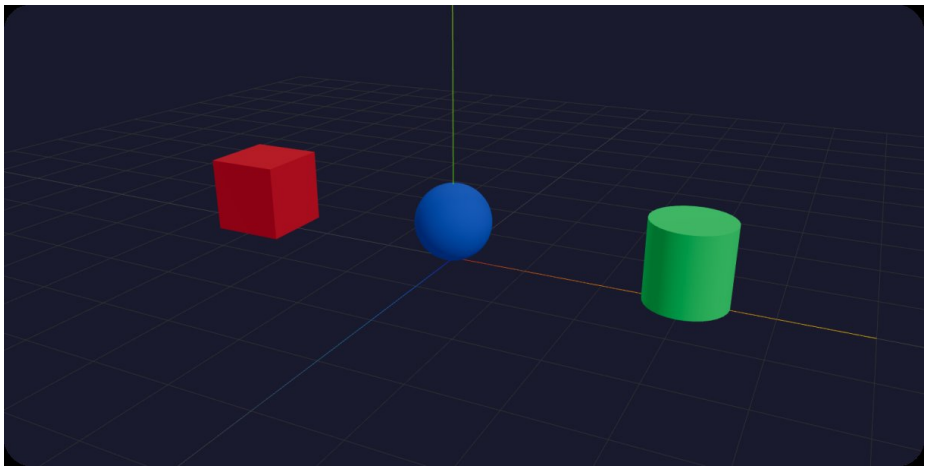
No.	Task
	

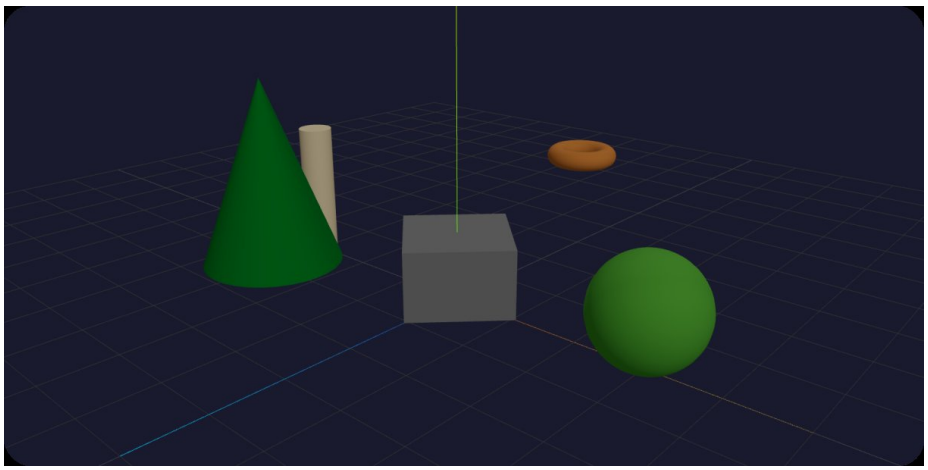
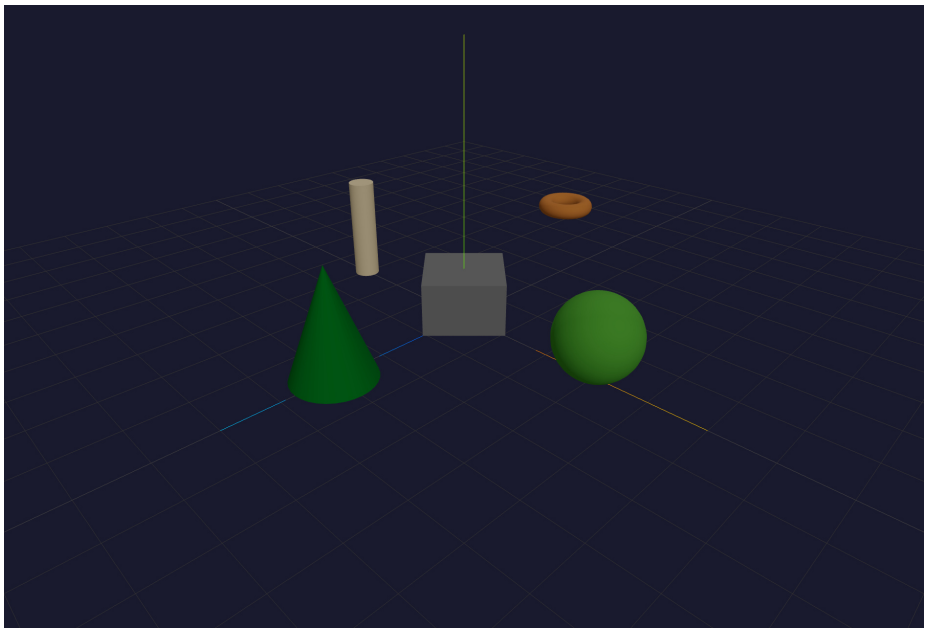
27	`# Create a Multi-Node Workflow with Configuration`
	
	`## GOAL`
	`Create a five-node workflow from scratch with HTTP, mapping, filtering, and email nodes, each configured with specific settings.`
	
	`## STEPS`
	`1. Create a new workflow named "Newsletter System".`
	`2. Add exactly these 5 nodes and wire them it the following in sequence: ‘Manual Trigger -> HTTP Request -> Map -> Filter -> Send Email‘.`
	`3. Configure the nodes:`
	`- "HTTP Request" method: "GET"`
	`- "HTTP Request" URL: ‘https://jsonplaceholder.typicode.com/users‘`
	`- "Filter" condition: pass only items where ‘item.address?.city === "Gwenborough"‘`
	`- "Send Email" From: ‘newsletter@example.com‘`
	`- "Send Email" Recipients: ‘local-subscribers@example.com‘`
	`- "Send Email" Subject: ‘Local Newsletter‘`
	
	`# RESULT FORMAT`
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

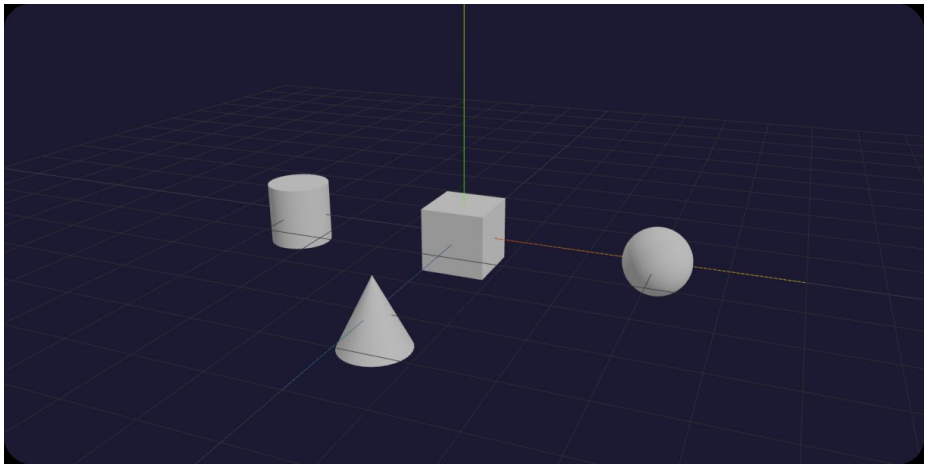
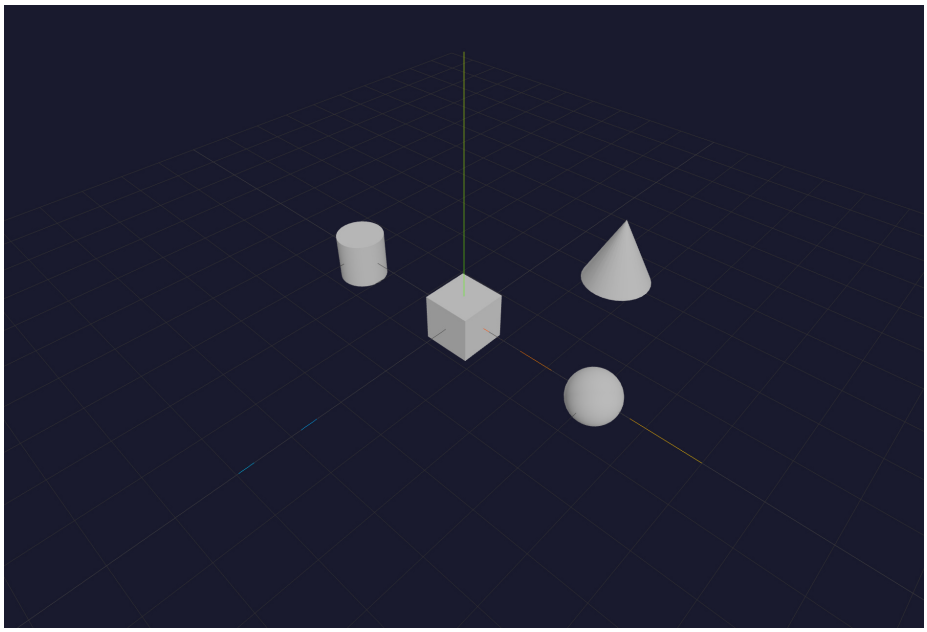
	Ground truth
	

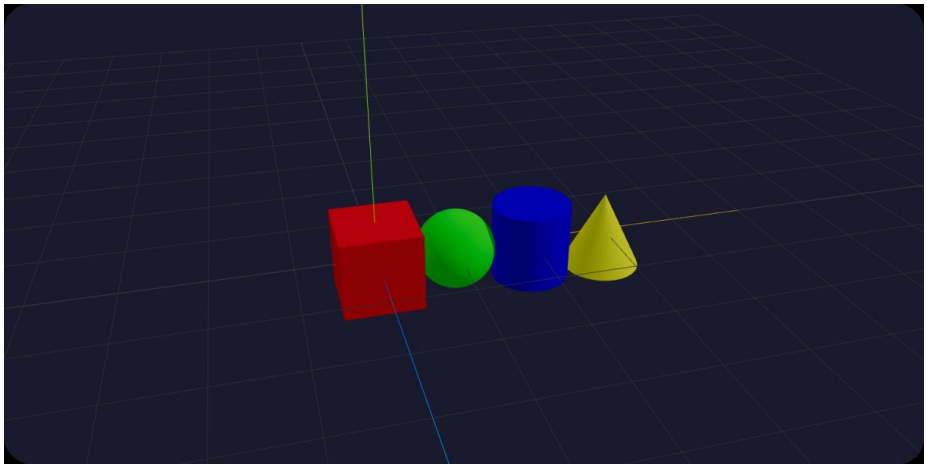
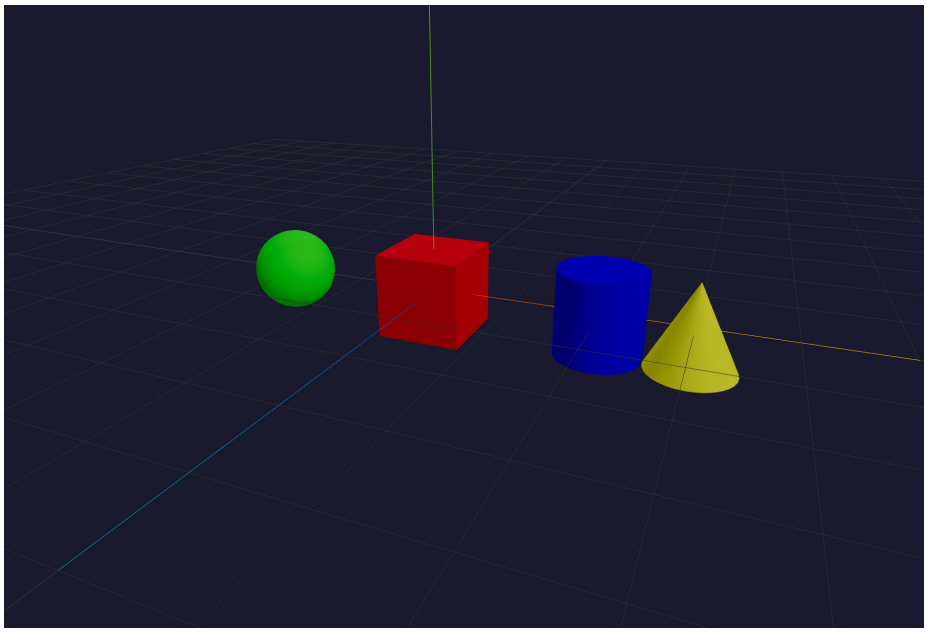
G.3 3D MODELLER

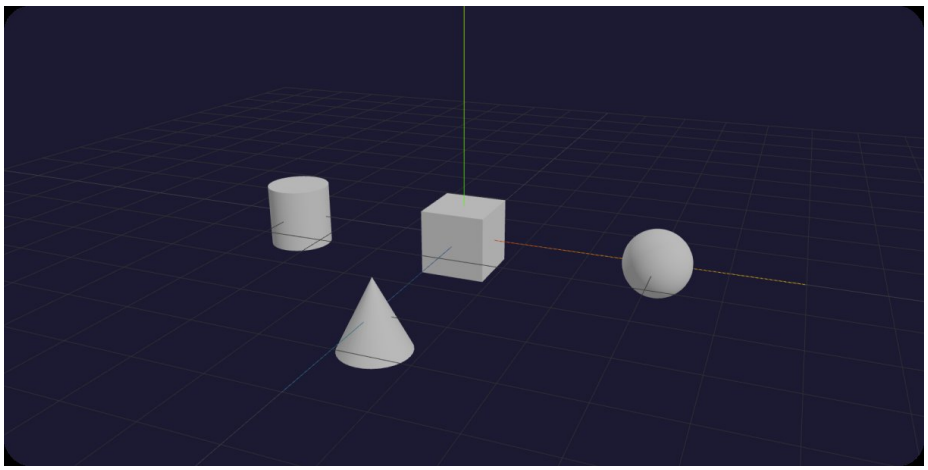
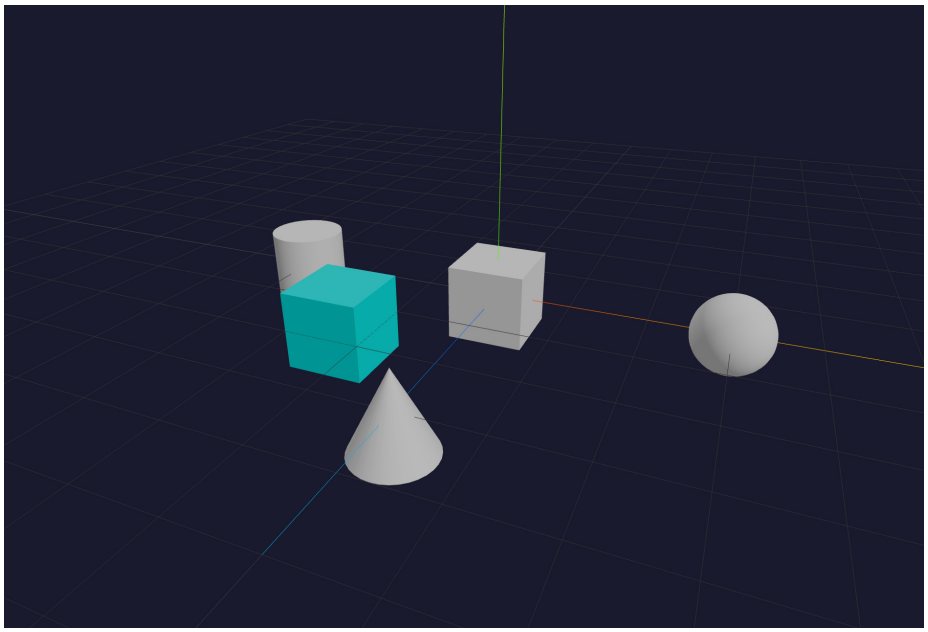
No.	Task
1	# Double the Cube's Scale ## GOAL Open a single-object scene, double the cube's scale on all axes, and change the scene background color. ## STEPS 1. Open the "Single Cube Scene". 2. Double its current scale along all three axes, and set the scene's background color to white (#ffffff). # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

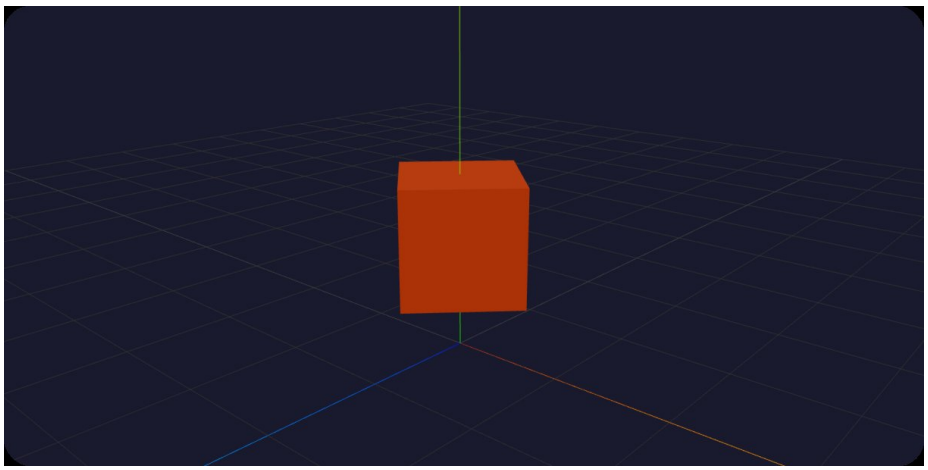
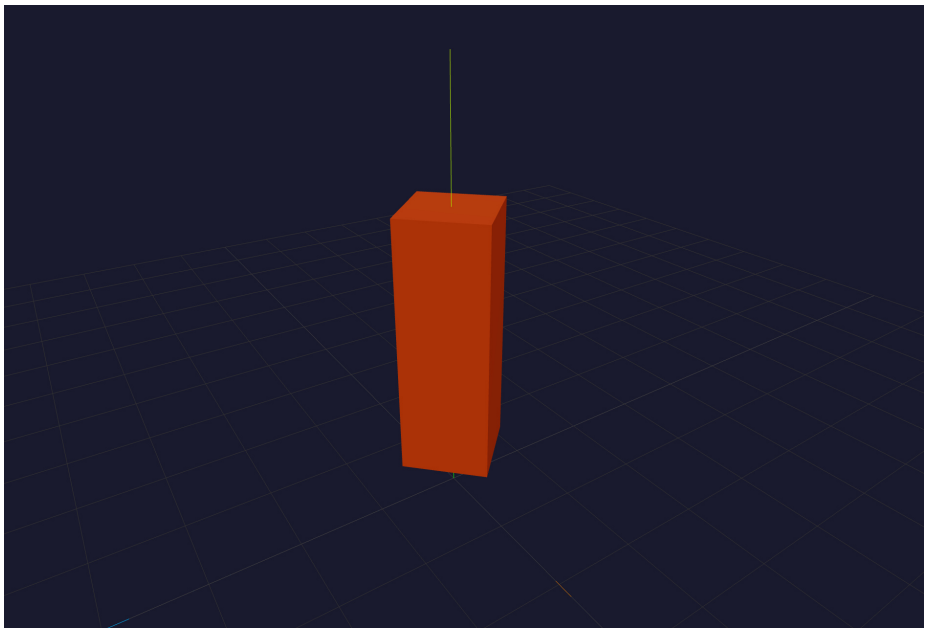
No.	Task
2	# Position Elements Relative to Reference ## GOAL Open a scene and reposition two elements at a fixed distance from a reference object along the X-axis, preserving their orientations. ## STEPS 1. Open the "Basic Shape Scene". 2. Use the blue sphere as the reference point. Position the other two elements at a distance of 3 units from the reference along the X-axis, while preserving their original orientations. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

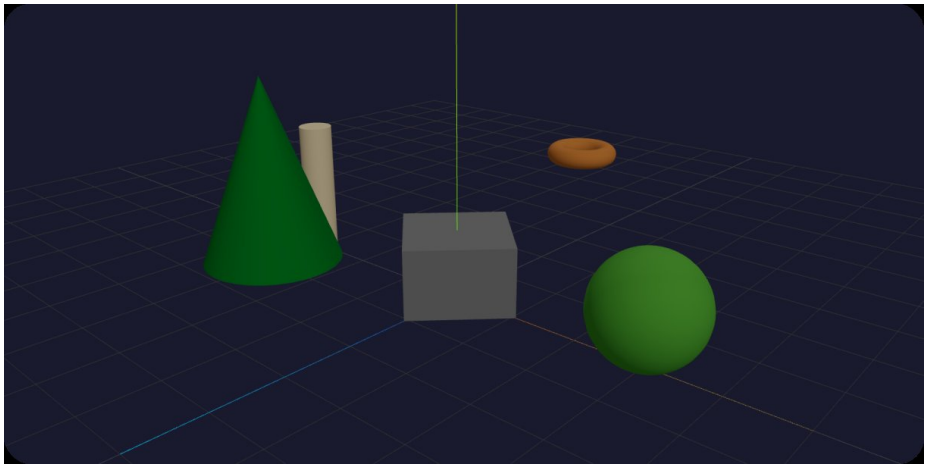
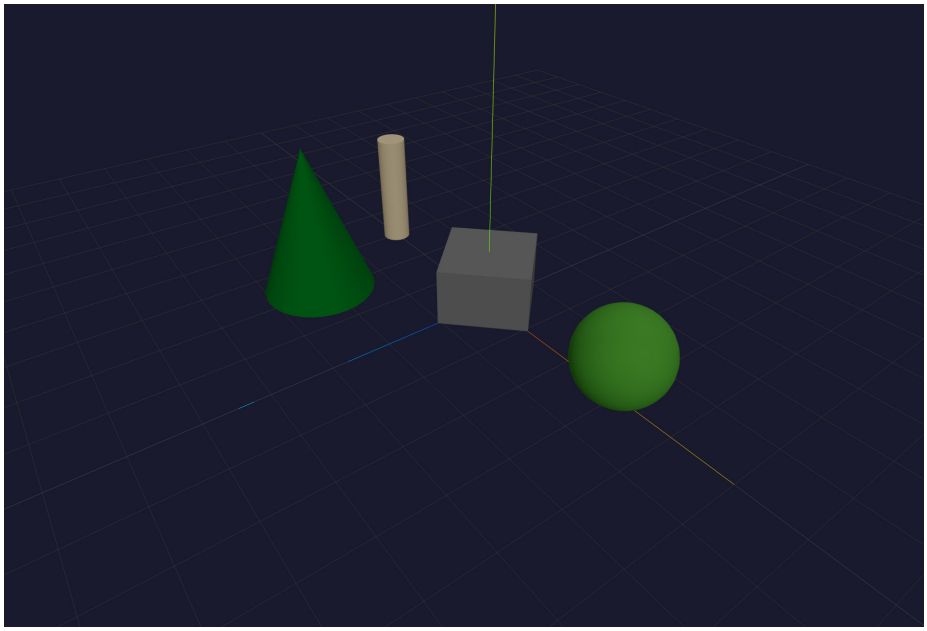
No.	Task
3	# Land the Floating Cone on the Ground ## GOAL Move one object down so that it rests exactly on the ground plane, deriving the required height from its size. ## STEPS 1. Open the "Geometric Garden". 2. Select the cone named "Tree Top". It currently floats above the ground plane (the y = 0 plane). 3. Change only its Y position so that its base touches the ground plane exactly, with no gap and no sinking below it. Keep its X and Z positions unchanged. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

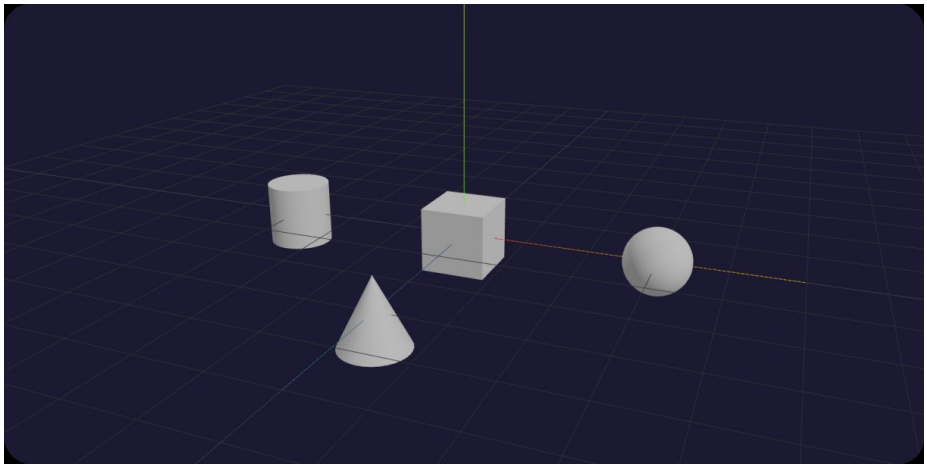
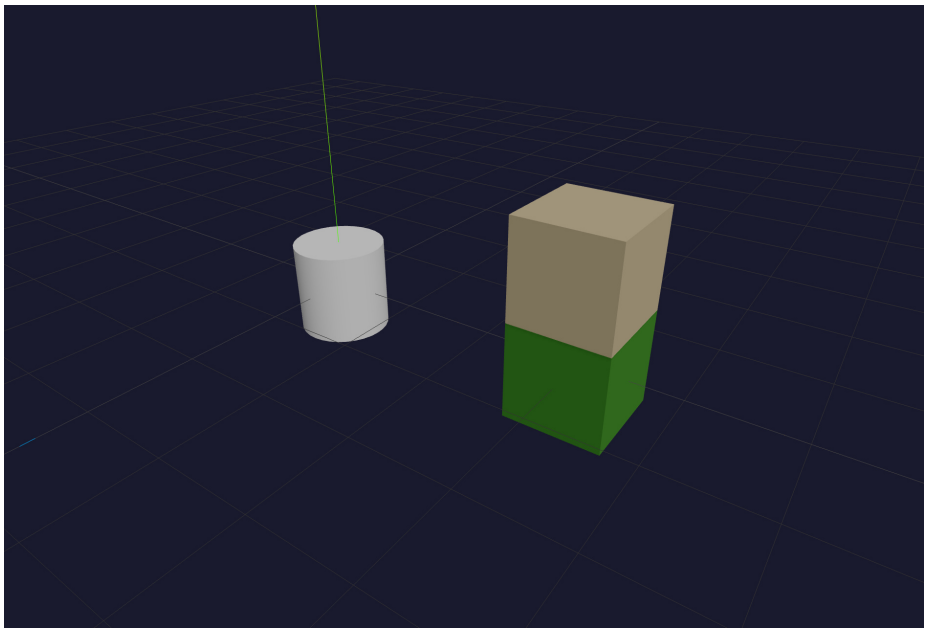
No.	Task
4	# Put the Cone Directly Above the Sphere ## GOAL Reposition one object so it is vertically aligned above another object, reading the reference position from the scene. ## STEPS 1. Open the "Four Objects Scene". 2. Read the position of "Sphere 1". 3. Move "Cone 1" so that its center is directly above the center of "Sphere 1" (same X and same Z), and 3 units higher along the Y axis. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

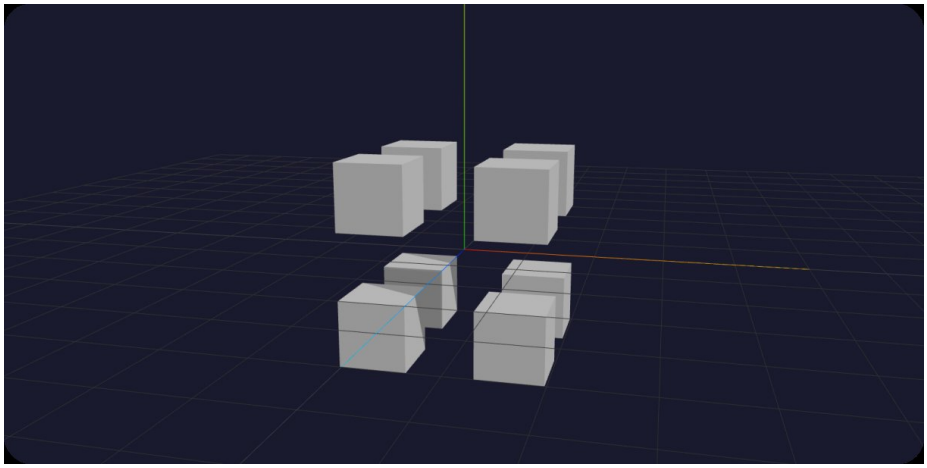
No.	Task
5	# Open a One-Unit Gap Between Two Surfaces ## GOAL Move one object along a single axis so that the gap between two surfaces, not between two centers, has a given size. ## STEPS 1. Open the "Row of Objects Scene". 2. Move "Sphere 1" along the X axis to the negative side of "Cube 1" so that the gap between the sphere's surface and the cube's nearest face is exactly 1 unit. Keep its Y and Z positions unchanged. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

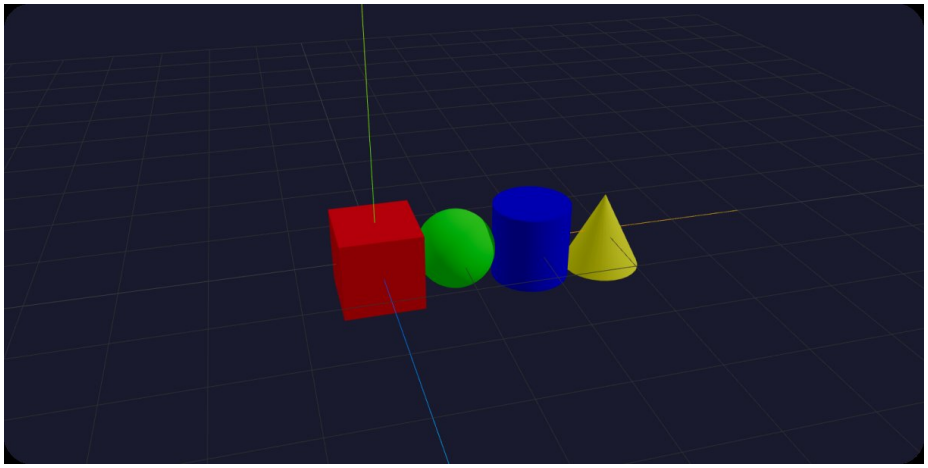
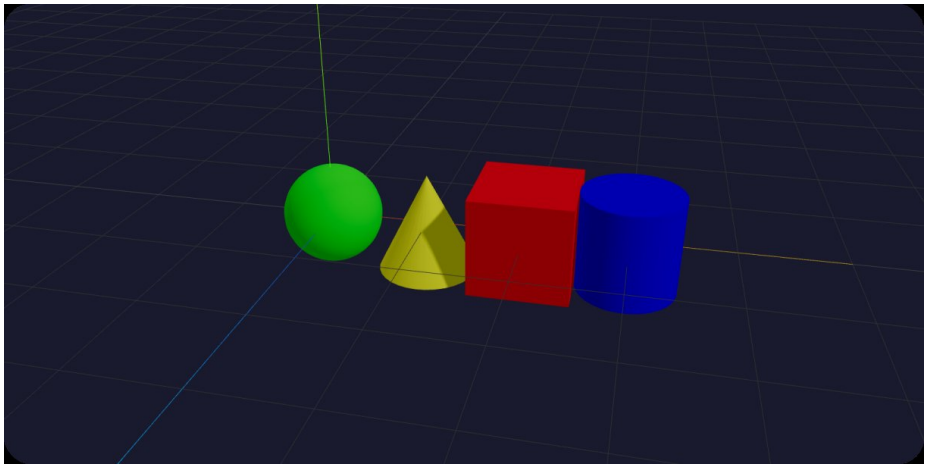
No.	Task
6	# Add a Cube at the Midpoint of Two Objects ## GOAL Add one new object exactly halfway between two existing objects. ## STEPS 1. Open the "Four Objects Scene". 2. Read the positions of "Cylinder 1" and "Cone 1". 3. Add a cube, name it "Midpoint", and set its position to the point exactly halfway between those two positions. Keep its default rotation and unit scale. 4. Set the cube's color to cyan (#00ffff). # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

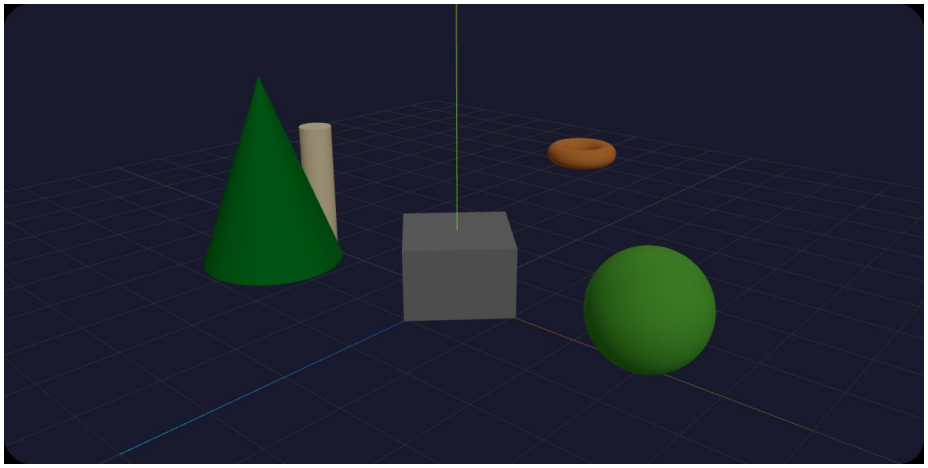
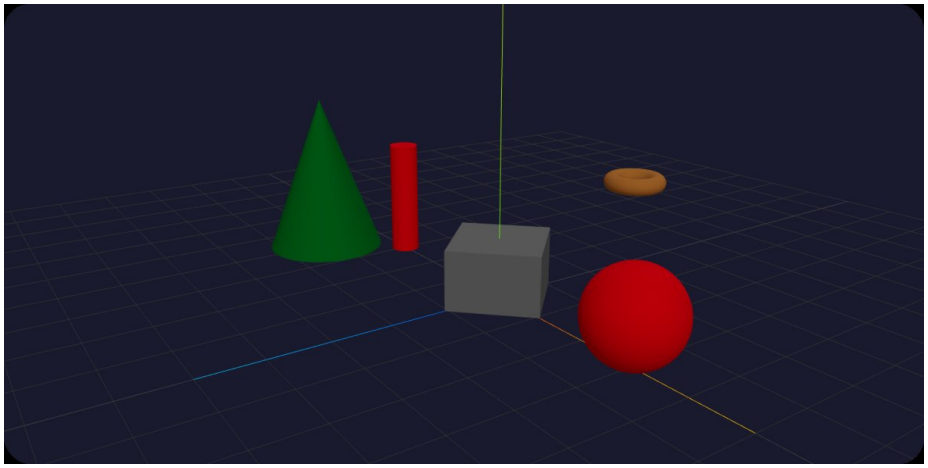
No.	Task
7	# Make the Cube Taller Without Moving Its Base ## GOAL Change one object's height while keeping its bottom face at the same level, which requires adjusting position together with scale. ## STEPS 1. Open the "Single Cube Scene". 2. Note the cube's current position and scale, and work out where its bottom face currently is. 3. Change its Y scale so the cube becomes 3 units tall, and change its Y position so its bottom face stays at exactly the same height as before. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

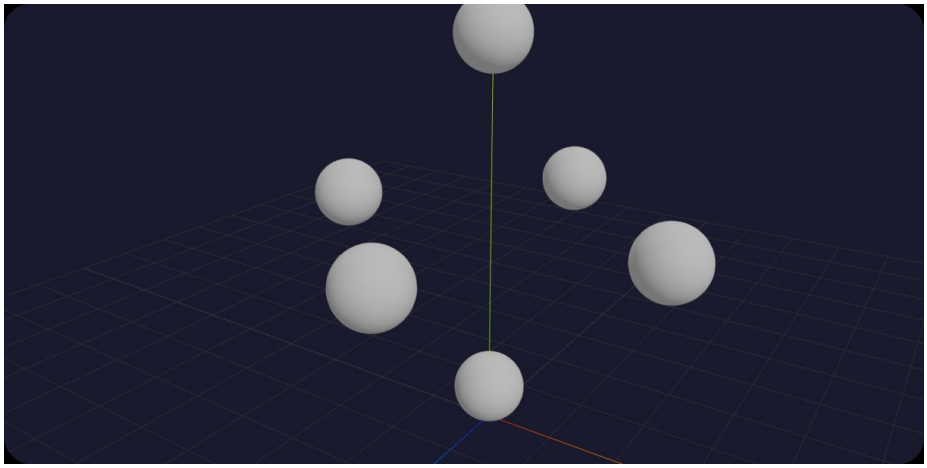
No.	Task
8	# Delete the Object on the Negative Z Axis ## GOAL Identify the one object that sits on a given axis direction and delete it. ## STEPS 1. Open the "Geometric Garden". 2. Inspect the objects' positions and identify the single object whose center lies on the negative Z axis. 3. Delete that object, and only that object. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

No.	Task
9	# Stack a Copy on Top of the Cube ## GOAL Duplicate an object and place the copy so it rests on top of the original, touching it without overlap. ## STEPS 1. Open the "Two Objects Scene". 2. Duplicate the cube. 3. Move the duplicate so that its bottom face rests exactly on the top face of the original cube: same X and Z as the original, touching it with no gap and no overlap. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth
	

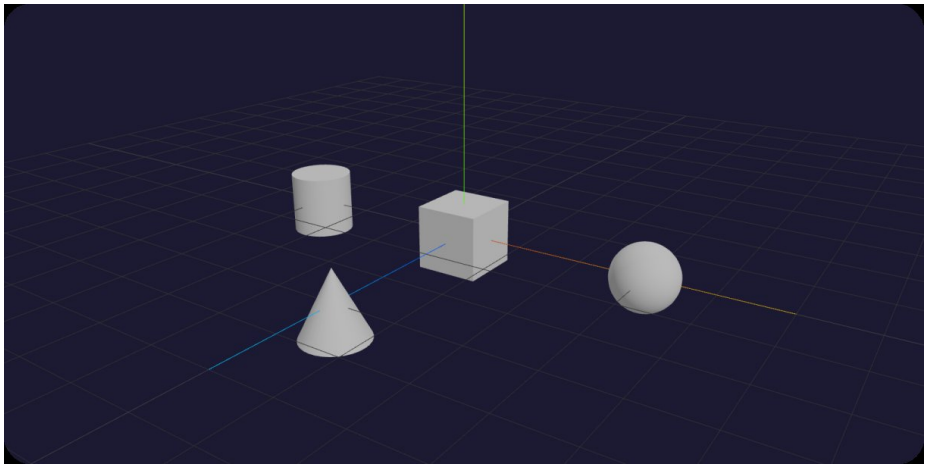
No.	Task
10	# Place Cubes in All Octants ## GOAL Create a new scene and place one cube in each of the eight 3D octants at symmetric coordinates. ## STEPS 1. Create a new scene named "Octant Cubes". 2. Generate cubes by assigning one default cube to each octant in 3D space, positioned at coordinates with absolute values of (1, 1, 1). Name all the cubes as Cube. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
11	# Reorder Elements Along X-Axis ## GOAL Rearrange four objects into a specified order along the positive X-axis with uniform spacing between centers. ## STEPS 1. Open the "Row of Objects Scene". 2. Reorder the elements as sphere, cone, cube, and cylinder along the positive X-axis. Final distance between centers of consecutive objects must be equal to 1. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state

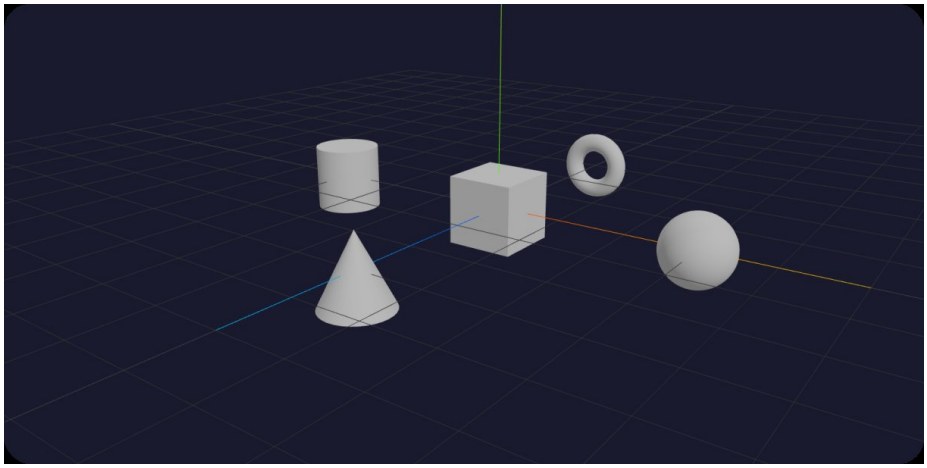
No.	Task
	
	Ground truth
	
12	# Find and Recolor Nearest Elements ## GOAL Identify the two elements closest to a reference cube and change their color to red. ## STEPS 1. Open the "Geometric Garden". 2. Use the stone block cube as the reference. Identify the two remaining elements whose centers are closest to that of the stone block cube. Finally, change their color to red (#ff0000). # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state

No.	Task
	
	Ground truth
	
13	# Spheres Touching Cube Faces From Inside ## GOAL Create unit spheres that touch the faces of an imagined cube from inside. ## STEPS 1. Open the "Empty Scene". 2. Imagine a cube of side 6 laying on the XZ plane. 3. Add one unit sphere per cube face. 4. Place each sphere inside the imagined cube so it touches, but does not intersect, its assigned face at the center of that face. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth

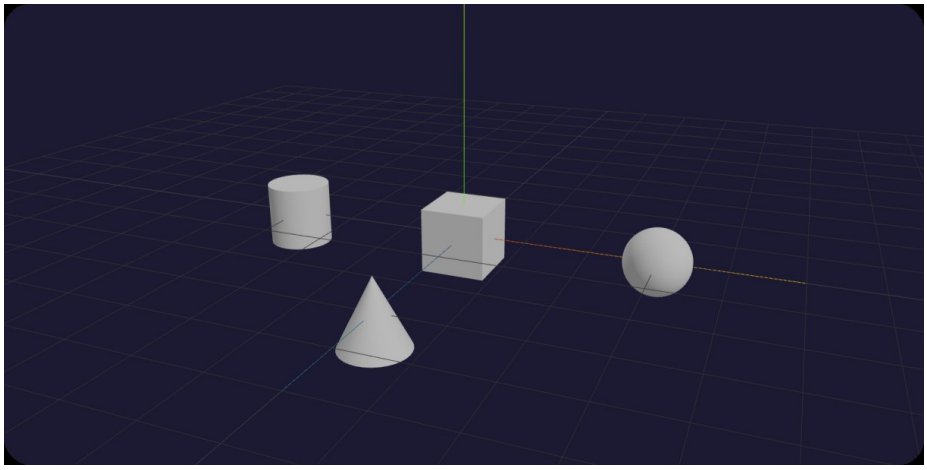
No.	Task
	

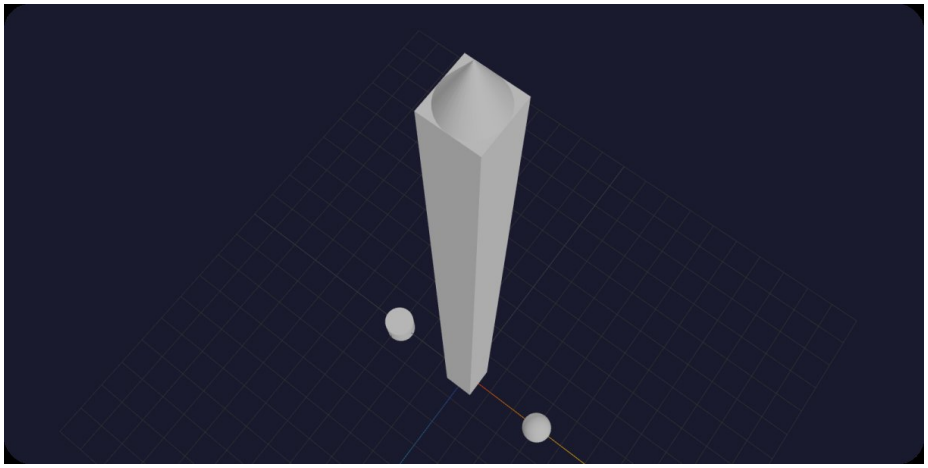
14	`# Place a Torus on the Missing Axis`
	
	`## GOAL`
	`Identify the unoccupied axis in the XZ plane and place a new torus there, matching the distance of existing elements from the center.`
	
	`## STEPS`
	`1. Open the "Four Objects Scene".`
	`2. Add a torus and place it along the only axis in the XZ plane that does not contain any existing element. Set its distance from the central point to be the same as that of the other elements.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````

	Initial state
	
	Ground truth

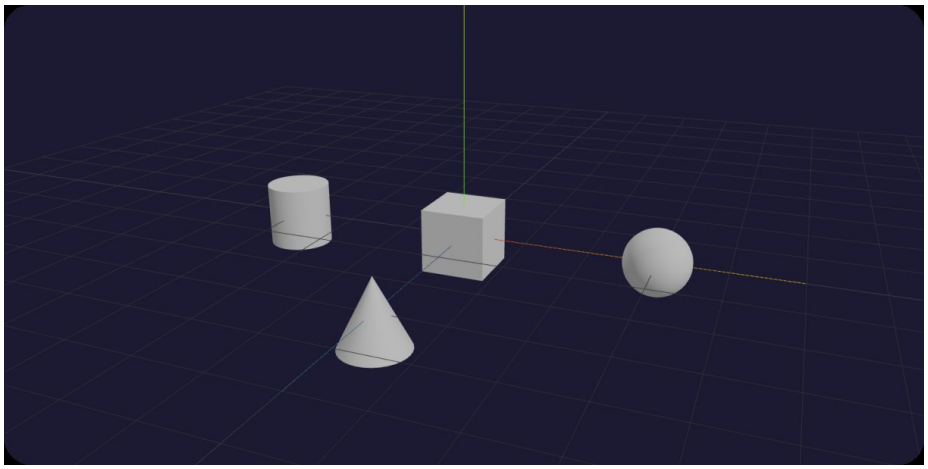
No.	Task
	

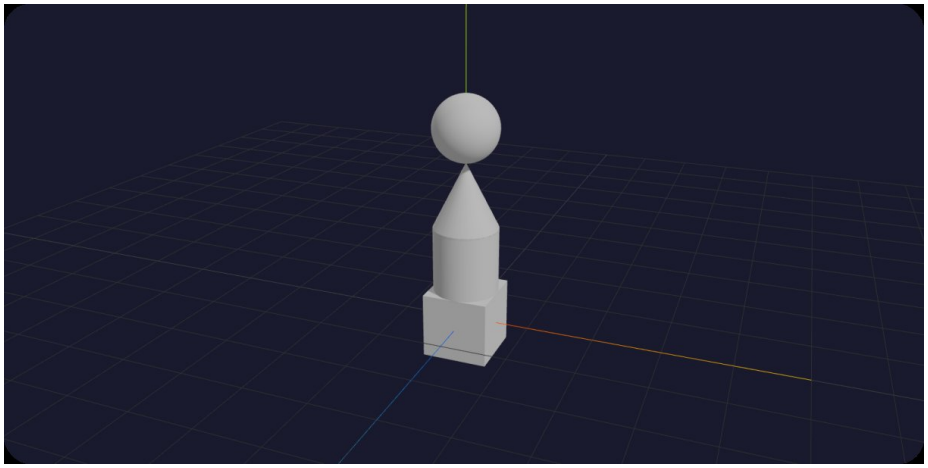
15	`# Adjust Cube (Cuboid) Scale and Stack Cone`
	
	`## GOAL`
	`Modify the cuboid's Y-axis scale and reposition it to rest on the XZ plane, then place the cone on top of the cuboid with proper surface contact.`
	
	`## STEPS`
	`1. Open the ''Four Objects Scene''.`
	`2. Set the cuboid's scale along the Y-axis to 20, and adjust its position so that its bottom surface is in contact with the XZ plane.`
	`3. Adjust the position of the cone so that its bottom surface is in contact with the top surface of the cuboid, without overlap.`
	
	`# RESULT FORMAT`
	
	``json
	{"answer": "done"}
	``

	Initial state
	
	Ground truth

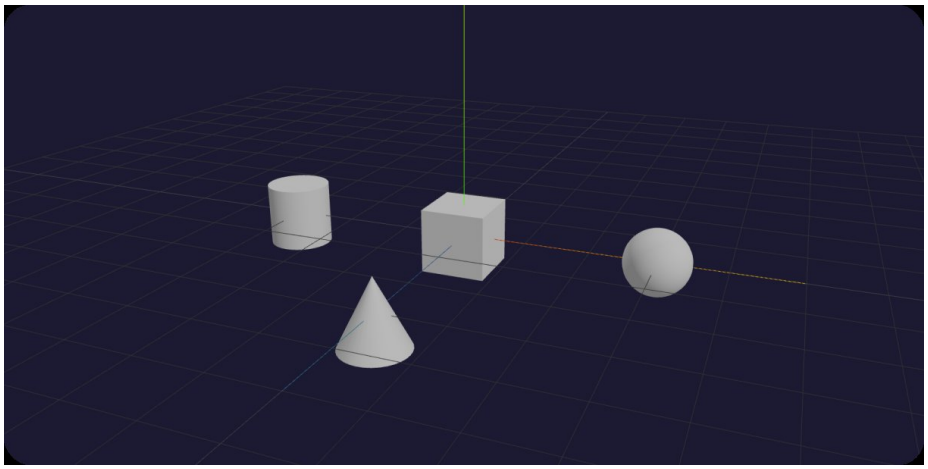
No.	Task
	

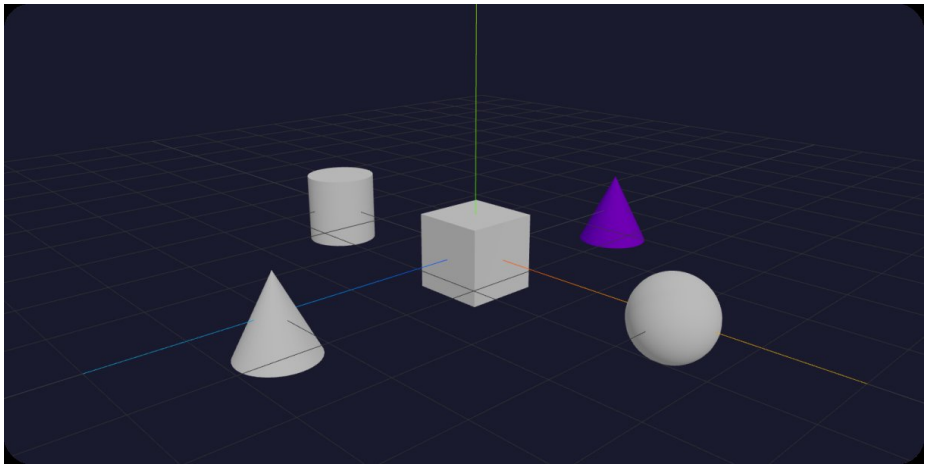
16	`# Symmetric Scene Around the Origin Object`
	
	`## GOAL`
	`Make the scene symmetric using the object at the origin as the scene center.`
	
	`## STEPS`
	`1. Open the "Two Objects Scene".`
	`2. Treat the object located at the origin as the center of the scene. Do not move or modify this center object.`
	`3. Change the existing cube's color to red (#ff0000).`
	`4. Add exactly one new object.`
	`5. Change the new object's properties so that the final scene is symmetric around the center object.`
	`6. Do not add any guide objects or extra objects.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{"answer": "done"}`
	```` ``` ````
	
	Initial state

	
	Ground truth

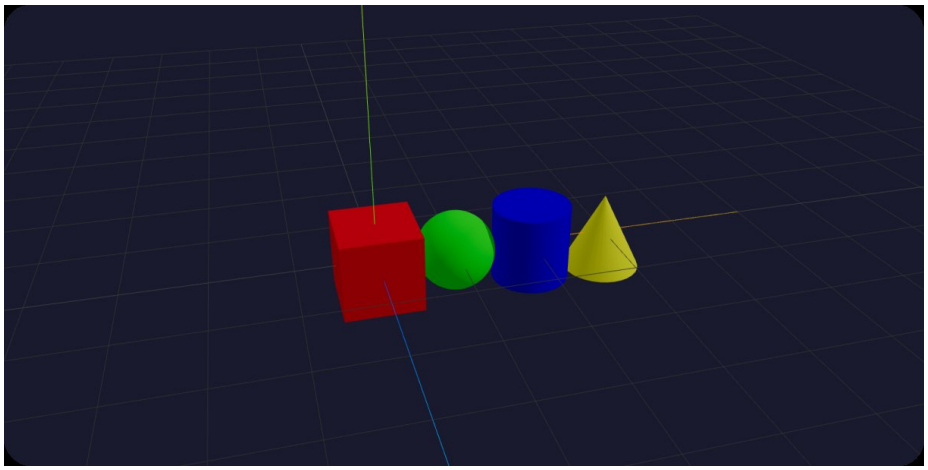
No.	Task
	

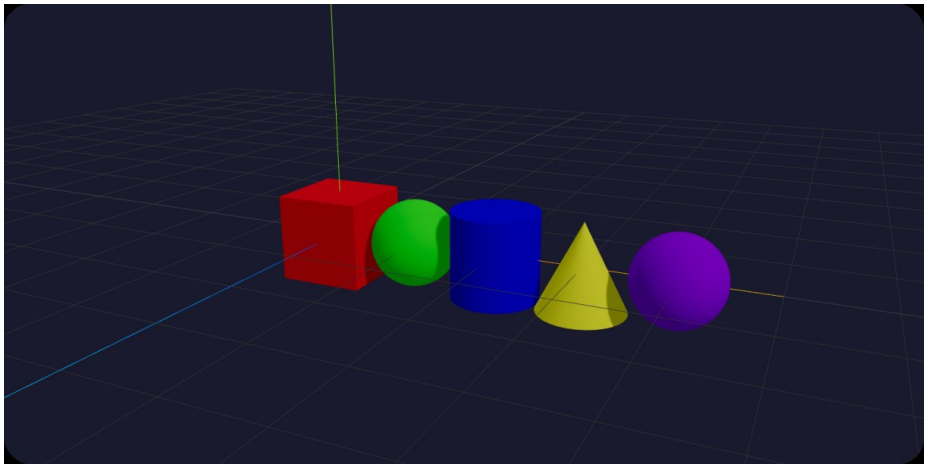
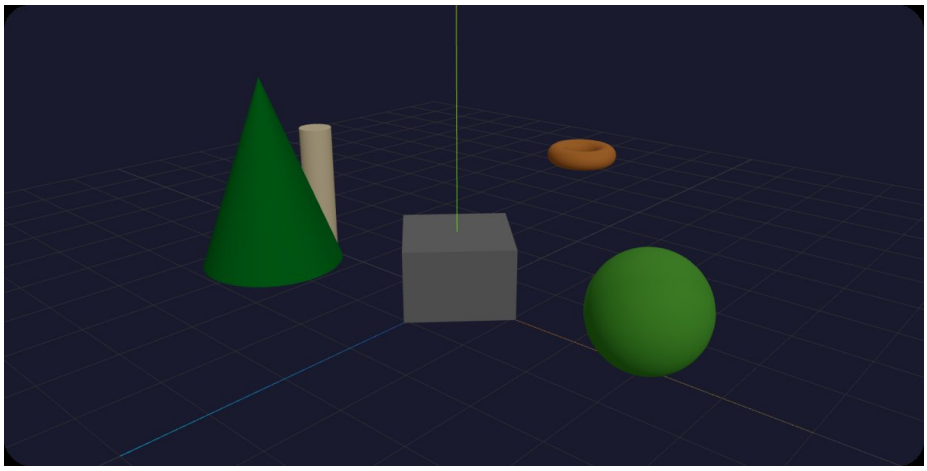
17	`# Complete the Top-Down Plus`
	
	`## GOAL`
	`Look at the scene from above. The four existing objects form an incomplete plus sign - one arm of the plus is missing. Add a single cube that visually completes the plus as seen from the +Y (top-down) direction.`
	
	`## STEPS`
	`1. Open the "Four Objects Scene".`
	`2. From top down view figures form a plus sign without one arm. Identify which arm of the top-down plus shape is missing.`
	`3. Add a new object named "Plug" in the missing arm position so that, viewed from directly above (+Y), the five objects form a symmetric plus sign. The new object's shape must match the existing object on the opposite arm of the plus, while keeping default rotation and unit scale.`
	`4. Place the "Plug" object on the same Y-axis level as all the other objects and color it purple (#aa00ff).`
	
	`# RESULT FORMAT`
	
	`‘‘‘json`
	`{"answer": "done"}`
	`‘‘‘`

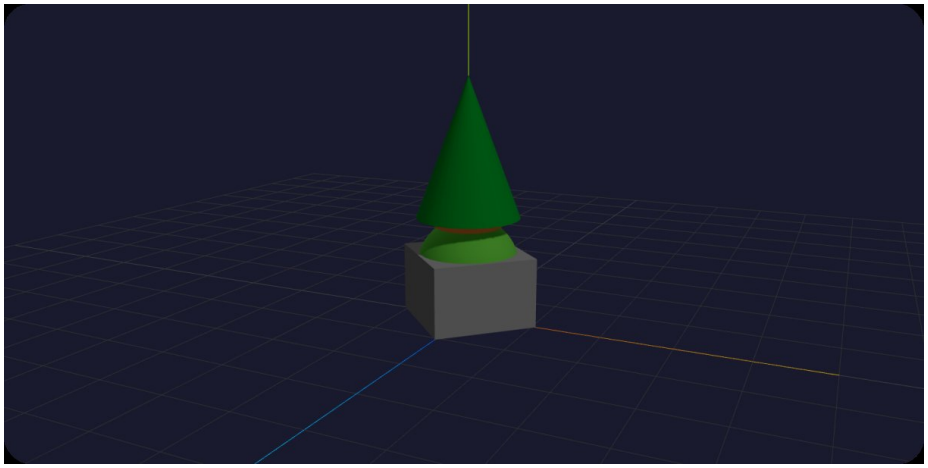
	Initial state
	
	Ground truth

No.	Task
	

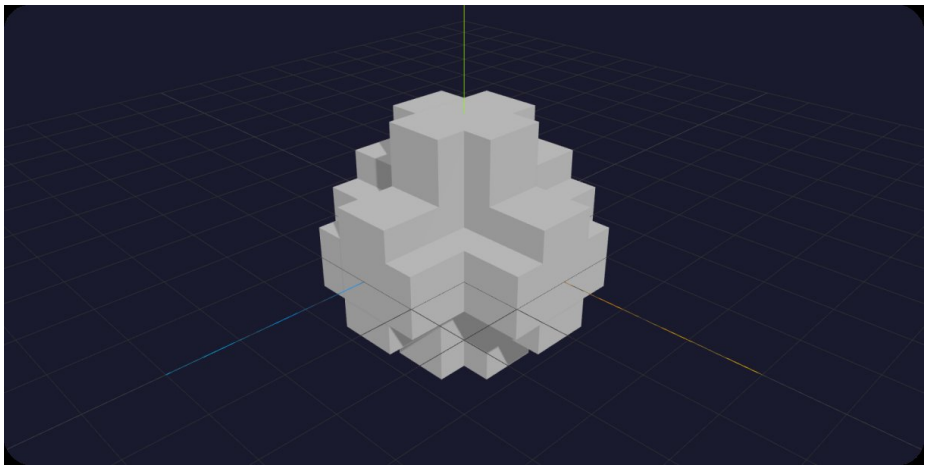
18	`# Fill the Missing End`
	
	`## GOAL`
	`Add one object that visually extends the row pattern, without disturbing the existing row.`
	
	`## STEPS`
	`1. Open the "Row of Objects Scene".`
	`2. In the default camera, observe the four objects as a left-to-right row of distinct silhouettes.`
	`3. Add exactly one new object named "Row Extension" at the next equally spaced position beyond the current rightmost visible silhouette, continuing the same straight row and center-to-center spacing.`
	`4. The new object's must:`
	`* match the shape of the second (starting from the origin) object in the row,`
	`* match the row's Y and Z levels, use default rotation and unit scale,`
	`* have a purple (#aa00ff) color.`
	`5. Preserve every original row object's position, rotation, scale, color, name, visibility, opacity, and wireframe setting.`
	
	`# RESULT FORMAT`
	
	``json
	{"answer": "done"}
	``

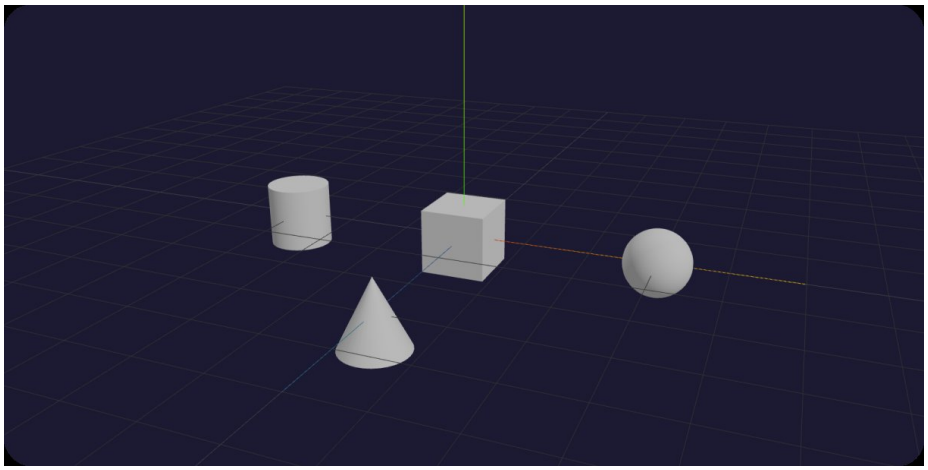
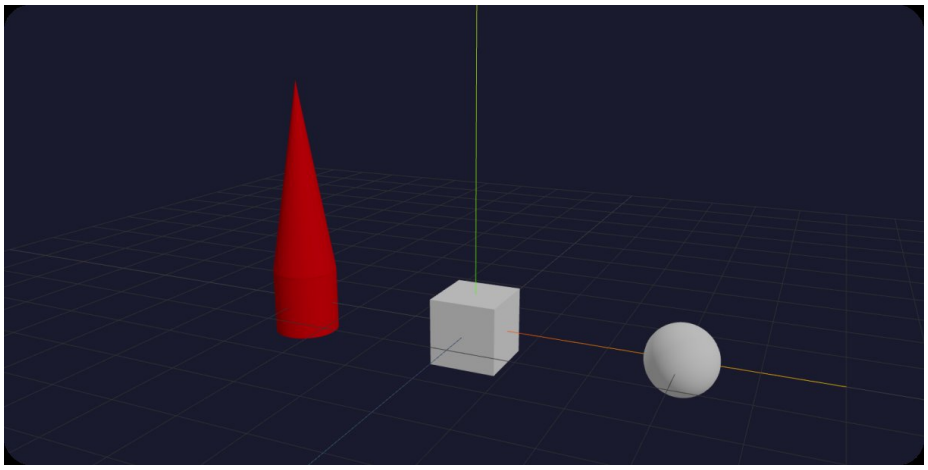
	Initial state
	
	Ground truth

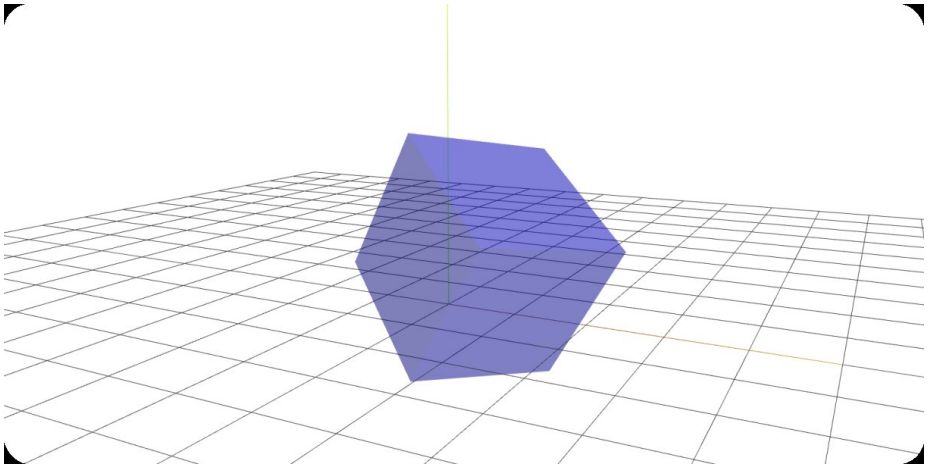
No.	Task
	
19	`# Minimize Total Exposed Volume` `## GOAL` `Reposition objects in the XZ plane to minimize total exposed volume by maximizing overlap, keeping Y positions and all rotations/scales fixed.` `## STEPS` `1. Open the "Geometric Garden".` `2. Use the stone block as the reference. Adjust the positions of the other objects along the X- and Z-axes while keeping their Y-axis positions fixed, with all rotations and scales unchanged. The objective is to minimize the total exposed volume of all objects, excluding any overlapping volume from the calculation.`  `# RESULT FORMAT` ```json {"answer": "done"} ```
	Initial state
	
	Ground truth

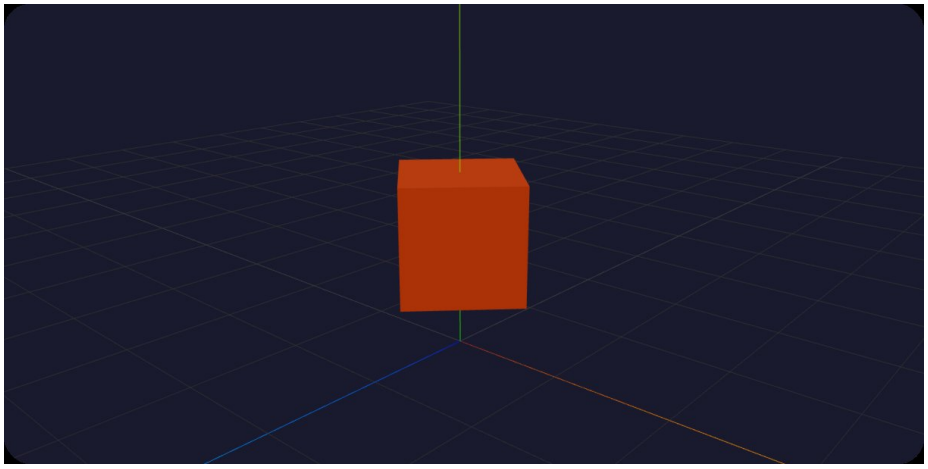
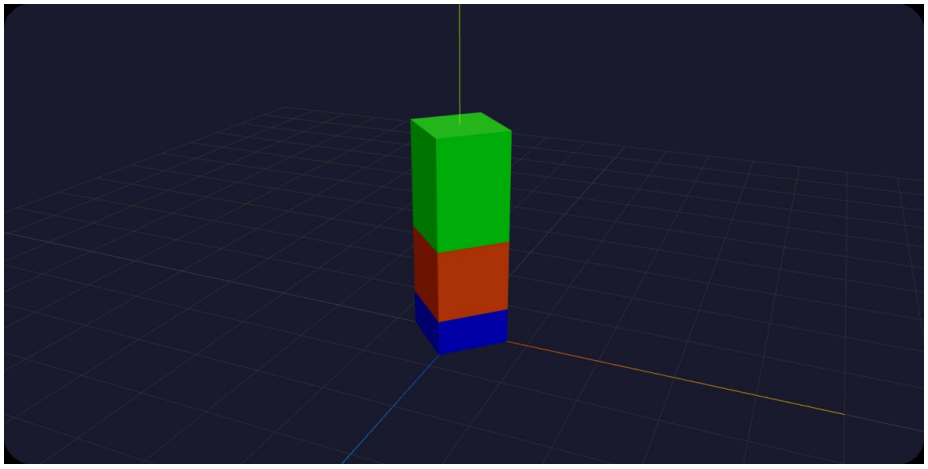
No.	Task
	

20	`# Create Cube (Cuboid) with Scale Combinations`
	
	`## GOAL`
	
	`Find all possible integer scale combinations of a cube using the values 1, 2, and 4 on the x, y, and z axes. The three scale values must be assigned in all possible permutations, and each axis may be either positive or negative.`
	
	`## STEPS`
	
	`1. Open the "Empty Scene".`
	
	`2. Create cubes (cuboids) for all possible combinations of scale values [1, 2, 4] across the x, y, and z axes, allowing each axis to be positive or negative.`
	
	`3. Set the position of each cube to (0, 0, 0) and assign all of them the same name, Cube 1.`
	
	`## RESULT FORMAT`
	
	```` ```json ````
	
	`{"answer": "done"}`
	```` ``` ````

	Ground truth
	

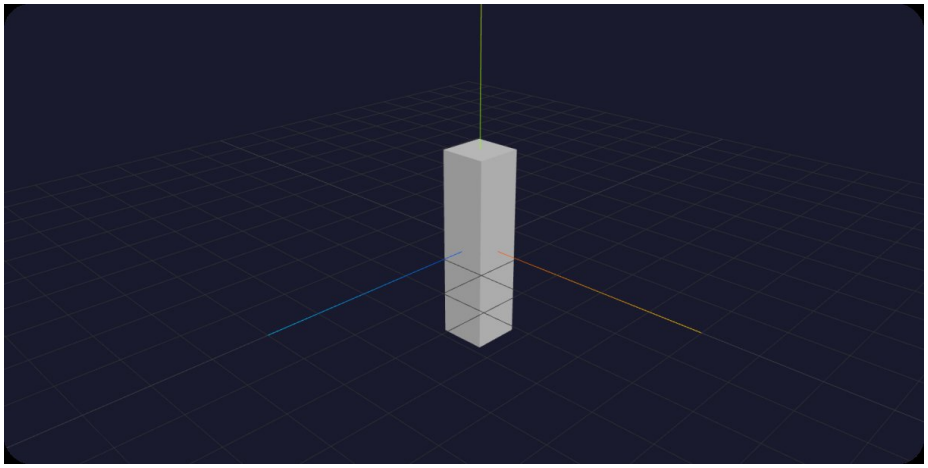
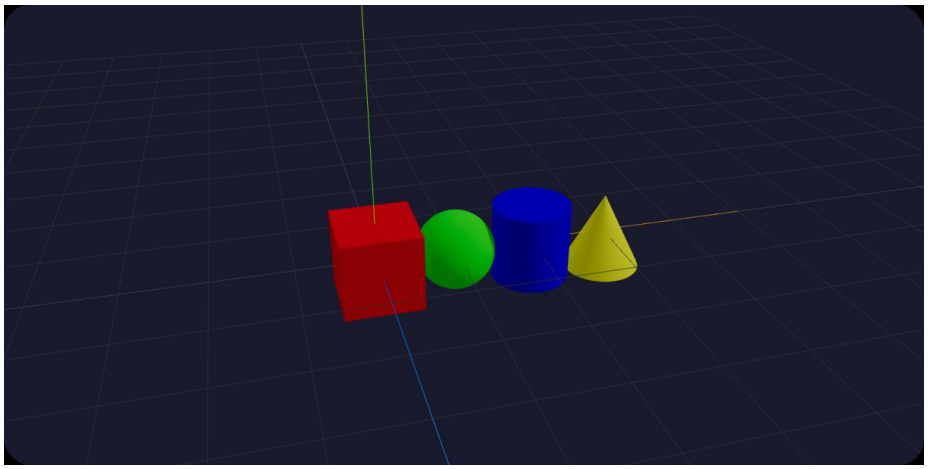
No.	Task
21	# Match Cone Volume to Cylinder and Stack ## GOAL Adjust a cone's Y-axis scale to match the cylinder's volume, place the cone on top of the cylinder, and recolor both to red. ## STEPS 1. Open the ''Four Objects Scene''. 2. Adjust the cone so that its volume matches that of the cylinder by modifying only the scale along the Y-axis. The volume of the cylinder is given by $V = \pi r^2 h$, and the volume of the cone is given by $V = (1/3)\pi r^2 h$. Determine the required Y-axis scaling factor to achieve equal volumes. Next, position the cone on top of the cylinder without any overlap. Finally, treat the cylinder and cone as a single object and change its color to red (#ff0000). # RESULT FORMAT '''json {"answer": "done"} '''
	Initial state
	
	Ground truth
	

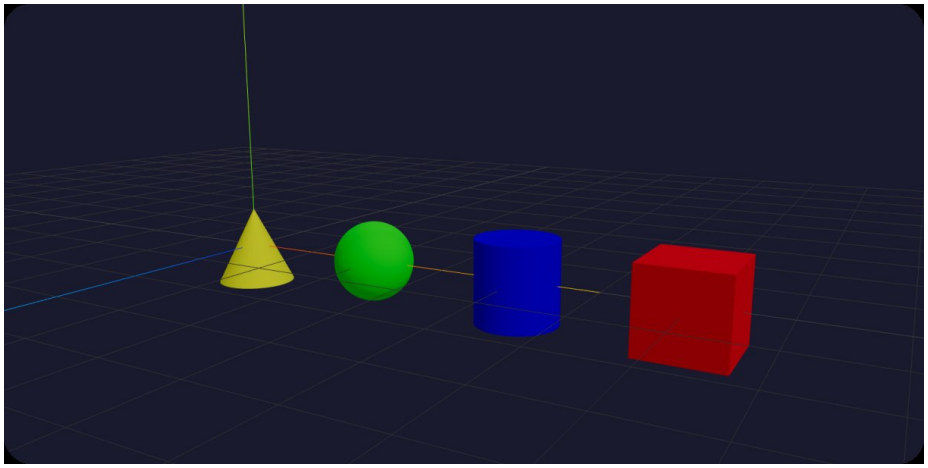
No.	Task
22	# Create a Minimum-Surface-Area Cube (Cuboid) ## GOAL Create a cube with a volume of 8 and the smallest possible surface area, then set specific position, rotation, color, opacity, and background. ## STEPS 1. Open the ‘‘Empty Scene’’. 2. Create a cube (cuboid) with integer side lengths a, b, and c such that the volume constraint abc = 8 is satisfied. The surface area is given by the formula S = 2(ab + bc + ac). Determine the integer dimensions that minimize the surface area, and construct the cuboid accordingly. 3. Set the cuboid’s position to (1, 1, 1), rotation to (45, 45, 45), color to blue (#0000ff), and opacity to 0.5. 4. Set the background color to white (#ffffff). # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	
23	# Stack Three Cubes into a Larger Cube (Cuboid) ## GOAL Add two cubes (cuboids) to an existing scene so that all three cubes together form a single larger cube stacked along the Y-axis with no overlap. ## STEPS 1. Open the "Single Cube Scene". 2. Add two cubes (cuboids) that, together with the given Cube 1, form a unified larger cube (cuboid) with the same rotation. The bottom cubes (cuboids) should have its lower surface in contact with the XZ plane, and the total height along the Y-axis should be 3. Ensure that there is no overlap among the three individual cubes (cuboids). Name the bottom cube Cube 2 and set its color to blue (#0000ff). Name the top cube Cube 3 and set its color to green (#00ff00). # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state

No.	Task
	
	Ground truth
	

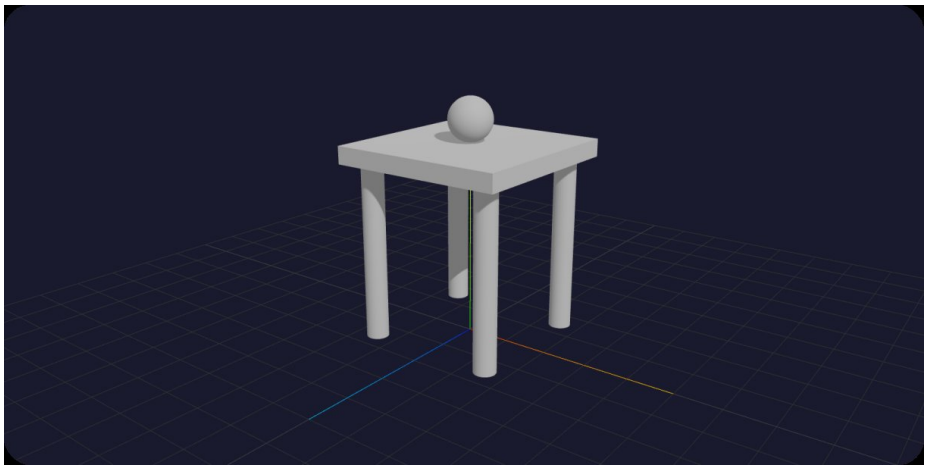
24	`# Iterative Volume-Based Filtering and Scaling`
	
	`## GOAL`
	`Iteratively remove the object with the smallest volume, double the Y-axis scale of the remaining objects, and repeat until only one object remains, then set its color to white.`
	
	`## STEPS`
	`1. Open the “Row of Objects Scene”.`
	`2. Compute the volumes of all objects (cuboid, cone, ellipsoid, cylinder). The relevant surface area formulas are:`
	`* Cube (Cuboid): V = a * b * c`
	`* Cylinder: V = pi * r^2 * h`
	`* Sphere (Ellipsoid): V = 1/6 * pi * a * b * c`
	`* Cone: V = 1/3 * pi * r^2 * h`
	`3. Identify and delete the object with the smallest volume.`
	`4. Double the Y-axis scale of all remaining objects.`
	`5. Repeat Steps 2-4 until only one object remains.`
	`6. Set the remaining object’s color to white (255, 255, 255).`
	
	`# RESULT FORMAT`
	
	`“‘json`
	`{"answer": "done"}`
	`“‘`

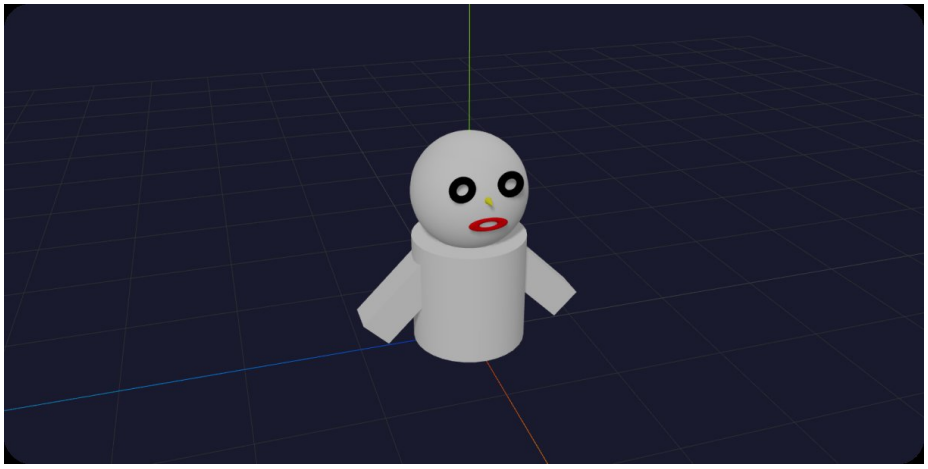
	Ground truth

No.	Task
	
25	# Reorder by Surface Area Plus Volume ## GOAL Sort four elements along the positive X-axis by the sum of their surface area and volume (smallest to largest), preserving the original color order. ## STEPS 1. Open the ‘‘Row of Objects Scene’’. 2. Reorder the given elements (cube (cuboid), sphere, cylinder, cone) along the positive X-axis, arranging them from smallest to largest based on the sum of their surface area and volume. The relevant formulas are: * Cube (cuboid): surface area S = 2(ab + bc + ac), volume V = abc * Sphere: surface area $S = 4\pi r^2$, volume $V = (4/3)\pi r^3$ * Cylinder: surface area $S = 2\pi r^2 + 2\pi rh, volume V = \pi r^2 h$ * Cone: surface area $S = \pi r^2 + \pi rl, volume V = (1/3)\pi r^2 h$ (where l is the slant height) Preserve the original color order while reordering. 3. Distance between centers of consecutive object must me 2. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Initial state
	
	Ground truth

No.	Task
	

26	`# Build a Ball on Table (Open-Ended)`
	
	`## GOAL`
	`Construct a table from a tabletop cube and four cylinder legs, then place a ball on the tabletop, computing positions so all parts are in contact.`
	
	`## STEPS`
	`1. Create a new scene named “Ball on Table.” Then, construct a 3D table and place a ball on the tabletop.`
	`The table consists of the following components: a cube named Tabletop, which serves as the reference point, with its position set to (0, 4.2, 0) and scaled to (4, 0.4, 4); four cylinders of equal size named TableLeg, each scaled to (0.5, 4, 0.5); and a sphere named Ball with default settings.`
	`These components constitute all the elements required to construct the “Ball on Table” scene. Your task is to determine their positions (except for the Tabletop, which is fixed as the reference). Each element should be connected to or tangent to the others.`
	
	`# RESULT FORMAT`
	
	“‘`json`
	`{"answer": "done"}`
	“‘

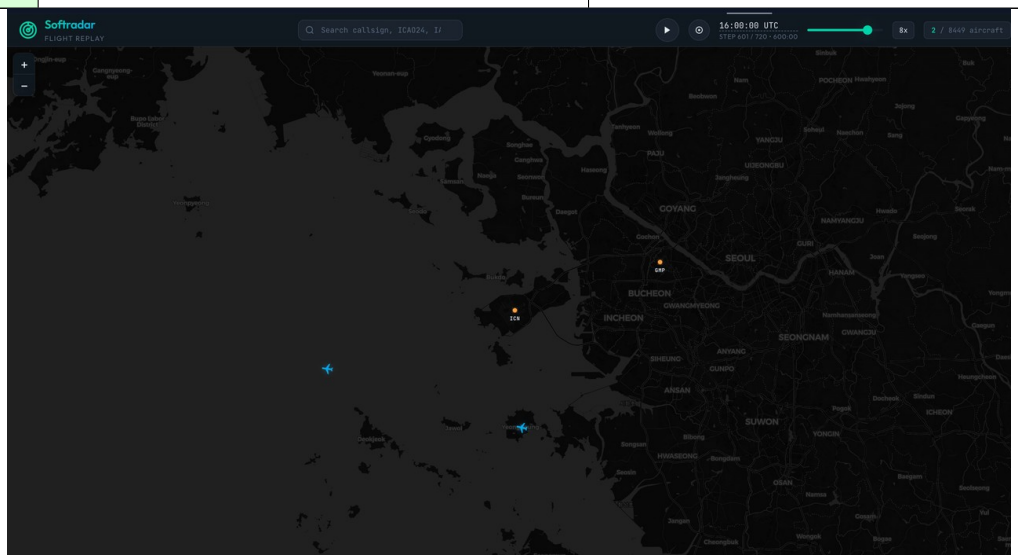
	Ground truth
	

No.	Task
27	# Build a 3D Snowman ## GOAL Build a 3D snowman from scratch using multiple primitives (cylinder body, sphere head, torus eyes, cone nose, torus mouth, cube arms) with precise transforms. ## STEPS 1. Create a new scene named "Snowman". 2. Construct a 3D snowman. First, add a cylinder and name it Body. Set its position to (0, 0.5, 0), using the default rotation, scale, color, and opacity. Second, add a sphere named Head and set its position to (0, 1.4, 0). Third, add a torus named Right Eye. Set its position to (0.45, 1.6, 0.2), rotation to (0, 90, 0), scale to (0.2, 0.2, 0.2), and color to black (#000000). Fourth, duplicate the Right Eye and rename it Left Eye. Set its position symmetrically along the Z-axis opposite to the right eye. Fifth, add a cone named Nose. Set its position to (0.5, 1.5, 0), rotation to (0, 0, -90), scale to (0.1, 0.2, 0.1), and color to yellow (#ffff00). Sixth, add another torus named Mouth. Set its position to (0.5, 1.3, 0), rotation to (0, 90, 0), scale to (0.3, 0.1, 0.1), and color to red (#ff0000). Seventh, add a cube named Right Arm. Set its position to (0, 0.5, 0.6), rotation to (-45, 0, 0), and scale to (0.3, 0.8, 0.3). Finally, duplicate the Right Arm and rename it Left Arm. Set its position symmetrically along the Z-axis opposite to the right arm, and apply the opposite rotation along the X-axis. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘
	Ground truth
	

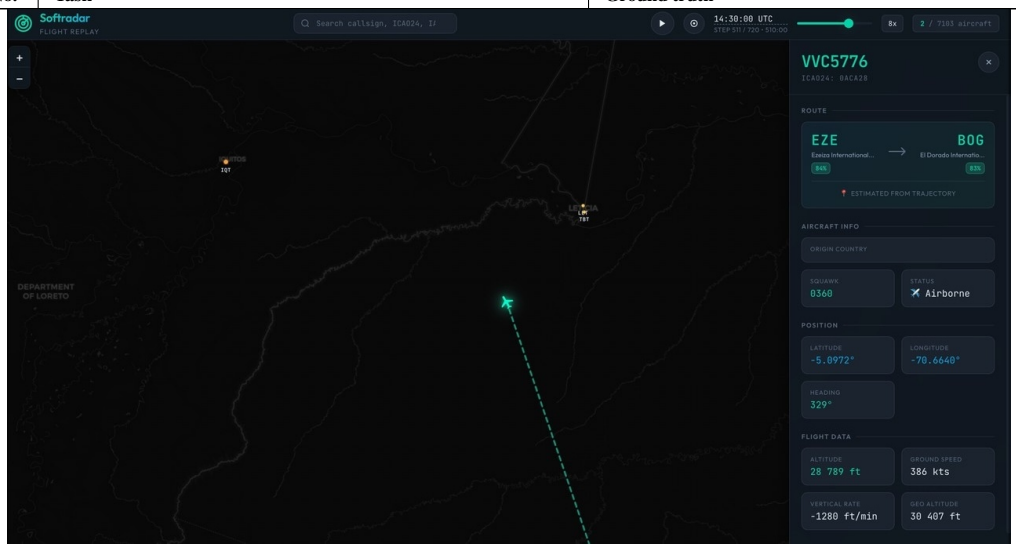
G.4 FLIGHT ANALYSER

No.	Task	Ground truth

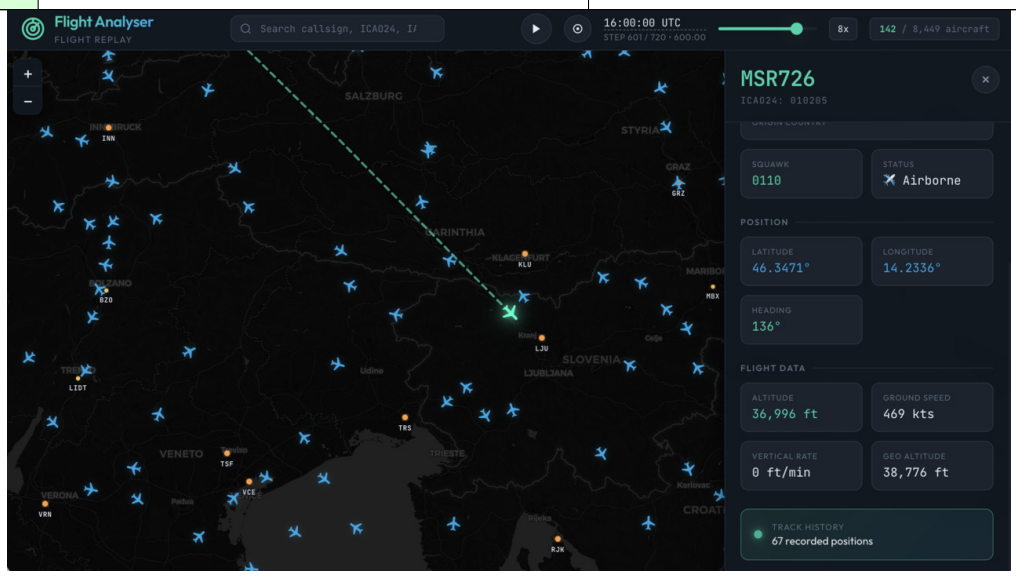
1	# Count Visible Aircraft at a Specific Zoom Level
	
	## GOAL
	Count how many aircraft icons are visible on screen after centering on an airport and zooming in.
	
	## STEPS
	1. Set the playback time to 14:00 UTC.
	2. Search for airport ICN to center the map on it.
	3. Zoom in exactly 3 steps.
	4. Count the total number of aircraft icons currently visible on the screen.
	
	# RESULT FORMAT
	
	``json
	{
	"count": <N>
	}
	``
		{
		"count": 2
		}

2	# Read Telemetry at a Specific Time
	
	## GOAL
	Look up a flight's geo altitude and ground speed at a precise moment.
	
	## STEPS
	1. Set the playback time to 14:30 UTC.
	2. Pause the playback.
	3. Search for flight VVC5776 using the search bar.
	4. Open its telemetry data.
	5. Read and report the exact geo altitude and ground speed values.
	
	# RESULT FORMAT
	
	``json
	{
	"geo_altitude": "<value>",
	"ground_speed": "<value>"
	}
	``
		{
		"geo_altitude": "30,407 ft",
		"ground_speed": "386 kts"
		}

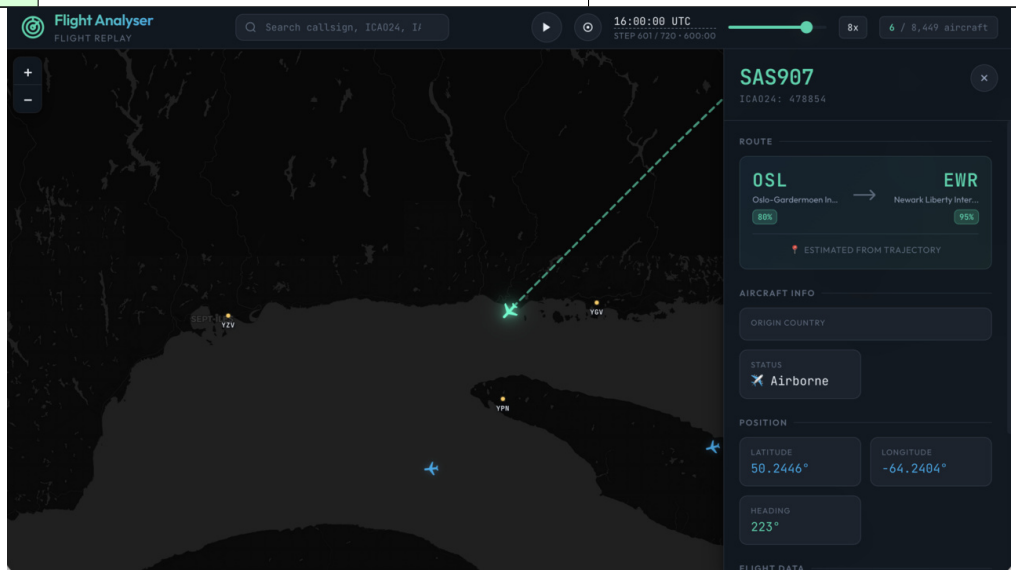
No.	Task	Ground truth
		

3	# Read Geo Altitude and Vertical Rate
	
	## GOAL
	Look up a single flight's geo altitude and vertical rate at a precise moment.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Pause the playback.
	3. Search for flight MSR726 using the search bar.
	4. Open its flight details and scroll to the "Flight Data" section.
	5. Read and report the geo altitude and vertical rate values.
	
	# RESULT FORMAT
	
	``json
	{
	"geo_altitude": "<value>",
	"vertical_rate": "<value>"
	}
	``
		{
		"geo_altitude": "38,776 ft",
		"vertical_rate": "0 ft/min"
		}

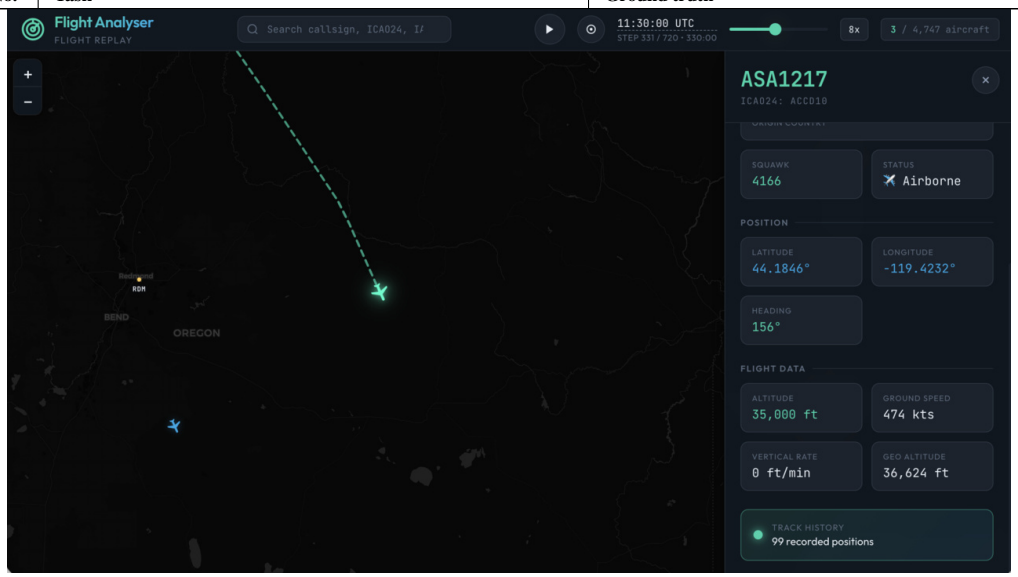
		

No.	Task	Ground truth

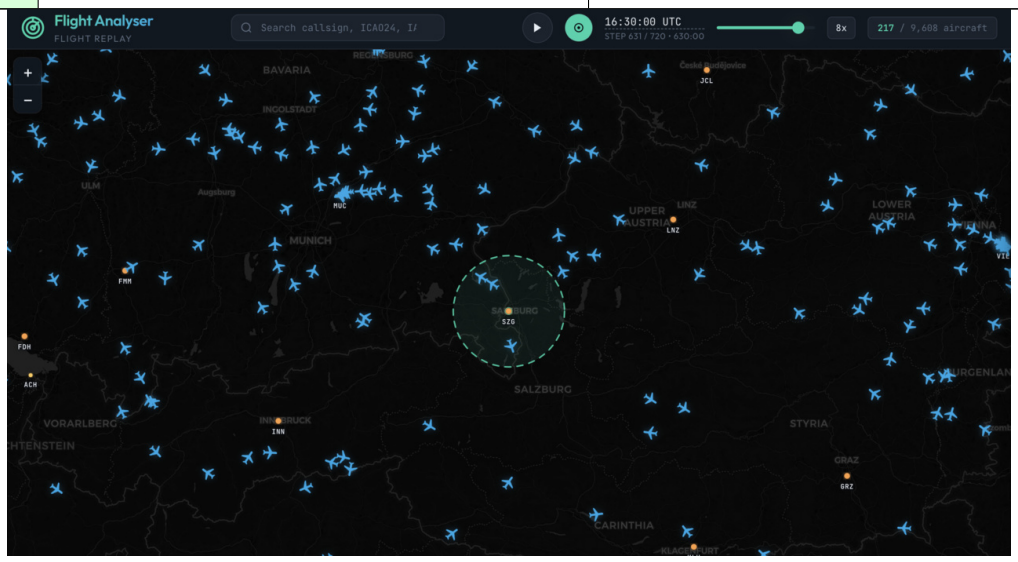
4	# Read a Flight's Route Endpoints
	
	## GOAL
	Report the origin and destination airports of a single flight.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Search for flight SAS907 using the search bar.
	3. Open its flight details and read the route.
	4. Report the origin and destination airport codes exactly as shown.
	
	# RESULT FORMAT
	
	``json
	{
	"origin": "<origin>",
	"destination": "<destination>"
	}
	``
		{
		"origin": "OSL",
		"destination": "EWR"
		}

5	# Read a Flight's Ground Speed at Two Times
	
	## GOAL
	Read the same telemetry field for one flight at two different playback times.
	
	## STEPS
	1. Set the playback time to 11:15 UTC and pause the playback.
	2. Search for flight ASA1217 using the search bar and open its flight details.
	3. Read its ground speed and note it down.
	4. Set the playback time to 11:30 UTC, and read the ground speed of the same flight again.
	5. Report both values exactly as shown.
	
	# RESULT FORMAT
	
	``json
	{
	"ground_speed_1115": "<value>",
	"ground_speed_1130": "<value>"
	}
	``
		{
		"ground_speed_1115": "480 kts",
		"ground_speed_1130": "474 kts"
		}

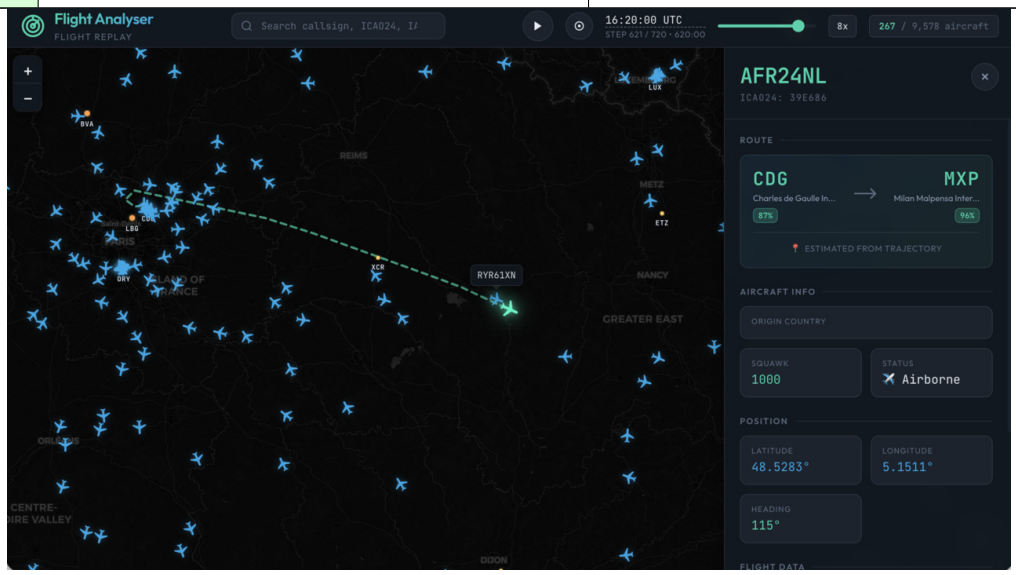
No.	Task	Ground truth
		

6	`# Count Aircraft Inside a Radius Around Salzburg`
	
	`## GOAL`
	`Use the radius tool to count the aircraft around a single airport at one moment.`
	
	`## STEPS`
	`1. Set the playback time to 16:30 UTC.`
	`2. Pause the playback.`
	`3. Search for Salzburg Airport (SZG) to center the map on it.`
	`4. Draw a 50 km radius circle around the airport.`
	`5. Count the aircrafts inside the circle.`
	`6. Report that number.`
	
	`# RESULT FORMAT`
	
	```` ```json ````
	`{`
	`"count": <N>`
	`}`
	```` ``` ````
		`{`
		`"count": 3`
		`}`

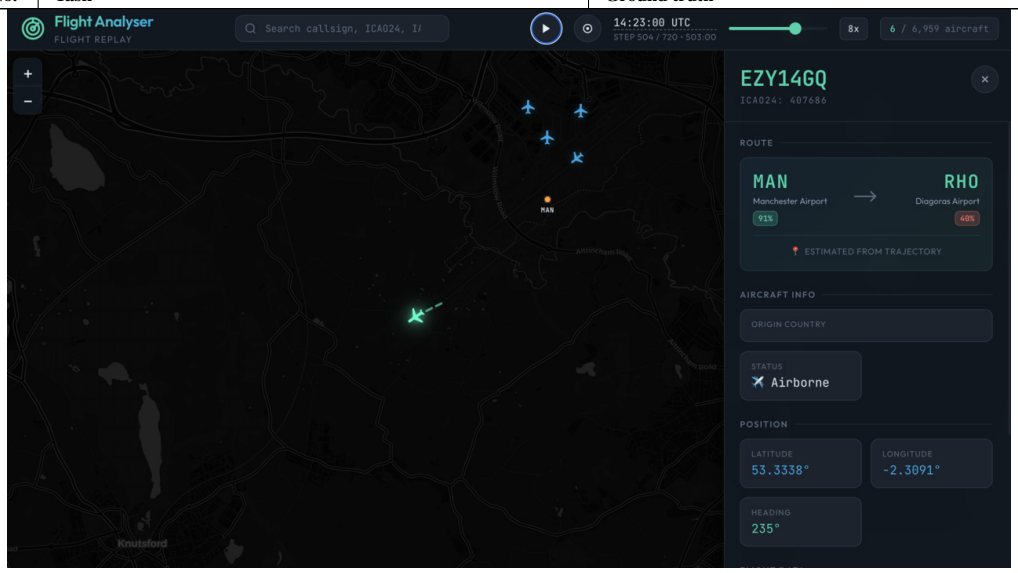
		

No.	Task	Ground truth

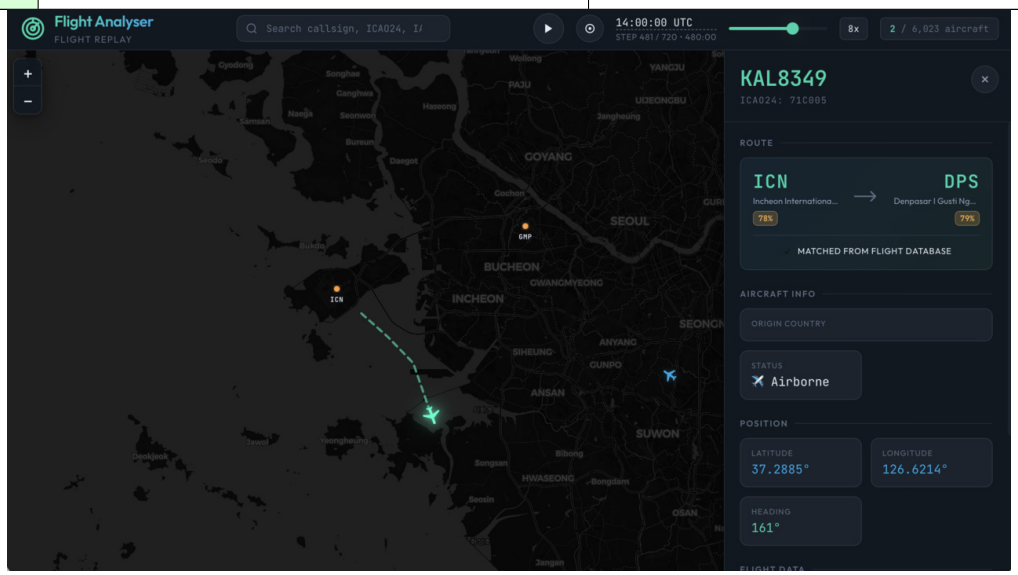
7	# Read the Geo Altitude of the Eastern Flight
	
	## GOAL
	Decide which of two flights is further east on the map, then report the geo altitude of that flight only.
	
	## STEPS
	1. Set the playback time to 16:20 UTC.
	2. Pause the playback.
	3. Search for flight AFR24NL and note its position on the map.
	4. Search for flight RYR61XN and note its position on the map.
	5. Determine which of the two flights is further east.
	6. Open the details of that flight and report its callsign and its geo altitude exactly as shown.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>",
	"geo_altitude": "<value>"
	}
	``
		{
		"callsign": "AFR24NL",
		"geo_altitude": "32,451 ft"
		}

8	# Find the Minute a Flight Leaves the Ground
	
	## GOAL
	Use the timeline to find the exact minute at which a flight's status changes.
	
	## STEPS
	1. Set the playback time to 14:22 UTC and pause the playback.
	2. Search for flight EZY14GQ and open its flight details. At this moment its status is "On Ground".
	3. Wait until the status is no longer "On Ground".
	4. Report the first UTC minute at which the new status appears, and that status exactly as shown.
	
	# RESULT FORMAT
	
	``json
	{
	"status_change_utc": "<HH:MM UTC>",
	"new_status": "<status>"
	}
	``
		{
		"status_change_utc": "14:23 UTC",
		"new_status": "Airborne"
		}

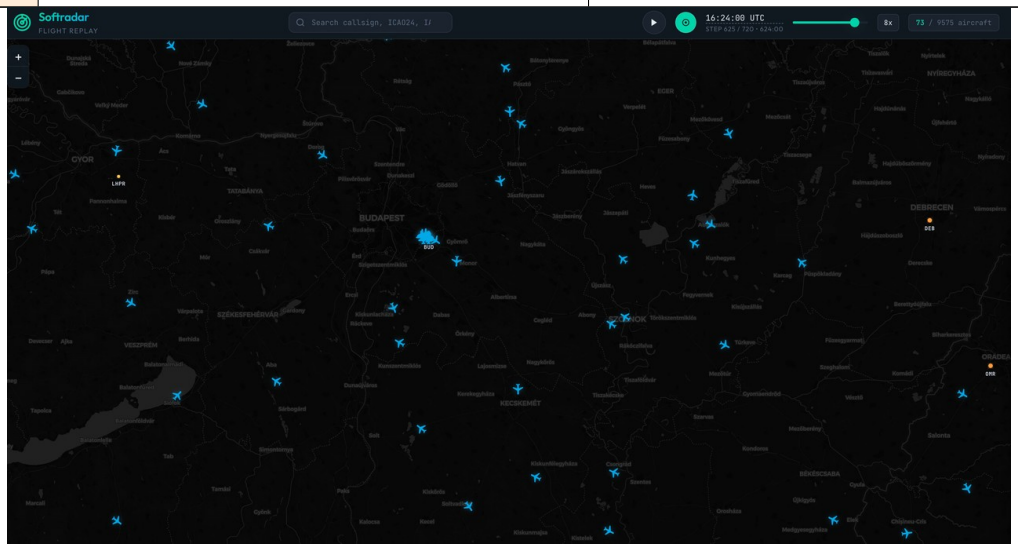
No.	Task	Ground truth
		

9	# Identify an Aircraft by Clicking It on the Map
	
	## GOAL
	Open a flight's details by clicking its icon on the map instead of searching for its callsign.
	
	## STEPS
	1. Set the playback time to 14:00 UTC.
	2. Search for airport ICN to center the map on it.
	3. Click the aircraft icon that is closest to the airport marker.
	4. Report that aircraft's callsign and its destination airport code exactly as shown.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>",
	"destination": "<destination>"
	}
	``
		{
		"callsign": "KAL8349",
		"destination": "DPS"
		}

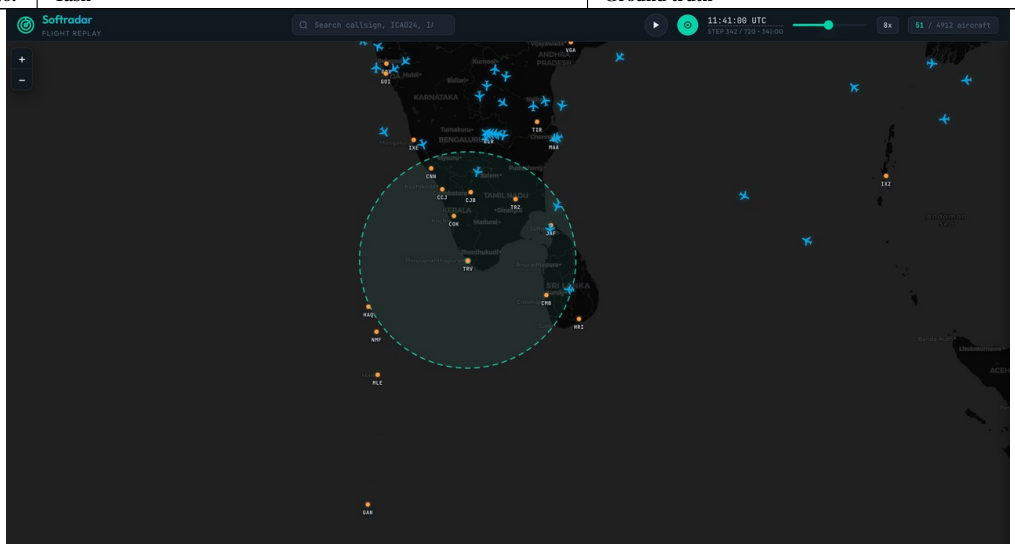
		

No.	Task	Ground truth

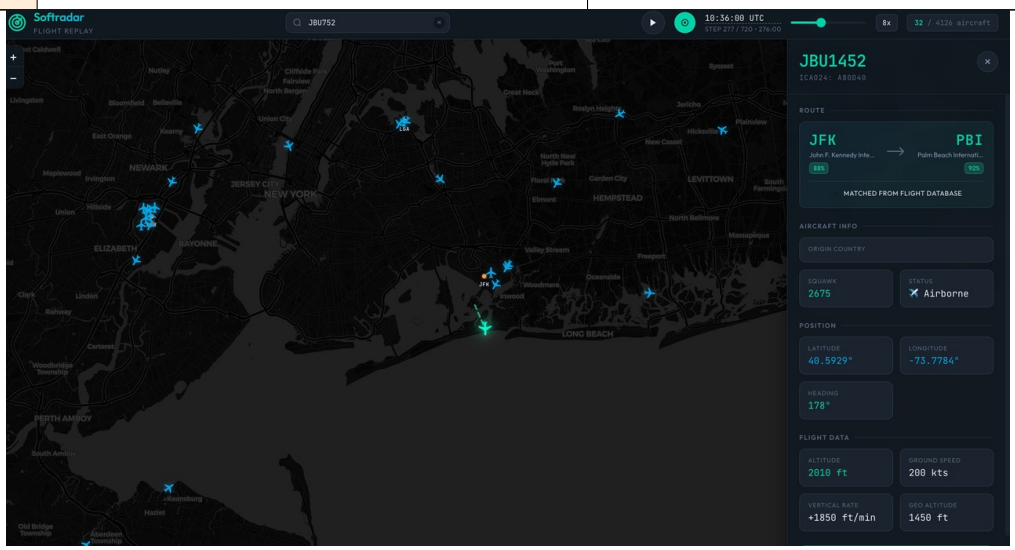
10	# Count Landings at an Airport Over a Time Window
	
	## GOAL
	Observe an airport and count how many aircraft land during a specified period.
	
	## STEPS
	1. Set the playback time to 16:14 UTC.
	2. Navigate to Budapest Ferenc Liszt Airport (BUD).
	3. Observe the airport for 10 minutes of playback time.
	4. Count how many new aircraft land at the airport during this period.
	
	# RESULT FORMAT
	
	``json
	{
	"count": <N>
	}
	``
		{
		"count": 5
		}

11	# Count Aircraft Within a Circle at a Specific Time
	
	## GOAL
	Count how many aircraft are located inside a defined radius around an airport.
	
	## STEPS
	1. Set the playback time to 11:41 UTC.
	2. Search for Thiruvananthapuram International Airport (TRV) to center the map on it.
	3. Draw a 450 km radius circle centered on the airport.
	4. Count the number of aircraft located within this circle.
	
	# RESULT FORMAT
	
	``json
	{
	"count": <N>
	}
	``
		{
		"count": 4
		}

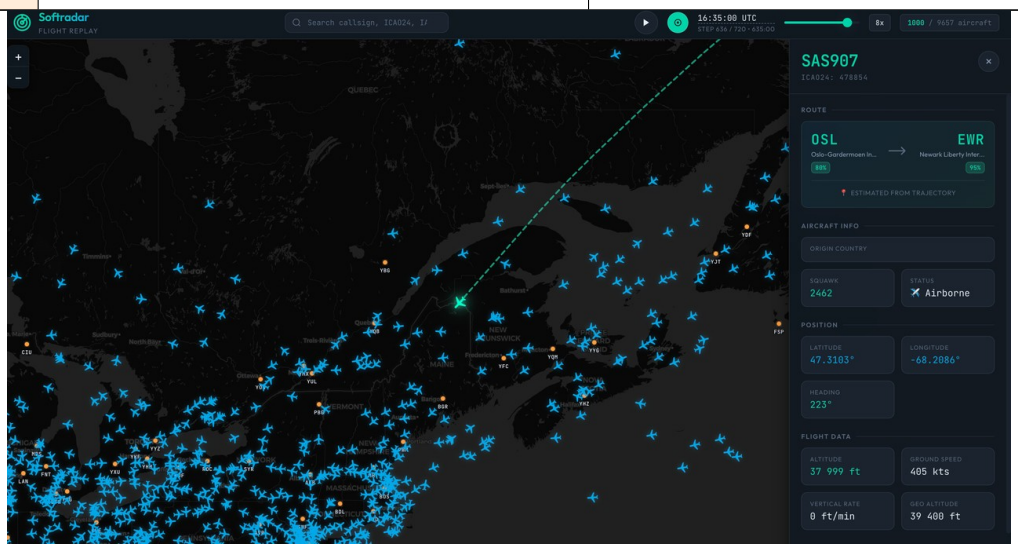
No.	Task	Ground truth
		

12	# Identify the First Departure After a Given Time
	
	## GOAL
	Find the first flight that departs from an airport after a specific time.
	
	## STEPS
	1. Set the playback time to 11:30 UTC.
	2. Zoom into John F. Kennedy International Airport (JFK) in New York.
	3. Observe departures from the airport.
	4. Identify the first flight that departs after this time.
	5. Report that flight's callsign.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>"
	}
	``
		{
		"callsign": "AAL1535"
		}

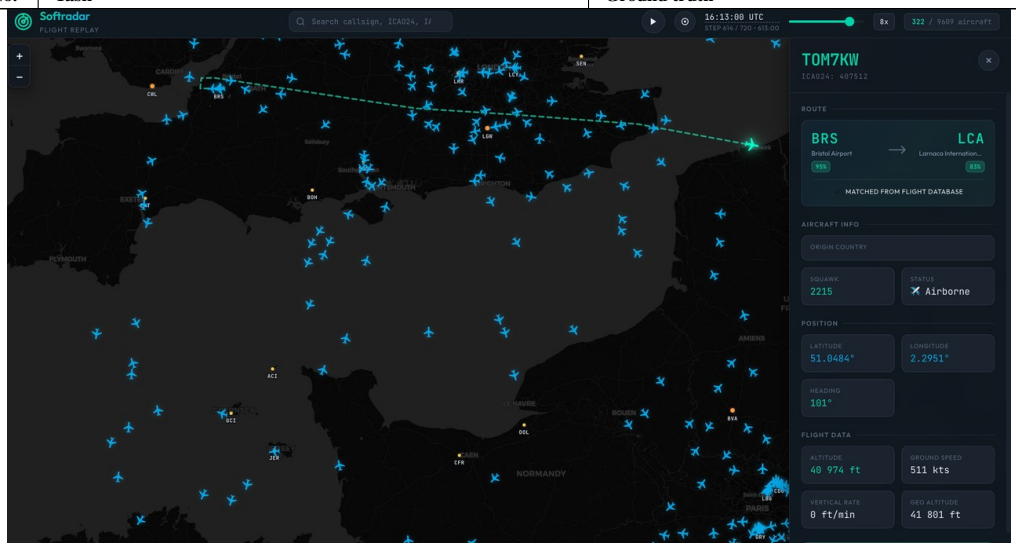
		

No.	Task	Ground truth

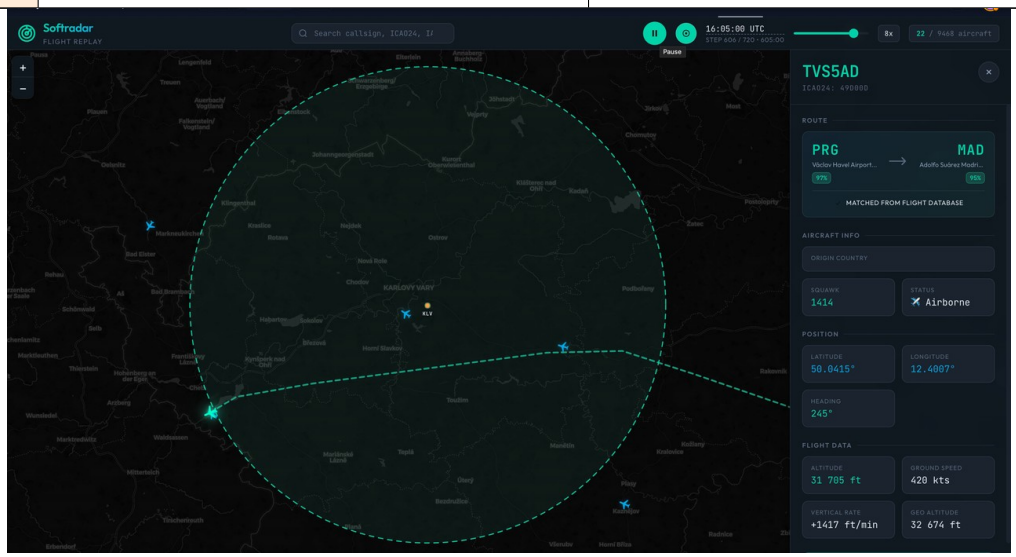
13	# Determine a Border Crossing Time (Canada to US)
	
	## GOAL
	Track a flight and report the exact time it crosses the Canada-United States border.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Search for flight SAS907 using the search bar.
	3. Locate the flight on the map and begin tracking it forward in time.
	4. Report the UTC time at which it crosses the Canada-United States border.
	
	# RESULT FORMAT
	
	``json
	{
	"crossing_time_utc": "<HH:MM UTC>"
	}
	``
		{
		"crossing_time_utc": "16:35 UTC"
		}

14	# Identify the First Channel Crossing (UK to France)
	
	## GOAL
	Find the first flight to cross from England into France over the English Channel after a given time.
	
	## STEPS
	1. Set the playback time to 16:08 UTC.
	2. Zoom into the narrowest part of the English Channel.
	3. Identify all flights that were over England at 16:08 UTC.
	4. Observe which of those flights is the first to cross into France.
	5. Report that flight's callsign.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>"
	}
	``
		{
		"callsign": "TOM7KW"
		}

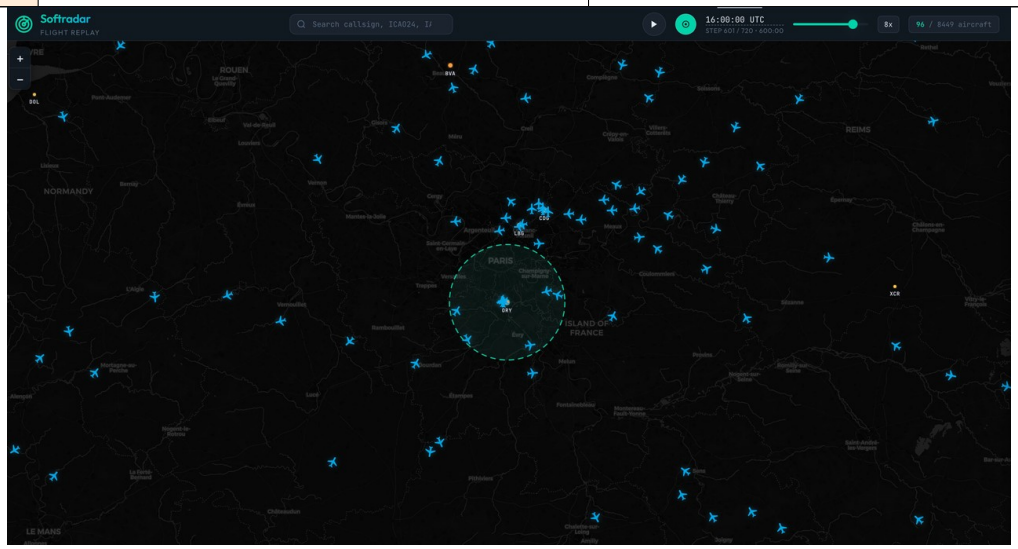
No.	Task	Ground truth
		

15	# Find the Last Flight to Leave a Circle
	
	## GOAL
	Determine which flight is the last to exit a radius around a given airport.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Navigate to Karlovy Vary Airport.
	3. Draw a 40 km radius circle around the airport.
	4. Identify all flights currently inside the circle.
	5. Track them forward in time and determine which flight leaves the circle last.
	6. Report that flight's callsign.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>"
	}
	``
		{
		"callsign": "TVS5AD"
		}

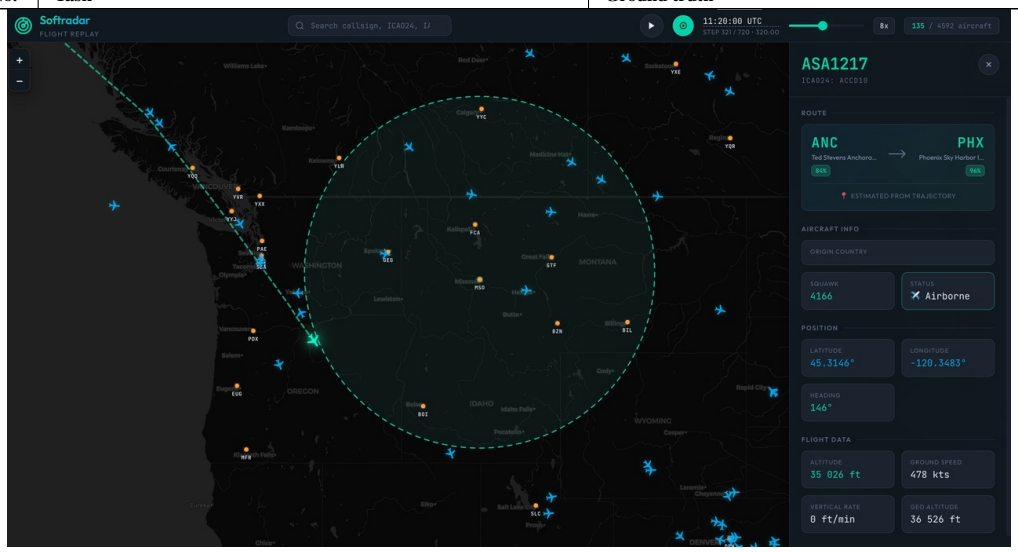
		

No.	Task	Ground truth

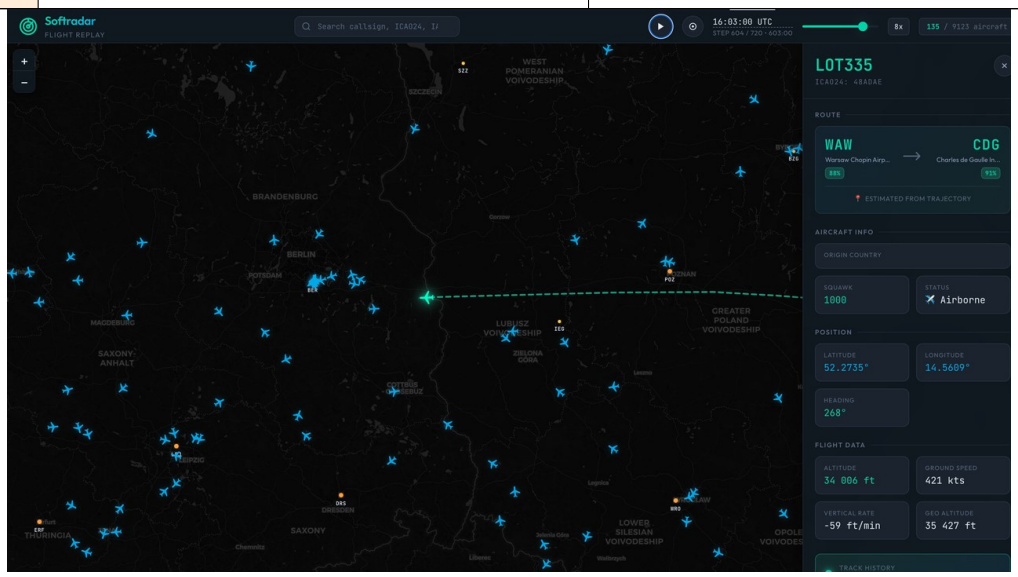
16	# Count Airborne Aircraft Near an Airport
	
	## GOAL
	Count how many aircraft within a radius of a given airport have an "Airborne" status.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Search for airport ORY to center the map on it.
	3. Zoom in exactly 3 steps.
	4. Within a 20 km radius of ORY, open the details of each aircraft in view.
	5. Count how many have an "Airborne" status at 16:00 UTC and collect their callsigns.
	
	# RESULT FORMAT
	
	``json
	{
	"count": <N>,
	"callsigns": ["<CS1>", "<CS2>", "..."]
	}
	``
		{
		"count": 5,
		"callsigns": [
		"CTM3813",
		"TVF3828",
		"EJU54MG",
		"TVF3124",
		"FMY8055"
		]
		}

17	# Check Whether a Flight Path Enters a Circle
	
	## GOAL
	Determine whether a specific flight's path ever enters a defined radius around an airport.
	
	## STEPS
	1. Set the playback time to 11:15 UTC.
	2. Search for Missoula Montana Airport (MSO) to center the map on it.
	3. Draw a 500 km radius circle centered on the airport.
	4. Search for flight ASA1217 and identify the flight closest to ASA1217 at that moment whose track crosses the 500 km radius circle.
	5. Report the callsign of that closest flight.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>"
	}
	``
		{"callsign": N213AL}

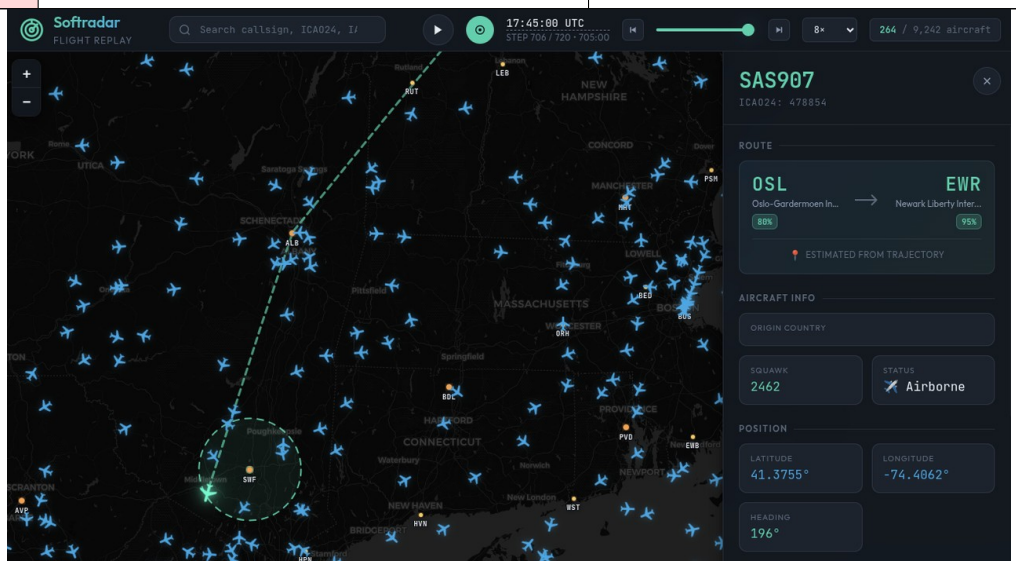
No.	Task	Ground truth
		

18	# Detect a Border Crossing (Poland to Germany)
	
	## GOAL
	Find the first aircraft whose track crosses from Poland into Germany after a given time.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Navigate to the Germany-Poland border region.
	3. Observe aircraft tracks crossing the border.
	4. Identify the first aircraft that crosses from Poland into Germany (not the reverse direction) and does not re-enter Polish airspace afterward.
	5. Confirm the crossing direction by comparing two successive positions - one before and one after the border.
	6. Report that aircraft's callsign.
	
	# RESULT FORMAT
	
	``json
	{
	"callsign": "<callsign>"
	}
	``
		{
		"callsign": "LOT335"
		}

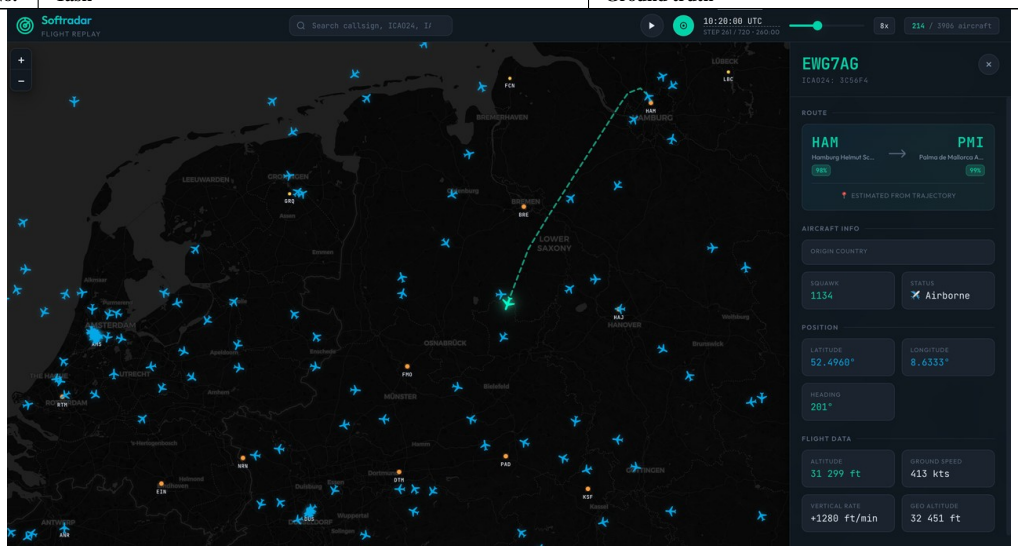
		

No.	Task	Ground truth

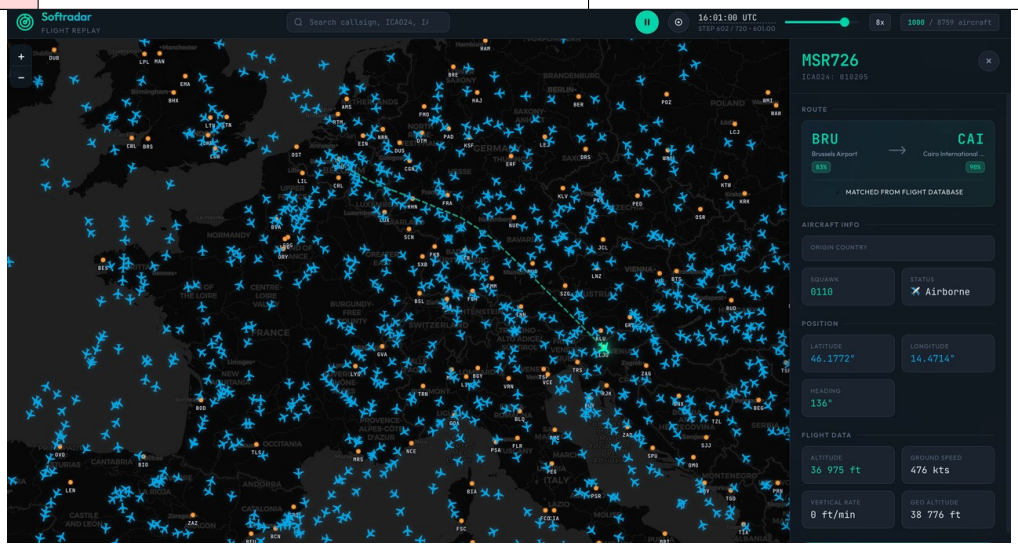
19	# Count Airports Passed Along a Flight Path
	
	## GOAL
	Track a flight over a period and count how many airports it passes within close proximity.
	
	## STEPS
	1. Set the playback time to 17:15 UTC.
	2. Search for flight SAS907 using the search bar and locate it on the map.
	3. Track the flight for the next 30 minutes of playback time.
	4. Count how many airports the flight passes within a 30 km radius and list their airport codes.
	
	# RESULT FORMAT
	
	``json
	{
	"count": <N>,
	"airports": ["<airport1>", "<airport2>", "..."]
	}
	``
		{
		"count": 3,
		"airports": [
		"RUT",
		"ALB",
		"SWF"
		]
		}

20	# Detect the First Stable East-to-West Ordering of Three Flights
	
	## GOAL
	Determine the first time three aircraft reach a specified west-to-east ordering and keep it for a sustained period.
	
	## STEPS
	1. Set the playback time to 16:20 UTC.
	2. Search for flights AFR24NL, RYR61XN, and BELOGA and track them forward in time.
	3. Find the first UTC minute when the west-to-east order of the flights becomes BELOGA -> AFR24NL -> RYR61XN.
	4. Report that UTC time.
	
	# RESULT FORMAT
	
	``json
	{
	"order_start_utc": "<HH:MM UTC>"
	}
	``
		{
		"order_start_utc": "16:27 UTC"
		}

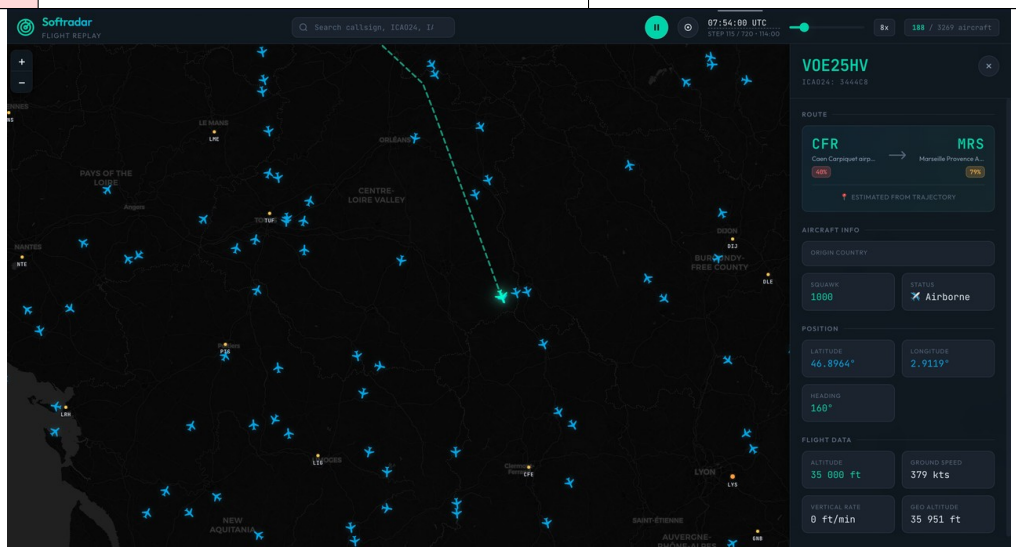
No.	Task	Ground truth
		

21	# Count Country Borders Crossed from Track History
	
	## GOAL
	Determine how many country borders a flight has crossed based on its visible track history.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Search for flight MSR726 using the search bar.
	3. Examine the flight's visible track history (trajectory) as shown at the current moment.
	4. Count how many country borders the track crosses and list each country entered.
	
	# RESULT FORMAT
	
	‘‘‘json
	{
	"num_country_borders": <N>,
	"countries": ["<country1>", "<country2>", "..."]
	}
	‘‘‘
		{
		"num_country_borders": 4,
		"countries": [
		"Belgium",
		"Germany",
		"Austria",
		"Slovenia"
		]
		}

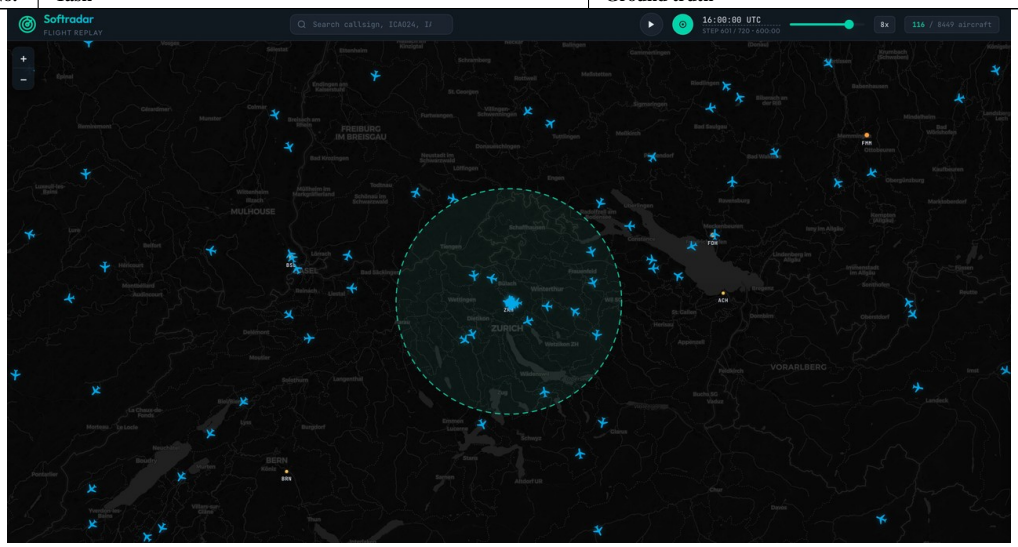
		

No.	Task	Ground truth

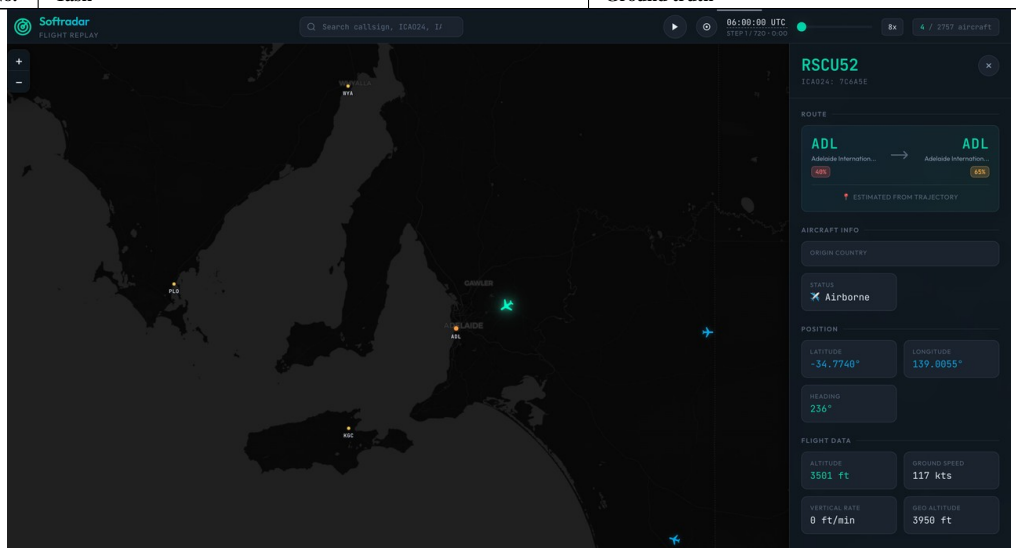
22	# Find the Closest Aircraft and Its Overtake Time
	
	## GOAL
	Locate the aircraft nearest to a given flight and determine when it overtakes that flight.
	
	## STEPS
	1. Set the playback time to 07:50 UTC.
	2. Search for flight VOE25HV using the search bar.
	3. Identify the aircraft currently closest to VOE25HV.
	4. Track both aircraft forward in time.
	5. Report the closest aircraft's callsign and the UTC time at which it overtakes (passes ahead of) VOE25HV.
	
	# RESULT FORMAT
	
	``json
	{
	"closest_callsign": "<callsign>",
	"overtake_time_utc": "<HH:MM UTC>"
	}
	``
		{
		"closest_callsign": "N656FN",
		"overtake_time_utc": "07:54 UTC"
		}

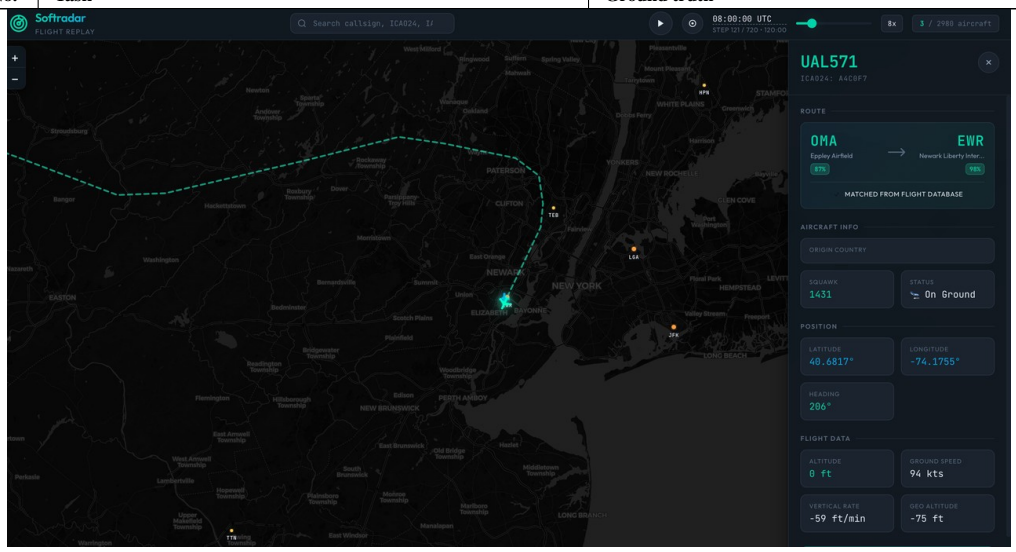
23	# Find the Closest Aircraft Pair Near an Airport
	
	## GOAL
	Among airborne aircraft near Zurich, find the two that are closest to each other.
	
	## STEPS
	1. Set the playback time to 16:00 UTC.
	2. Zoom into Zurich, Switzerland.
	3. Identify all aircraft within 40 km of the airport at the time. Exclude any aircraft on the ground in the airport area.
	4. Determine which two airborne aircraft are closest to each other.
	5. Report both callsigns. List the aircraft with the higher geographical altitude first.
	
	# RESULT FORMAT
	
	``json
	{
	"aircraft_1_callsign": "<callsign>",
	"aircraft_2_callsign": "<callsign>"
	}
	``
		{
		"aircraft_1_callsign": "AFR33SD",
		"aircraft_2_callsign": "SWR60L"
		}

No.	Task	Ground truth
		

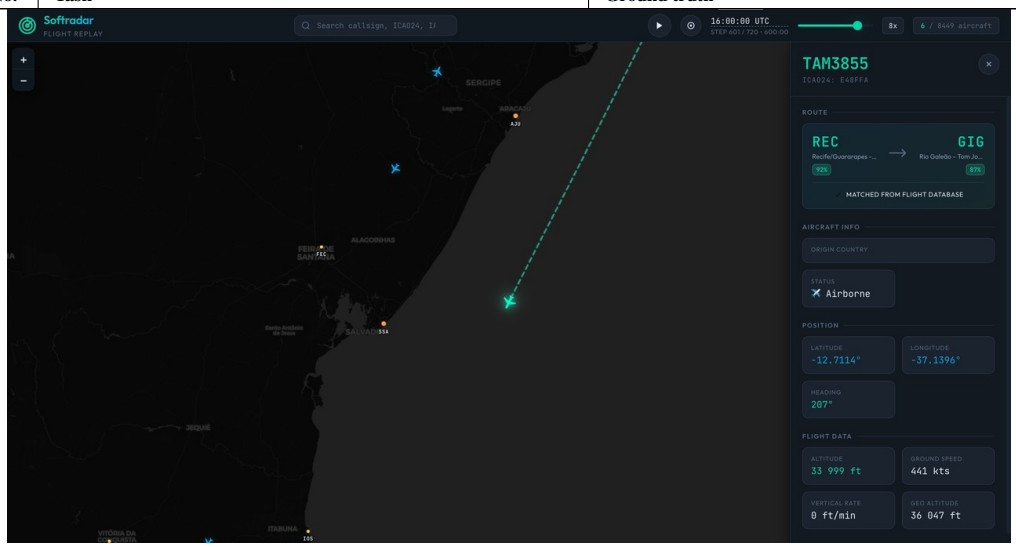
24	# Detect the First Landing and First Departure from an Airport
	
	## GOAL
	Monitor one airport over a short time window and identify the first aircraft that lands and the first aircraft that departs after the start time.
	
	## STEPS
	1. Set the playback time to 10:05 UTC.
	2. Find Vienna Airport (VIE).
	3. Observe aircraft activity for the next 10 minutes of playback time.
	4. Identify:
	- the first aircraft that departs from the airport after the start time.
	- the aircraft that is second to land at the airport after the start time.
	5. Report both callsigns and their UTC times.
	
	## RESULT FORMAT
	
	``json
	{
	"second_landing_callsign": "<callsign>",
	"first_departure_callsign": "<callsign>"
	}
	``
		{
		"second_landing_callsign": "AUA87",
		"first_departure_callsign": "AUA7F",
		}

No.	Task	Ground truth
		

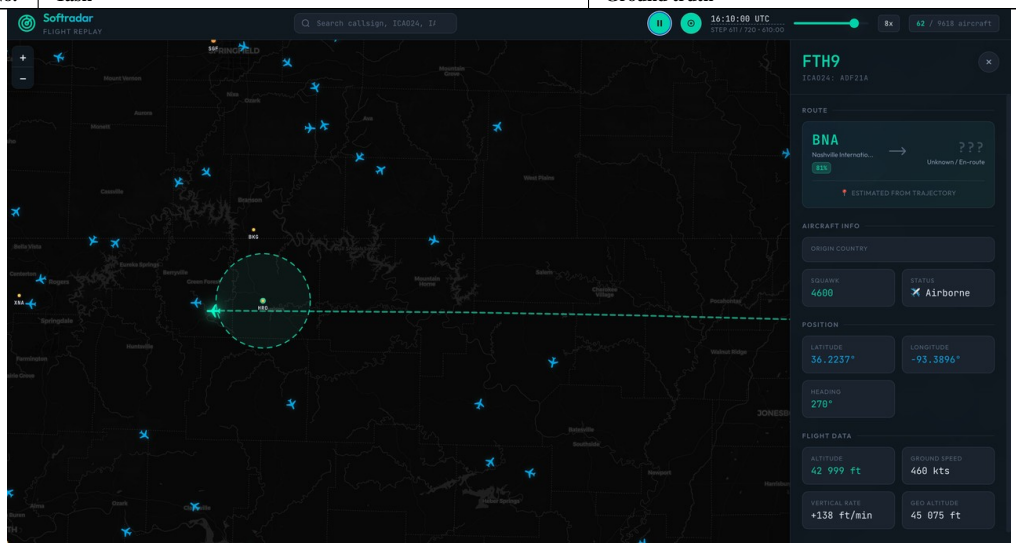
25	# Find the Previous Departure Before a Target Flight
	
	## GOAL
	Monitor departures from Manchester Airport within a defined radius and identify which flight departed immediately before a given target flight.
	
	## STEPS
	1. Set the playback time to 14:22 UTC.
	2. Search for Manchester Airport (MAN).
	3. Draw a 10 km radius circle centered on the airport.
	4. Monitor departures from the airport.
	5. Locate the departure of flight EZY14GQ.
	6. Determine which flight departed from Manchester Airport immediately before EZY14GQ.
	7. Report:
	- the previous flight callsign,
	- the UTC time when the previous flight left the radius circle.
	
	## RESULT FORMAT
	
	``json
	{
	"previous_callsign": "<callsign>",
	"previous_left_time": "<HH:MM UTC>"
	}
	``
		{
		"previous_callsign": "RUK6797",
		"previous_left_time": "14:25 UTC",
		}

No.	Task	Ground truth
		

26	# Identify the Last Remaining Flight Inside a Zone
	
	## GOAL
	Among several flights near an airport, determine which one is the last to remain inside a monitored radius.
	
	## STEPS
	1. Set the playback time to 16:21 UTC.
	2. Search for airport XCR and draw a 50 km radius circle around it.
	3. Locate flights SWR441R, UAE4PV, THY6FE, and S5BBC in the area.
	4. Track all four flights forward in time.
	5. Determine the moment when all of those flights left the circle.
	6. Report:
	- the UTC time when all 4 initial flights left the circle,
	- the callsign of the first new flight entering the circle,
	- the first UTC time when the circle contains 4 new flights.
	
	# RESULT FORMAT
	
	‘‘‘json
	{
	"initial_left_utc": "<HH:MM UTC>",
	"first_new_callsign": "<callsign>",
	"first_time_with_four_new_utc": "<HH:MM UTC>"
	}
	‘‘‘
		{
		"initial_left_utc": "16:25 UTC",
		"first_new_callsign": "EJU79XU",
		"first_time_with_four_new_utc":
		"16:28 UTC"
		}

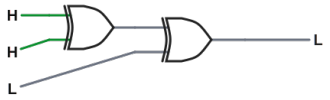
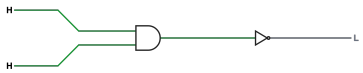
No.	Task	Ground truth
		

27	# Track a Two-Stage Zone Transition for Two Flights
	
	## GOAL
	Determine a multi-step temporal relationship between two flights around the same airport.
	
	## STEPS
	1. Set the playback time to 11:43 UTC.
	2. Search for the target airport, Ostend-Bruges International Airport (OST).
	3. Identify the two approaching flights that will be the first new flights to enter a 35 km radius zone around the airport.
	4. Find the first UTC time when exactly one of the two flights enters the 35 km zone around the airport.
	5. Continue tracking until both flights are outside the circle.
	6. Report:
	- the first time either of the two planes enters the zone,
	- which flight is the second to be inside the zone,
	- the UTC time when both flights finally exit the zone.
	
	# RESULT FORMAT
	
	‘‘‘json
	{
	"any_enters_utc": "<HH:MM UTC>",
	"last_inside_callsign": "<callsign>",
	"final_exit_time_utc": "<HH:MM UTC>"
	}
	‘‘‘
		{
		"any_enters_utc": "11:44 UTC",
		"last_inside_callsign": "RYR496K",
		"final_exit_time_utc": "11:50 UTC"
		}

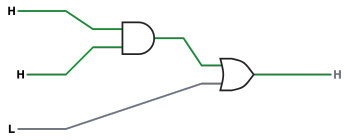
No.	Task	Ground truth
		

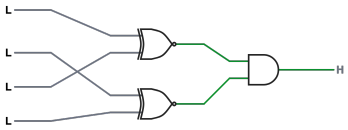
G.5 CIRCUIT DESIGNER

No.	Task	Ground truth
1	# Build a NOT Gate Using a NAND Gate ## GOAL Implement a logical NOT function using only a universal NAND gate. ## STEPS 1. Add the following components: - One input switch: A - One output switch: Y - Logical gates as needed 2. Wiring: - Make a NOT gate using only NAND gate. 3. Define output: - The output of the scheme must be the circuit output Y. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

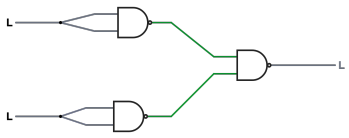
No.	Task	Ground truth
2	# Build a 3-Input Odd-Parity Circuit ## GOAL Construct a circuit with three binary inputs (A, B, C) and one output (Y) such that: - Y = 1 when an odd number of inputs are 1 - Y = 0 when an even number of inputs are 1 ## STEPS 1. Add the following components: - Three input switches: A, B, C - Two XOR gates - One output switch: Y 2. Wiring: - Connect A and B to the inputs of the first XOR gate. - Connect the output of the first XOR gate and C to the inputs of the second XOR gate. 3. Define output: - The output of the second XOR gate must be the circuit output Y. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	
3	# Build a NAND Gate From an AND Gate and a NOT Gate ## GOAL Implement the NAND function without using a NAND gate, i.e. Y = NOT (A AND B). ## STEPS 1. Add the following components: - Two input switches: A, B - One AND gate - One NOT gate - One output switch: Y 2. Wiring: - Connect A and B to the inputs of the AND gate. - Connect the output of the AND gate to the input of the NOT gate. 3. Define output: - The output of the NOT gate must be the circuit output Y. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

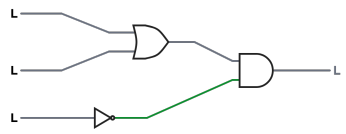
No.	Task	Ground truth

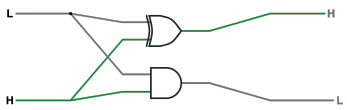
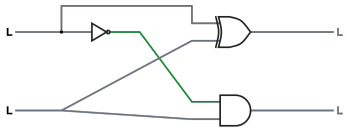
4	# Build an AND Gate With an Override Input
	
	## GOAL
	Construct a circuit with two data inputs (A, B) and one override input (C) such that Y = (A AND B) OR C - the output is high when both A and B are high, and is also forced high whenever C is high.
	
	## STEPS
	
	1. Add the following components:
	- Three input switches: A, B, C
	- One AND gate
	- One OR gate
	- One output switch: Y
	
	2. Wiring:
	- Connect A and B to the inputs of the AND gate.
	- Connect the output of the AND gate and C to the inputs of the OR gate.
	
	3. Define output:
	- The output of the OR gate must be the circuit output Y.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

5	# Build a 2-Bit Equality Comparator
	
	## GOAL
	Compare two 2-bit numbers A = (A1, A0) and B = (B1, B0) and drive one output EQ high exactly when the two numbers are equal.
	
	## STEPS
	
	1. Add the following components:
	- Four input switches: A1, A0, B1, B0
	- Two XNOR gates
	- One AND gate
	- One output switch: EQ
	
	2. Wiring:
	- Connect A1 and B1 to the inputs of the first XNOR gate.
	- Connect A0 and B0 to the inputs of the second XNOR gate.
	- Connect the outputs of both XNOR gates to the inputs of the AND gate.
	
	3. Define output:
	- The output of the AND gate must be the circuit output EQ.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

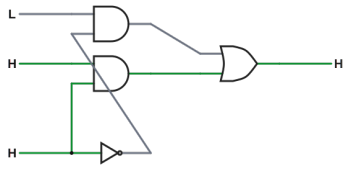
No.	Task	Ground truth

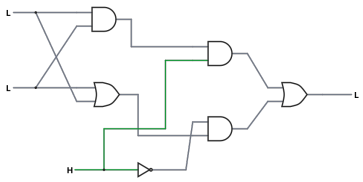
6	# Build an OR Gate Using Only NAND Gates
	
	## GOAL
	Implement the OR function using only universal NAND gates, i.e. Y = (NOT A) NAND (NOT B).
	
	## STEPS
	
	1. Add the following components:
	- Two input switches: A, B
	- Three NAND gates
	- One output switch: Y
	
	2. Wiring:
	- Use the first NAND gate as an inverter for A by connecting A to both of its inputs.
	- Use the second NAND gate as an inverter for B by connecting B to both of its inputs.
	- Connect the outputs of those two NAND gates to the inputs of the third NAND gate.
	
	3. Define output:
	- The output of the third NAND gate must be the circuit output Y.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

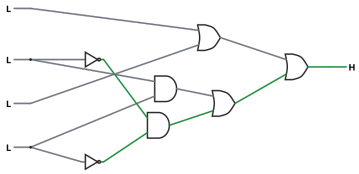
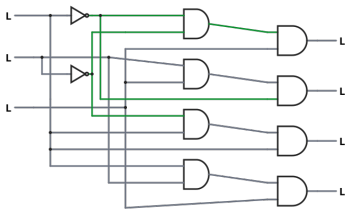
7	# Build an OR Gate With an Inhibit Input
	
	## GOAL
	Construct a circuit with two data inputs (A, B) and one inhibit input (C) such that Y = (A OR B) AND NOT C - the output follows "A or B", but is forced low whenever C is high.
	
	## STEPS
	
	1. Add the following components:
	- Three input switches: A, B, C
	- One OR gate
	- One NOT gate
	- One AND gate
	- One output switch: Y
	
	2. Wiring:
	- Connect A and B to the inputs of the OR gate.
	- Connect C to the input of the NOT gate.
	- Connect the output of the OR gate and the output of the NOT gate to the inputs of the AND gate.
	
	3. Define output:
	- The output of the AND gate must be the circuit output Y.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

No.	Task	Ground truth
8	# Build a Half Adder ## GOAL Construct a 1-bit half adder with two inputs (A, B) and two outputs: SUM = A XOR B CARRY = A AND B ## STEPS 1. Add the following components: - Two input switches: A, B - One XOR gate - One AND gate - Two output switches: SUM and CARRY 2. Wiring: - Connect A and B to the inputs of the XOR gate. - Connect A and B to the inputs of the AND gate. 3. Define outputs: - The output of the XOR gate must be the circuit output SUM. - The output of the AND gate must be the circuit output CARRY. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	
9	# Build a Half Subtractor ## GOAL Construct a 1-bit half subtractor with two inputs (A, B) that computes A - B and produces two outputs: DIFF = A XOR B BORROW = (NOT A) AND B ## STEPS 1. Add the following components: - Two input switches: A, B - One XOR gate - One NOT gate - One AND gate - Two output switches: DIFF and BORROW 2. Wiring: - Connect A and B to the inputs of the XOR gate. - Connect A to the input of the NOT gate. - Connect the output of the NOT gate and B to the inputs of the AND gate. 3. Define outputs: - The output of the XOR gate must be the circuit output DIFF. - The output of the AND gate must be the circuit output BORROW. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

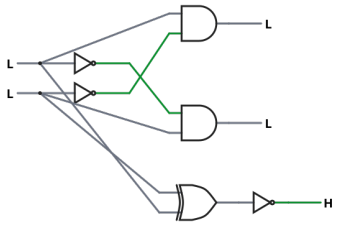
No.	Task	Ground truth

10	# Build a Full-Adder Using Logic Gates
	
	## GOAL
	Construct a circuit that computes the 1-bit full-adder functions:
	
	SUM = A XOR B XOR CIN
	COUT = (A AND B) OR (A AND CIN) OR (B AND CIN)
	
	## STEPS
	
	1. Add the following components:
	- Three input switches: A, B, CIN
	- Two output switches: SUM and COUT
	- Logical gates as needed
	
	2. Build the circuit so that:
	- SUM is computed as the XOR of A, B, and CIN
	- COUT is computed as the carry bit of the full adder
	
	3. Define outputs:
	- The output of the second XOR gate must be the circuit output SUM.
	- The output of the second OR gate must be the circuit output COUT.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

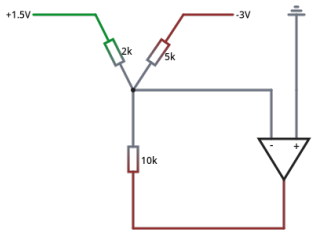
11	# Build a 1-Bit ALU With 2 Operations
	
	## GOAL
	Construct a simple 1-bit ALU that takes two data inputs A, B and one operation-select bit OP, then produces one output RESULT.
	
	The ALU must support two operations OR and AND
	
	## STEPS
	
	1. Add three input switches: A, B and OP
	
	2. Add one output switch: RESULT
	
	3. Build both candidate operation wires:
	- W_AND = A AND B
	- W_OR = A OR B
	
	4. Use OP to select which operation appears at RESULT:
	- When OP = 0, RESULT = W_AND
	- When OP = 1, RESULT = W_OR
	
	5. Verification:
	- For OP = 0, the circuit must behave like an AND gate.
	- For OP = 1, the circuit must behave like an OR gate.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

No.	Task	Ground truth
12	# Build a BCD-to-7-Segment Decoder for Segment a ## GOAL Implement a circuit that takes a BCD input D = (D3, D2, D1, D0) and drives output SEG_A high when segment a of a 7-segment display must be lit. Use the following Boolean expression: SEG_A = D3 + D1 + (D2 AND D0) + (NOT D2 AND NOT D0) ## STEPS 1. Add the following components: - Four input switches: D3, D2, D1, D0 - One output switch: SEG_A - Logical gates as needed 2. Build the circuit: - Treat D3 as the most significant bit. - Implement ‘SEG_A‘ exactly as given in the goal equation. - Use NOT gates where negated signals are required. - Combine the terms using the necessary AND and OR gates. 3. Define the output: - The single circuit output must be SEG_A. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	
13	# Build a 1-to-4 Demultiplexer ## GOAL Implement a circuit with one data input D, two select bits S1 and S0, and four outputs Y0..Y3, such that D is routed to exactly one output chosen by (S1, S0) while the others stay low: Y0 = (NOT S1 AND NOT S0 AND D), Y1 = (NOT S1 AND S0 AND D), Y2 = (S1 AND NOT S0 AND D), Y3 = (S1 AND S0 AND D) ## STEPS 1. Add the following components: - One data input switch: D - Two select input switches: S1, S0 - Four output switches: Y0, Y1, Y2, Y3 - Logical gates as needed 2. Build the circuit so that: - For each Yi, build the 3-input AND of D and the select pattern for i. - Exactly one Yi can be 1 at any time, and only when D = 1. 3. Define outputs: - The four circuit outputs must be Y0, Y1, Y2, Y3. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

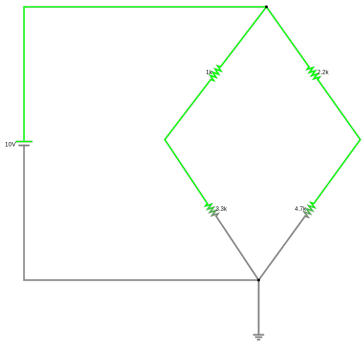
No.	Task	Ground truth

14	# Build a 1-Bit Comparator with Three Outputs
	
	## GOAL
	Construct a circuit that compares two 1-bit inputs A and B and produces three outputs:
	
	A_GT_B = 1 when A > B
	A_EQ_B = 1 when A = B
	A_LT_B = 1 when A < B
	
	## STEPS
	
	1. Add the following components:
	- Two input switches: A and B
	- Two NOT gates
	- Two AND gates
	- One XNOR gate
	- Three output switches: A_GT_B, A_EQ_B, and A_LT_B
	
	2. Wiring:
	- Connect input B to the first NOT gate.
	- Connect input A and the output of the first NOT gate to the inputs of the first AND gate.
	- Connect input A to the second NOT gate.
	- Connect the output of the second NOT gate and input B to the inputs of the second AND gate.
	- Connect inputs A and B to the inputs of the XNOR gate.
	
	3. Define outputs:
	- The output of the first AND gate must be the circuit output A_GT_B.
	- The output of the second AND gate must be the circuit output A_LT_B.
	- The output of the XNOR gate must be the circuit output A_EQ_B.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

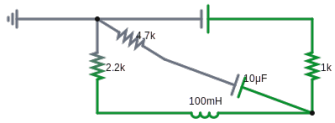
No.	Task	Ground truth

15	# Inverting Summing Amplifier
	
	## GOAL
	Construct an inverting summing amplifier using an ideal op-amp and measure its output voltage.
	
	## STEPS
	
	1. Add and configure components:
	- Add an Ideal Op-Amp.
	- Add a 1-terminal DC Voltage Source (V1) and set it to 1.5 V.
	- Add a Resistor (R1) and set its resistance to 2 k ohms.
	- Add a 1-terminal DC Voltage Source (V2) and set it to -3.0 V.
	- Add a Resistor (R2) and set its resistance to 5 k ohms.
	- Add a Feedback Resistor (Rf) and set its resistance to 10 k ohms.
	- Add a Ground connection.
	
	2. Wiring:
	- Connect the non-inverting (+) input of the Op-Amp directly to Ground.
	- Connect V1 to one end of R1.
	- Connect V2 to one end of R2.
	- Connect the free ends of both R1 and R2 to the inverting (-) input of the Op-Amp.
	- Connect Rf between the Op-Amp's output terminal and its inverting (-) input terminal.
	
	3. Once the circuit reaches a steady state, hover your mouse over the Op-Amp and report the exact Output Voltage (Vout) displayed in the information box at the bottom right corner of the screen.
	
	# RESULT FORMAT
	
	‘‘‘json
	{
	"opamp_output_voltage": "value"
	}
	‘‘‘
		
		
		{
		"opamp_output_voltage": "-1.5 V"
		}

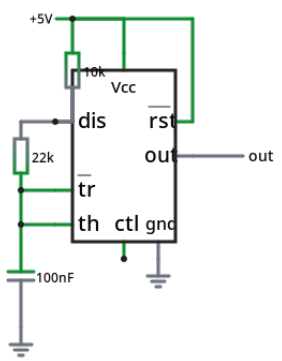
No.	Task	Ground truth

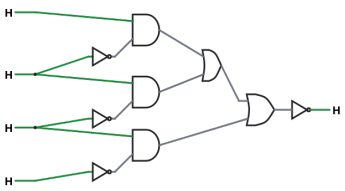
16	# Wheatstone Bridge Voltage Measurement
	
	## GOAL
	Construct a Wheatstone bridge circuit and measure the voltage drop across one of the bridge resistors.
	
	## STEPS
	
	1. Add and configure components:
	- Add a 2-terminal DC Voltage Source and set its voltage to 10 V.
	- Add a Resistor (R1) and set its resistance to 1 k ohms.
	- Add a Resistor (R2) and set its resistance to 2.2 k ohms.
	- Add a Resistor (R3) and set its resistance to 3.3 k ohms.
	- Add a Resistor (R4) and set its resistance to 4.7 k ohms.
	- Add a Ground connection.
	
	2. Wiring:
	- Ground the negative terminal of the voltage source.
	- Connect one end of R1 to the positive terminal of the voltage source.
	- Connect one end of R2 to the positive terminal of the voltage source.
	- Connect the free end of R1 to one end of R3.
	- Connect the free end of R2 to one end of R4.
	- Connect the free ends of R3 and R4 together and to Ground.
	
	3. Once the circuit reaches a steady state, hover your mouse over Resistor R4 (the 4.7 k ohm resistor) and report the exact Voltage Drop (Vd) displayed in the information box at the bottom right corner of the screen.
	
	# RESULT FORMAT
	
	``json
	{
	"voltage_drop_R4": "value"
	}
	``
		
		
		{
		"voltage_drop_R4": "6.812 V"
		}

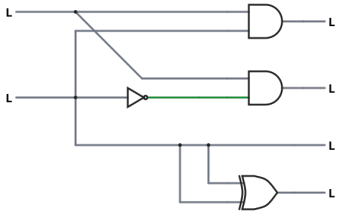
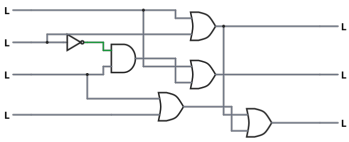
No.	Task	Ground truth

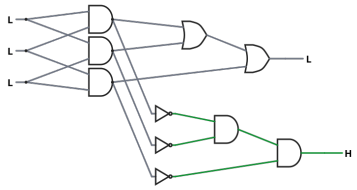
17	# RLC DC Steady-State Trick
	
	## GOAL
	Construct an RLC circuit driven by DC and measure the capacitor voltage at steady state.
	
	## STEPS
	
	1. Add and configure components:
	- Add a 2-terminal DC Voltage Source and set its voltage to 24 V.
	- Add a Resistor (R1) and set its resistance to 1 k ohms.
	- Add an Inductor and set its inductance to 100 mH.
	- Add a Resistor (R2) and set its resistance to 2.2 k ohms.
	- Add a Capacitor and set its capacitance to 10 uF.
	- Add a Resistor (R3) and set its resistance to 4.7 k ohms.
	- Add a Ground connection.
	
	2. Wiring:
	- Ground the negative terminal of the voltage source.
	- Connect one end of R1 to the positive terminal of the source. Call the other end of R1 "Node A".
	- From Node A, create a path to Ground containing the Inductor in series with R2.
	- From Node A, create a second parallel path to Ground containing the Capacitor in series with R3.
	
	3. Hover your mouse over the Capacitor. Wait for the simulation to charge the capacitor and reach a complete DC steady state (capacitor current displayed in the information box is less than 20 nA) and report the exact Voltage Drop (Vd) displayed in the information box at the bottom right corner of the screen.
	
	# RESULT FORMAT
	
	``json
	{
	"capacitor_voltage_drop": "value"
	}
	``
		
		{
		"capacitor_voltage_drop": "16.5 V"
		}

No.	Task	Ground truth

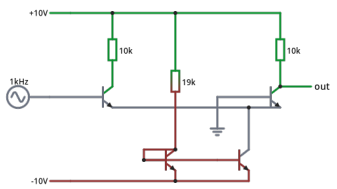
18	# 555 Timer Astable: Output Frequency
	
	## GOAL
	Configure a 555 timer in astable (free-running) mode and measure the exact output frequency and duty cycle on the oscilloscope.
	
	## STEPS
	
	1. Add and configure components:
	- Add a 555 Timer chip.
	- Add a Voltage Source 1-terminal (Vcc) and set it to 5 V.
	- Add a Resistor (Ra) and set its resistance to 10 k ohms.
	- Add a Resistor (Rb) and set its resistance to 22 k ohms.
	- Add a Capacitor (Ct) and set its capacitance to 100 nF.
	- Add a Ground connection.
	
	2. Wiring:
	- Connect Vcc to Pin 8 (V+) of the 555.
	- Connect Pin 1 (GND) of the 555 to Ground.
	- Connect Pin 4 (RESET) of the 555 directly to Vcc (keep reset inactive).
	- Connect Ra between Vcc and Pin 7 (DISCH).
	- Connect Rb between Pin 7 (DISCH) and Pin 6 (THRESH).
	- Connect Pin 6 (THRESH) directly to Pin 2 (TRIG).
	- Connect Ct between Pin 6 (THRESH) and Ground.
	
	3. Right-click Pin 3 (OUT) of the 555 and select "View in New Scope". In the scope's properties, enable BOTH "Show Frequency" AND "Show Duty Cycle". Let the simulation run for at least 30 output cycles to let the measurements settle, then report the exact Frequency and Duty Cycle values displayed on the scope.
	
	# RESULT FORMAT
	
	``json
	{
	"output_frequency": "value",
	"duty_cycle": "value"
	}
	``
		
		{
		"output_frequency": "260.417 Hz",
		"duty_cycle": "58%"
		}

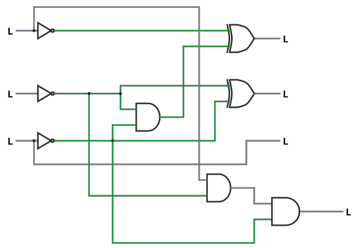
19	# Build a 4-bit Monotonic Sequence Detector
	
	## GOAL
	Construct a combinational circuit with four binary inputs A, B, C, D (read left-to-right as a sequence in that order) and one output Y, such that Y = 1 if and only if the bit sequence A, B, C, D is non-decreasing - i.e., a 1 is never followed by a 0 anywhere in the sequence.
	
	## STEPS
	
	1. Add the following components:
	- Four input switches: A, B, C, D
	- One output switch: Y
	- AND, OR, NOT, NAND, NOR, XOR, XNOR gates as needed
	
	2. Build the circuit so that:
	- Y must produce the correct value for all 16 possible input combinations.
	
	# RESULT FORMAT
	
	``json
	{"answer": "done"}
	``
		

No.	Task	Ground truth
20	# Build a 2-bit Binary Squarer ## GOAL Construct a combinational circuit with two binary inputs A1, A0 (A1 the most significant bit) representing the unsigned integer N = 2·A1 + A0, and four binary outputs P3, P2, P1, P0 (P3 the most significant bit) representing the unsigned binary value of N^2, such that for every one of the 4 input combinations: N^2 = 8·P3 + 4·P2 + 2·P1 + P0 ## STEPS 1. Add the following components: - Two input switches: A1, A0 - Four output switches: P3, P2, P1, P0 - AND, OR, NOT, NAND, NOR, XOR, XNOR gates as needed 2. The circuit must produce correct values for all 4 input combinations. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	
21	# Build a 4-to-2 Priority Encoder with Valid Flag ## GOAL Construct a combinational circuit with four request inputs R3, R2, R1, R0 (R3 is the highest priority) and three outputs Y1, Y0, V such that: - V = 1 if at least one request is high; V = 0 otherwise. - When V = 1, (Y1, Y0) is the binary index of the highest-priority asserted input: R3 high → (Y1, Y0) = (1, 1) R3 low, R2 high → (1, 0) R3 low, R2 low, R1 high → (0, 1) only R0 high → (0, 0) - When V = 0, (Y1, Y0) is don’t-care. Implement the outputs using these expressions (priority is enforced by masking lower-priority requests with NOT of higher-priority ones): Y1 = R3 OR (NOT R3 AND R2) = R3 OR R2 Y0 = R3 OR (NOT R2 AND R1) V = R3 OR R2 OR R1 OR R0 Outputs must be correct for all 16 input combinations. ## STEPS 1. Add the following components: - Four input switches: R3, R2, R1, R0 - Three output switches: Y1, Y0, V - AND, OR, NOT, NAND, NOR, XOR, XNOR gates as needed 2. Build Y1, Y0, V exactly as given above. 3. Define outputs: - The three circuit outputs must be Y1, Y0, V. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

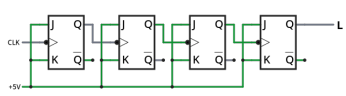
No.	Task	Ground truth
22	# Build a 3-Input Majority and Minority Detector ## GOAL Construct a combinational circuit with three binary inputs A, B, C and two outputs MAJ and MIN such that: - MAJ = 1 when at least two of the three inputs are 1 - MIN = 1 when at least two of the three inputs are 0 Exactly one of MAJ and MIN is high for every input combination, and they are logical complements. Implement the outputs using these expressions, sharing the three pairwise-AND terms across both outputs: T_AB = A AND B T_AC = A AND C T_BC = B AND C MAJ = T_AB OR T_AC OR T_BC MIN = (NOT T_AB) AND (NOT T_AC) AND (NOT T_BC) The three pairwise-AND wires (T_AB, T_AC, T_BC) must each be computed by a single AND gate and fan out to both the MAJ OR-tree and the MIN NOT/AND-tree. ## STEPS 1. Add the following components: - Three input switches: A, B, C - Two output switches: MAJ, MIN - AND, OR, NOT, NAND, NOR, XOR, XNOR gates as needed 2. Build T_AB, T_AC, T_BC exactly as given. Then build MAJ and MIN by reusing those three wires. 3. Define outputs: - The two circuit outputs must be MAJ and MIN. - Outputs must be correct for all 8 input combinations. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

No.	Task	Ground truth

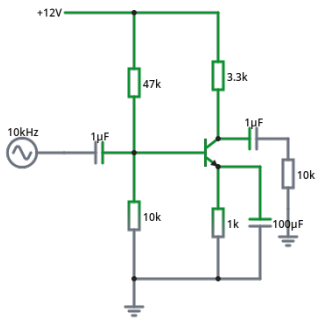
23	# BJT Differential Pair with Resistor Loads
	
	## GOAL
	Build a BJT differential pair with resistor collector loads and an NPN current-mirror tail. Apply a small differential input and measure the peak-to-peak output voltage at the collector of the right-side transistor.
	
	## STEPS
	
	1. Add and configure components:
	- Add three supply nodes: Vcc = +10 V, Vee = -10 V, and Ground.
	- Add two default NPN transistors for the input pair: Q1 and Q2.
	- Add two collector-load resistors:
	- Rc1 = 10 k ohms
	- Rc2 = 10 k ohms
	- Add two default NPN transistors for the tail current mirror: Q5 and Q6.
	- Add a reference resistor Rref and set its resistance to 19 k ohms.
	- Add a 1-terminal A/C Voltage Source (Vin) and set: Max Voltage = 1 mV, Frequency = 1000 Hz, DC offset = 0 V, phase = 0 deg.
	
	2. Wiring:
	- Connect the emitters of Q1 and Q2 together. Call this node TAIL.
	- Connect Rc1 from Vcc to the collector of Q1.
	- Connect Rc2 from Vcc to the collector of Q2. Call the collector of Q2 OUT.
	- Connect the emitters of Q5 and Q6 to Vee.
	- Diode-connect Q5 by shorting its collector to its base.
	- Connect Rref from Vcc to the collector/base node of Q5.
	- Connect the base of Q6 to the base of Q5.
	- Connect the collector of Q6 to the TAIL node.
	- Connect Vin to the base of Q1.
	- Connect the base of Q2 directly to Ground.
	
	3. Measurement:
	- Run the simulation.
	- Confirm that the DC operating point places OUT between the rails, approximately around +5 V DC.
	- Scope the OUT node.
	- In steady state, read the Peak-to-Peak value shown in the scope overlay.
	- Report the measured peak-to-peak AC voltage at OUT.
	
	# RESULT FORMAT
	
	‘‘‘json
	{
	"diff_pair_output_peak_to_peak": "value"
	}
		
		{
		"diff_pair_output_peak_to_peak": "191.615 mV"
		}

No.	Task	Ground truth
24	# Build a 3-bit Two's-Complement Negator with Overflow Flag ## GOAL Construct a combinational circuit that takes a 3-bit two's-complement signed integer A = (A2, A1, A0), where A2 is the sign bit and the represented value is val(A) = -4·A2 + 2·A1 + A0 (range: -4 .. +3) and produces: - A 3-bit two's-complement output Y = (Y2, Y1, Y0) such that val(Y) = -val(A) whenever -val(A) is representable in 3-bit two's complement. - A 1-bit overflow flag OVF that is 1 if and only if A = 100 (i.e. val(A) = -4), because +4 cannot be represented in 3-bit two's complement. When OVF = 1, the value of Y is unconstrained (don't-care). The negation must be computed as Y = (NOT A) + 1 using bitwise inversion followed by a 3-bit ripple-carry incrementer. ## STEPS 1. Add the following components: - Three input switches: A2, A1, A0 - Four output switches: Y2, Y1, Y0, OVF - AND, OR, NOT, NAND, NOR, XOR, XNOR gates as needed 2. Build the circuit so that: - First stage: compute B0 = NOT A0, B1 = NOT A1, B2 = NOT A2 using NOT gates. - Second stage: ripple-carry increment B by 1 using three half-adders chained by carry: - Bit 0: Y0 = B0 XOR 1 = NOT B0 (equivalently, Y0 = A0); carry C1 = B0 AND 1 = B0. - Bit 1: Y1 = B1 XOR C1; carry C2 = B1 AND C1. - Bit 2: Y2 = B2 XOR C2. (The final carry-out is discarded - it does not feed OVF.) - Overflow: OVF = A2 AND (NOT A1) AND (NOT A0). This is the only input pattern for which the negation overflows. - Y must produce the correct value for all 7 non-overflow input combinations, and OVF must be correct for all 8 input combinations. 3. Define outputs: - The four circuit outputs must be Y2, Y1, Y0, OVF. # RESULT FORMAT ‘‘‘json {"answer": "done"} ‘‘‘	

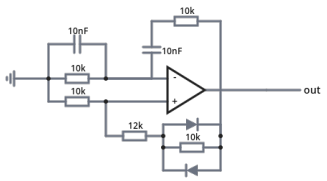
No.	Task	Ground truth

25	# 4-Bit Asynchronous Ripple Counter: MSB Output Frequency
	
	## GOAL
	Build a 4-bit asynchronous (ripple) counter from JK flip-flops driven by a 1 kHz clock, and measure the output frequency at the MSB (Q3). Each flip-flop divides its clock input by 2, so a 4-bit ripple counter divides the master clock by 2^4 = 16.
	
	## STEPS
	
	1. Add and configure components:
	- Add a Clock source (Inputs/Outputs menu → Clock, or use a Square-wave Voltage Source). Set: frequency = 1000 Hz, duty cycle = 50%, high level = 5 V, low level = 0 V.
	- Add four JK Flip-Flops (Chips menu → JK Flip-Flop). Label them FF0 (LSB) through FF3 (MSB).
	- Add a Logic Input set permanently to high (5 V) - used as a constant logic-1 source for the J and K pins.
	- Add a Logic Output for visualising Q3.
	- Add Ground.
	
	2. Wiring (asynchronous toggle counter):
	- Tie BOTH J and K of every flip-flop (FF0-FF3) to the constant-high Logic Input - this configures each one to toggle on every clock edge.
	- Connect the Clock source to the CLK input of FF0.
	- Connect Q (output) of FF0 → CLK of FF1.
	- Connect Q of FF1 → CLK of FF2.
	- Connect Q of FF2 → CLK of FF3.
	- Connect Q of FF3 → the Logic Output indicator.
	- Leave the !Q (inverted) outputs unconnected, and leave any RESET/SET pins unconnected (or tie them to the inactive level if the chip block exposes them).
	
	3. Click RUN.
	
	4. Right-click the wire at Q3 (the output of FF3) → View in New Scope. In the scope's properties, enable "Show Frequency". Let the simulation run for at least 1 second of simulated time so the counter cycles through all 16 states many times and the frequency overlay locks in.
	
	5. Report the frequency value displayed on the scope at Q3.
	
	# RESULT FORMAT
	
	The value must be a string containing the numeric frequency in Hz with units, e.g. "62.5 Hz". Do NOT emit the literal placeholder text "value". Do NOT omit units. Round to 1 decimal place if needed.
	
	‘‘‘json
	{
	"msb_output_frequency": "<frequency>"
	}
	‘‘‘
		
		{
		"msb_output_frequency": "62.5 Hz"
		}

No.	Task	Ground truth

26	# Common-Emitter BJT Amplifier: Mid-Band Peak-to-Peak Output
	
	## GOAL
	Build a classic common-emitter BJT voltage amplifier with a voltage-divider bias network and a bypassed emitter resistor, and measure the peak-to-peak output voltage at the collector for a small AC input.
	
	## STEPS
	
	1. Add and configure components:
	- Add a Voltage Source 1-terminal (Vcc) and set it to 12 V.
	- Add a Transistor (bipolar, NPN). Leave default model parameters.
	- Add a Resistor (R1) and set it to 47 k ohms (top bias).
	- Add a Resistor (R2) and set it to 10 k ohms (bottom bias).
	- Add a Resistor (Rc) and set it to 3.3 k ohms (collector).
	- Add a Resistor (Re) and set it to 1 k ohms (emitter).
	- Add a Capacitor (Cin) and set it to 1 uF (input coupling).
	- Add a Capacitor (Ce) and set it to 100 uF (emitter bypass).
	- Add a Capacitor (Cout) and set it to 1 uF (output coupling).
	- Add a Resistor (R_load) and set it to 10 k ohms.
	- Add a 1-terminal A/C Voltage Source (Vin) and set: Max Voltage = 10 mV, Frequency = 10 kHz, DC offset = 0, phase = 0.
	- Add a Ground connection.
	
	2. Wiring:
	- Connect Vcc to the top of R1 and to the top of Rc.
	- Connect the bottom of R1 and the top of R2 together - call this node "Base_Bias". Connect the transistor's base to Base_Bias.
	- Connect the bottom of R2 to Ground.
	- Connect the bottom of Rc to the transistor's collector. Call this node "Collector".
	- Connect the transistor's emitter to the top of Re. Call this node "Emitter".
	- Connect the bottom of Re to Ground.
	- Connect Ce between Emitter and Ground (parallel with Re).
	- Connect Cin between Vin and Base_Bias.
	- Connect Cout between Collector and one end of R_load. Call the other end of R_load "Node OUT".
	- Connect Node OUT to Ground through R_load (i.e., R_load is between Cout and Ground, and Node OUT is the junction with Cout).
	
	3. Right-click the wire at Node OUT (the load side of Cout) and select "View in New Scope". In the scope's properties, enable "Show Peak-to-Peak". Let the simulation run for at least 50 input cycles so all coupling capacitors fully charge and the amplifier settles into its operating point, then report the exact Peak-to-Peak value displayed on the scope.
	
	# RESULT FORMAT
	
	``json
	{
	"amplifier_output_peak_to_peak": "value"
	}
	``
		
		{
		"amplifier_output_peak_to_peak": "2.605 V"
		}

No.	Task	Ground truth

27	# Wien Bridge Oscillator with Diode AGC
	
	## GOAL
	Build a Wien-bridge oscillator using an op-amp with positive-feedback RC network and a negative-feedback gain network. Use anti-parallel diodes for soft amplitude limiting (automatic gain control), let the loop self-start from a small initial-condition seed, and report both the steady-state oscillation frequency and the peak-to-peak output voltage at the op-amp output.
	
	## STEPS
	
	1. Add and configure components:
	- Add an Op Amp (use the default model, finite-GBW, rails at ±15 V).
	- Positive-feedback (Wien) network:
	- Resistor Rs = 10 k ohms (series arm).
	- Capacitor Cs = 10 nF (series arm).
	- Resistor Rp = 10 k ohms (parallel arm).
	- Capacitor Cp = 10 nF (parallel arm).
	- Negative-feedback (gain-setting) network with diode AGC:
	- Resistor Rg = 10 k ohms (from — input to ground).
	- Resistor Rf_a = 12 k ohms (first half of the feedback resistor).
	- Resistor Rf_b = 10 k ohms (second half of the feedback resistor; the diodes shunt this part as the swing grows).
	- Two anti-parallel diodes (default Si model), each connected across Rf_b.
	- Add Ground connections.
	
	2. Wiring:
	- Positive-feedback network (between op-amp OUTPUT and the + input):
	- Rs from op-amp OUTPUT to a node "Node_S".
	- Cs from Node_S to the op-amp + input (call this node "Node_Plus").
	- Rp from Node_Plus to Ground.
	- Cp from Node_Plus to Ground (in parallel with Rp).
	- Negative-feedback network:
	- Rg from the op-amp — input to Ground.
	- Rf_a from the op-amp — input to a midpoint node "Node_FB_Mid".
	- Rf_b from Node_FB_Mid to op-amp OUTPUT.
	- Diode D1 from Node_FB_Mid (anode) to op-amp OUTPUT (cathode) - i.e., across Rf_b.
	- Diode D2 from op-amp OUTPUT (anode) to Node_FB_Mid (cathode) - i.e., the anti-parallel partner of D1, also across Rf_b.
	
	3. Seed start-up:
	- Right-click Cp (the cap in the parallel arm) → Edit → set Initial Voltage = 0.1 V (or higher if needed). CircuitJS1's components are noiseless, so without a seed the oscillator will not spontaneously start.
	4. Run the simulation.
	5. Right-click the op-amp OUTPUT terminal node and select "View in New Scope". In the scope's properties, enable BOTH "Show Frequency" AND "Show Peak-to-Peak". Let the simulation run long enough for the amplitude to grow into the diode-clamped steady state (~50-100 ms simulated). The frequency overlay needs at least 5-10 cycles visible per screen to lock in - if it reads blank, zoom the scope's TIME axis (not the voltage axis) until cycles are clearly resolved, then wait for the readout to stabilise.
	
	6. Report the exact Frequency and Peak-to-Peak values displayed on the scope.
	
	# RESULT FORMAT
	
	‘‘‘json
	{ "oscillation_frequency": "value",
	"output_peak_to_peak": "value" }
	‘‘‘
		
		{
		"oscillation_frequency": "1.589 kHz",
		"output_peak_to_peak": "1.422 V"
		}

H Extended Related Work

Web Agent Benchmarks A growing number of work has focused on evaluating autonomous agents in web-based environments. Early benchmarks such as MiniWoB (Shi et al., 2017) and MiniWoB++ (Liu et al., 2018) introduced simplified web interaction tasks where agents had to perform basic operations like clicking buttons, filling forms, and navigating menus. While these benchmarks were instrumental in establishing the field, they relied on synthetic, toy-like interfaces that did not capture the complexity of real websites. More recent efforts are aiming to close this gap. WebArena (Zhou et al., 2024) proposed a suite of realistic, self-hosted web environments spanning e-commerce, forums, and content management systems, enabling agents to be evaluated on tasks that more closely resemble real-world usage. VisualWebArena (Koh et al., 2024) extended this idea by emphasizing visually grounded tasks that require agents to reason over rendered page content rather than relying solely on HTML structure. WebVoyager (He et al., 2024) took a different approach by deploying agents on live websites, which introduces non-determinism but better reflects practical deployment conditions. Mind2Web (Deng et al., 2023) contributed a large-scale dataset of real-world web tasks annotated with action sequences across diverse domains. REAL (Garg et al., 2025) proposed deterministic simulations of real websites to combine realism with reproducibility. AssistantBench (Yoran et al., 2024) evaluated agents on open-ended web tasks requiring information synthesis. Despite this progress, most existing web benchmarks focus on consumer-level scenarios such as online shopping or information lookup, and they primarily test basic UI navigation and form-filling skills. Our work differs in that we target professional-level tasks that require higher-order capabilities like dynamic perception, graphical understanding, and 3D spatial reasoning, which remain largely unexplored in prior benchmarks.

OS Agent Benchmarks A neighbouring line of work looks at agents that operate at the operating system level rather than inside a web browser. While this is not directly related to our work, the challenges are similar in many ways, and progress in one area often signals progress in the other. OSWorld (Xie et al., 2024) sets up a benchmark where agents control full desktop environments (Ubuntu, Windows, macOS) through keyboard and mouse actions across many different applications. AndroidWorld (Rawles et al., 2025) and AndroidEnv (Toyama et al., 2021) focus on mobile systems, asking agents to navigate apps and complete tasks on Android. WindowsAgentArena (Bonatti et al., 2024) targets productivity tasks inside the Windows operating system. CRAB (Xu et al., 2025) further broadens this direction by evaluating multimodal agents across desktop and mobile environments with graph-based fine-grained evaluation. OS-level benchmarks cover a very wide range of agent skills, but they can be harder to reproduce and control than web environments, since desktop state depends on many moving parts. Our benchmark goes in a different direction: we stay inside the browser, where we get cleaner control and reproducibility, while still picking tasks that require reasoning well beyond simple point-and-click interactions.

Domain-Specific Multimodal Benchmarks Several benchmarks probe the perceptual competences we target, but do so by querying a model rather than by having an agent act. MVBench (Li et al., 2024) and VEU-Bench (Li et al., 2025a) evaluate temporal and editing understanding through questions over pre-recorded videos; FlowVQA (Singh et al., 2024) asks questions about static flowchart images; MapEval (Dihan et al., 2025) poses geospatial questions over map text, APIs, and snapshots; and MMCircuitEval (Zhao et al., 2025) tests circuit knowledge through question–answer pairs. Their consistently low scores confirm that these capabilities are hard for current MLLMs, but they measure perception in isolation: the model never has to translate what it perceives into a sequence of grounded actions on a stateful interface, verify the outcome, and recover from its own mistakes. Closer to the agentic setting, ScreenSpot-Pro (Li et al., 2025b) evaluates single-step element grounding on high-resolution professional screenshots, BlenderGym (Gu et al., 2025) evaluates 3D editing through Blender's Python API rather than its GUI, and Spider2-V (Cao et al., 2024) evaluates full workflows in professional data tooling, where GUI interaction centres on forms, spreadsheets, and IDEs, and pipelines in its orchestration tools are defined in code rather than on a visual canvas. GauntletBench sits in the gap between these lines: it embeds temporal, graphical, and 3D perception inside closed-loop, multi-step GUI interaction, where a single mis-grounded action compounds over a long horizon (Appendix D). We therefore regard these benchmarks as complementary evidence of the same underlying deficit, and state our claim precisely: the targeted capabilities are underexplored not in multimodal understanding benchmarks at large, but in agentic benchmarks in particular.

Agentic Evaluation A key challenge in evaluating autonomous agents is designing evaluation pipelines that are both fair and informative. Traditional approaches rely on exact-match comparisons

between agent outputs and predefined ground truths (Drouin et al., 2024), which works reasonably well for simple retrieval tasks but becomes problematic for complex, open-ended scenarios where multiple valid solutions may exist. REAL (Garg et al., 2025) addressed this by introducing deterministic website simulations paired with step-wise trajectory tracing, implementing reproducible evaluation of agents through screenshots and action logs. Their framework decouples agent implementation from environment execution, which we adopt and extend in our work to support evaluation of both open-source models and closed-source agent frameworks through a unified interface. More recently, the use of LLMs as automated judges has gained popularity as a way to assess partial task completion (Chiang et al., 2024). Instead of only checking whether the final output is exactly correct, an LLM judge can inspect the agent's trajectory and the final visible state to assign a graded score that reflects how much progress was actually made. This is especially useful for tasks where an agent might get most of the way to the goal but fail on a small detail. However, LLM-based judges can be inconsistent and may hallucinate progress that did not actually occur (Zheng et al., 2023), so careful prompt design and calibration are needed. Some works have also explored hybrid evaluation strategies that combine rule-based checks with model-based assessment (Koh et al., 2024), taking the best of both worlds. In our benchmark, we implement a two-stage evaluation pipeline: a structured exact-match evaluator that checks domain-specific constraints, and an LLM-based judge that provides partial success scores based on visual trajectory evidence. We additionally report efficiency-oriented metrics including token consumption, latency, and number of interaction steps to give a more complete view of agent performance beyond just task accuracy.

Saturation of Computer-Use Agent Benchmarks Agent systems have improved a lot in a short time, and many of the earlier benchmarks are starting to look too easy. On MiniWoB++ (Kim et al., 2023), modern agents score close to perfect on most tasks. Even on harder benchmarks like WebArena (Zhou et al., 2024), the gap between top agents and humans is getting smaller every year. This trend suggests that soon current benchmarks might become insufficient for differentiating performance of strong systems. Some recent work attempted to address it. For example, WorkArena (Drouin et al., 2024) targets enterprise software tasks on the ServiceNow platform, and REAL (Garg et al., 2025) focuses on deterministic simulations of real websites that require longer interaction traces. But most of these benchmarks still test mostly text-heavy, form-based interactions. They do not really assess whether an agent can understand dynamic visual content, read graphical layouts, or reason about 3D scenes. Our benchmark targets exactly these underexplored skills, and our experiments show that even the strongest frontier models still perform far below human evaluators on these tasks.

I Discussion

Limitations Despite providing a challenging and professional benchmark for evaluating agent generalisation, GauntletBench covers only five specialised domains and a subset of interaction paradigms, which may not fully capture the diversity of real-world scenarios and could potentially introduce evaluation bias. In addition, our benchmark constrains agents' capabilities in visually grounded interaction to ensure faithful evaluation, which may limit their full potential when combined with more observational challenges.

Future Works We plan to expand GauntletBench to more diverse professional applications, richer multimodal interaction settings, and longer-horizon tasks. We also aim to incorporate additional evaluation dimensions, such as robustness, safety, and failure recovery, to better reflect the requirements of real-world deployment.

Broad Impacts We believe GauntletBench can support the development of more robust, reliable, and trustworthy agentic systems by providing realistic and challenging evaluation settings for assessing agent generalisation. By promoting more rigorous evaluation practices, our benchmark may facilitate future research on safer and more dependable autonomous agents in real-world applications.