WORKFLOW-GYM: TOWARDS LONG-HORIZON EVALUATION OF COMPUTER-USE AGENTIC TASKS IN REAL-WORLD PROFESSIONAL FIELDS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in AI agents have enabled increasingly complex interactions with real-world environments, yet existing GUI benchmarks largely focus on general-purpose software and short-horizon tasks. It therefore remains unclear whether modern agents can autonomously operate domain-specific professional software to complete long-horizon, economically valuable workflows end to end. To address this gap, we introduce Workflow-GYM, a benchmark for evaluating GUI agents on realistic professional workflows across diverse domains and specialized software environments. Extensive experiments show that even the strongest models achieve only slightly above 30% success rates, revealing substantial limitations in current GUI agents. Further analysis identifies several recurring failure modes, including workflow stage omission, error propagation, objective drift, and insufficient understanding of professional software environments. These results highlight long-horizon workflow consistency and reliable interaction with specialized software as key challenges for the next generation of GUI agents.

1 INTRODUCTION

In recent years, large language models (LLMs) and vision-language models (VLMs) have rapidly evolved from text-centric systems into multimodal agents capable of interacting with external environments (Yao et al., 2022; Wang et al., 2024). This shift raises a fundamental question: can AI systems autonomously complete complex, real-world tasks that constitute real professional work and therefore carry practical economic value? For many such tasks, especially in professional settings, the primary interface is not an API or command line, but graphical software operated through sequences of user interactions.

We refer to these end-to-end processes as *workflows*: structured, goal-directed sequences of interdependent GUI operations that transform an initial environment state into a desired outcome. Professional workflows often span multiple stages, involve specialized software and domain knowledge, and culminate in concrete artifacts or target environment states. Unlike isolated GUI actions or short-horizon tasks, completing such workflows requires agents to sustain coherent progress over extended interactions, recover from intermediate errors, and coordinate planning with execution toward the final objective. Professional workflows therefore provide a natural evaluation unit for assessing agents' ability to perform realistic, long-horizon, and economically meaningful GUI tasks.

Existing benchmarks, however, cover limited part of this setting. Prior GUI-agent benchmarks (Liu et al., 2025; Mu et al., 2025; Xie et al., 2024; Sun et al., 2026) primarily evaluate short-horizon, general-purpose interactions with common applications, and rarely involve specialized professional software or domain-specific expertise. In contrast, benchmarks targeting long-horizon reasoning or economically valuable tasks largely focus on text-based environments such as code generation, document understanding, and knowledge-intensive QA (Patwardhan et al., 2025; Ding et al., 2025; Yang et al., 2026). As a result, there remains a clear gap in evaluating agents on the intersection of **professional expertise, long-horizon execution, real GUI interaction, and economically meaningful outcomes**.

To bridge this gap, we introduce **Workflow-GYM**, a benchmark designed to evaluate whether GUI Agents can execute end-to-end, domain-specific workflows in realistic working environments. In

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.

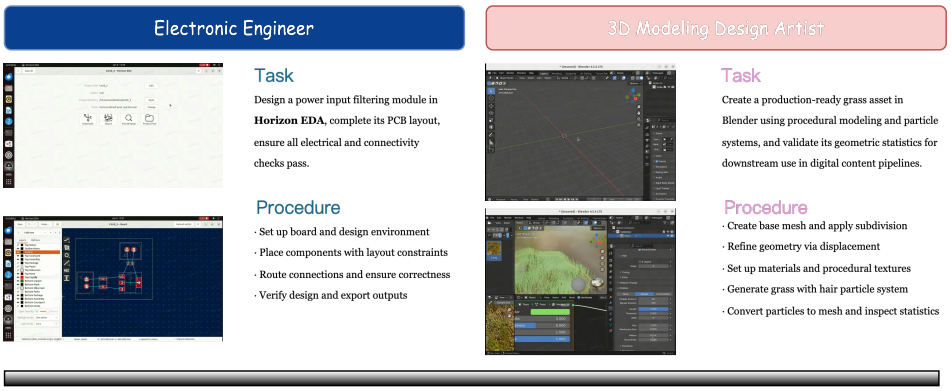


Figure 1: Examples of Workflow-GYM tasks from professional domains. Each task requires interacting with specialized software through GUI to accomplish a real-world objective.

Workflow-GYM, each task is instantiated in a fully configured environment with the required professional software pre-installed, and the agent is provided only with a natural language instruction describing the task objective—without access to intermediate guidance or additional input hints. Starting from this minimal specification, the agent must autonomously plan, reason, explore, and interact with the environment to progressively achieve the desired goal. Evaluation is performed based on the final state or artifact produced by the agent. For each task, we define task-specific and automatically verifiable success criteria, enabling scalable and reproducible assessment of task completion. This design ensures both evaluation reliability and scalability, while faithfully reflecting whether an agent can successfully carry out the entire workflow and produce the intended outcome in a real-world setting. Typical tasks of Workflow-GYM is shown at Figure 1.

Through extensive experiments with multiple state-of-the-art(SOTA) models under different agent frameworks, we find that long-horizon, domain-specific software workflows remain highly challenging for current systems. Even the strongest model can only achieve a success rate of 30%, highlighting a substantial gap between existing capabilities and the requirements of real-world tasks. Further analysis reveals several recurring failure modes, including workflow stage omission, error propagation, objective drift, and insufficient software-specific knowledge. Beyond these model-level limitations, Workflow-GYM also exposes a more fundamental challenge shared by current GUI-agent frameworks: the mismatch between continuous human interaction and the discrete observation-action paradigm adopted by most existing agents.

To summarize, our contributions are as follows:

- We introduce the concept of *workflow* as a unified abstraction of real-world, long-horizon, economically meaningful compute-use processes, and construct **Workflow-GYM**, a benchmark capturing such tasks across diverse professional domains.
- We develop a reliable and reusable data construction pipeline, combining expert-driven design, multi-stage validation, and environment-grounded execution, enabling scalable generation of high-quality, verifiable workflow tasks.
- We conduct a comprehensive evaluation of representative GUI agents and foundation models on Workflow-GYM, revealing substantial performance gaps and systematically analyzing key challenges, including long-horizon level failures, insufficient professional software knowledge, and repetitive action looping.

n

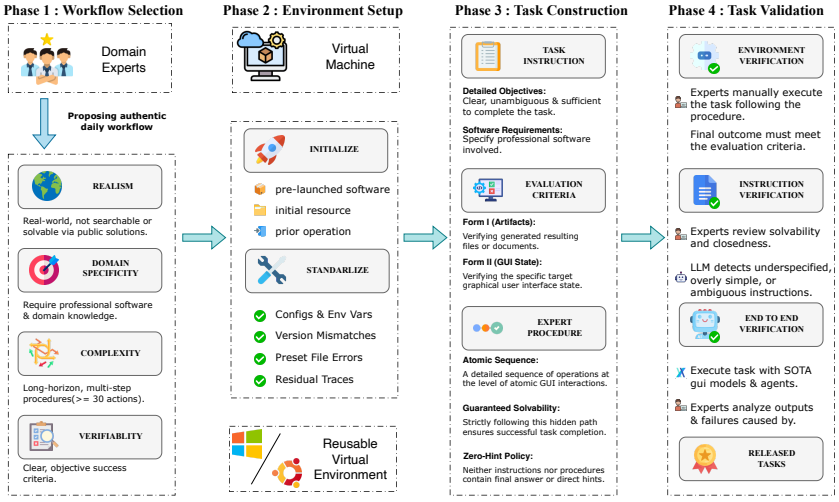


Figure 2: Construction pipeline of Workflow-GYM tasks.

2 Related Works

2.1 Benchmarks for GUI Agents and Long-Horizon Tasks

Existing GUI benchmarks cover both PC (Xie et al., 2024; Dai et al., 2025) and mobile environments (Rawles et al., 2024), spanning general-purpose interactions and domain- or software-specific applications. For example, AmbiBench (Sun et al., 2026) focuses on everyday mobile tasks, GUI-360 (Mu et al., 2025) evaluates productivity software, and Science-Board (Sun et al., 2025) targets scientific workflows. Despite this diversity, existing benchmarks predominantly evaluate relatively short-horizon interactions and rarely capture end-to-end professional workflows that require domain-specific software, specialized knowledge, and complex multi-stage execution.

2.2 MLLM-Driven GUI Agents

Compared with GUI question-answering benchmarks (Shi et al., 2025), end-to-end GUI interaction more directly evaluates an agent's ability to perceive, reason, and act in real interfaces. Recent advances in multimodal models have enabled increasingly capable GUI agents, including multi-agent frameworks such as Mobile-Agent-v2 (Deng et al., 2024) and dedicated agentic models trained for GUI interaction, such as UI-TARS (Wang et al., 2025) and Step-GUI (Yan et al., 2025). These systems improve planning, visual grounding, and long-horizon interaction through specialized architectures and training, but their ability to execute complex professional workflows in domain-specific software remains insufficiently studied.

3 Workflow-GYM

3.1 Dataset Construction

To faithfully evaluate the capability of GUI agents in completing real-world, high-value tasks, the benchmark must capture several key properties of professional workflows: Tasks should be grounded in authentic real-world scenarios to prevent retrieval-based solutions, require domain-specific knowledge and professional software, exhibit long-horizon multi-step complexity, and admit clear, objective success criteria for reliable and reproducible evaluation. Guided by these requirements, we design a multi-stage data construction pipeline that systematically sources, filters, instantiates, and validates tasks, ensuring both realism and evaluation rigor.

3.1.1 WORKFLOW SELECTION

We begin the construction pipeline by sourcing candidate tasks from domain experts across multiple professional fields. Experts are instructed to propose workflows grounded in their own daily work practices rather than synthetic or publicly documented examples. We then apply strict selection criteria, ensuring that each task reflects a realistic, domain-specific, long-horizon, and objectively verifiable professional workflow. Detailed criteria and expert demographics are provided in Appendix A and C.1. Applying this process, we curate an initial pool of over 1,000 candidate workflows for subsequent stages of benchmark construction.

3.1.2 ENVIRONMENT SETUP

To ensure reproducible and comparable evaluation, we adopt a two-level environment design consisting of reusable software-level base environments and lightweight task-specific initialization.

Base Environments. For each professional software and version, we construct a standardized virtual machine image with the required software and dependencies pre-installed and configured. These environments span both Windows and Ubuntu operating systems, depending on the native platform requirements of the corresponding software. Compared with containerized or API-based setups, full-system virtual machines better preserve real-world GUI behavior, including rendering, window management, and system-level dependencies. In total, Workflow-GYM contains 56 such software environments; the complete software list is provided in Appendix M.

Task Instantiation. At runtime, each task is initialized by injecting its required inputs, templates, and configuration files into the corresponding base environment. The software is pre-launched in a unified interaction-ready state, while no task-specific operations have been performed. Irrelevant setup steps, such as login or authentication, are removed.

This separation enables different tasks to share controlled software environments while retaining task-specific inputs and objectives, supporting scalable and repeatable evaluation across agents.

3.1.3 TASK CONSTRUCTION

Based on the selected workflows and their corresponding environments, domain experts instantiate each workflow into a fully specified and executable benchmark task. Each task contains three components: (1) a self-contained **task instruction** specifying the background, objectives, and required software; (2) an **expected outcome and evaluation criteria** defining the target artifact or GUI state and its objectively verifiable success conditions; and (3) a **step-by-step expert procedure** describing a valid sequence of atomic GUI operations that successfully completes the task. Together, these components ensure that each task is executable, objectively evaluable, and independently verified to be solvable in the provided environment. Detailed construction requirements for each component are provided in Appendix C.2.

3.1.4 TASK VALIDATION

To ensure task quality and reliability, we apply a three-stage validation process. **Environment Verification.** Domain experts manually execute each task following the annotated procedure to verify environment correctness and task solvability. **Instruction-Level Validation.** Senior domain experts and an LLM-based evaluator jointly inspect task instructions for completeness, ambiguity, solvability, and sufficient procedural complexity. **End-to-End Validation.** We further execute each task with randomly selected SOTA GUI agents and inspect their trajectories and outputs to uncover latent ambiguities, missing constraints, or environment inconsistencies that emerge only during actual execution. Tasks are iteratively revised and revalidated until all three stages are passed. Detailed validation protocols are provided in Appendix C.3.

3.2 BENCHMARK STATISTICS

Finally we construct **Workflow-GYM**, a benchmark evaluating the capabilities of GUI agents on completing compensated real-world professional computer-use tasks in different domains. Workflow-GYM comprises a total of **338** tasks, spanning 6 top-level domains and 23 fine-grained subdomains, ensuring broad coverage across diverse application areas. Detailed domain distribution

Table 1: Task difficulty distribution in Workflow-GYM based on the number of execution steps.

Difficulty Level	Step Range	Task Number	Percentage
Easy	30–44	129	38.2%
Medium	45–60	159	47.0%
Hard	61–110	50	14.8%

Table 2: Performance of different models on Workflow-GYM.

Model	**Avg Score (%)**	**Pass@3**	**Easy**	**Medium**	**Hard**
Gemini-3.1-Pro	**30.67**	41.12	39.28	26.62	21.33
Kimi-k2.6	29.68	**41.42**	34.63	27.04	25.33
GPT-5.4	17.85	26.33	21.71	16.14	13.33
Seed-2.0-lite	18.24	28.40	24.81	15.09	11.33
GPT-5.4-mini	15.98	27.22	18.09	15.51	12.00
Gemini-3-flash	7.89	15.98	12.14	6.08	2.67

is shown in Figure 5. Beyond domain diversity, the tasks exhibit substantial variation in procedural complexity: the number of execution steps required to complete a task ranges from 30 to 110. Based on this variation, we categorize task difficulty according to the length of the execution procedure shown in Table 1, using the number of required steps as a proxy for complexity.

3.3 Automatic Evaluation for Workflow-GYM Tasks

For each task in Workflow-GYM, evaluation is based on predefined task-specific criteria characterizing the expected final state or deliverable. Since different workflows culminate in different forms of deliverables, we adopt the verification mechanism best suited to the observable final outcome: non-artifact tasks are assessed from the final GUI state using a VLM, while artifact-producing tasks are verified through either deterministic rules or LLM-based assessment, depending on the artifact properties. Each task receives a binary score, with a score of 1 assigned only when all required criteria are satisfied and 0 otherwise. Detailed evaluation protocols and quality-control procedures are provided in Appendix D.

4 Experiments

4.1 Experimental Settings

We evaluate 6 representative models on Workflow-GYM: GPT-5.4-xhigh, GPT-5.4-mini-xhigh (OpenAI, 2026), Gemini-3.1-Pro, Gemini-3-Flash (Google Cloud, 2026a;b), Kimi-k2.6 (Moonshot AI, 2026), and Seed-2.0-Lite (Seed, 2026b). Each model receives only a natural-language task instruction and interacts with the task environment through GUI operations in an isolated VM. Since GUI models are tightly coupled with their native computer-use interfaces and action protocols, we evaluate each model with its **best-performing compatible agent framework and adaptation configuration** rather than enforcing a single shared action schema. All systems nevertheless share the same VM environment, screenshot-based observation setting, task initialization, and a maximum budget of 400 model–environment interaction rounds. Detailed framework differences are provided in Appendix G. Each task is evaluated over **3 independent trials** from the same initial state. We report **Avg Score**, averaging binary success over all task–trial executions, and **Pass@3**, which counts a task as successful if any of its 3 trials succeeds. Rule-based tasks are scored by task-specific verifiers; otherwise, Seed-1.8 (Seed, 2026a) evaluates the final outcome using predefined task-specific criteria.

4.2 Main Results

As shown at the results in Table 2, several significant observations emerges:

Long-Horizon professional task is still a challenging field for GUI-Agents and models. As shown in Table 2, even the best-performing models, Gemini-3.1-Pro and Kimi-k2.6, merely achieves an average score rate of 30.67% and pass@3 rate of 41.42%, respectively. Moreover, the majority of models exhibit pass@3 rates below 30%. In contrast, SOTA models report success rates exceeding 70% on benchmarks such as OS-World (Xie et al., 2024). This substantial performance gap suggests that, while current models and agent frameworks demonstrate strong capabilities in short-horizon tasks and interactions with commonly used general-purpose software, they remain significantly limited in professional, domain-specific settings. In particular, their ability to autonomously execute long-horizon, multi-step workflows in specialized software environments and complete tasks end-to-end at a human level still requires considerable improvement.

Model performance decreases with workflow length. As shown in Table 2, all models exhibit lower success rates on tasks with longer expert-annotated procedures. Further task-level analysis, detailed in Appendix E, further confirms a significant negative association between procedure length and model performance, persisting after controlling for task and evaluation characteristics. Longer workflows involve more interdependent stages and opportunities for execution errors, posing greater challenges to sustained planning, error recovery, and reliable GUI interaction. These findings identify expert-procedure length as a useful empirical proxy for workflow difficulty, highlighting the challenges of long-horizon execution in professional GUI environments.

Performance varies substantially across domains. As shown in Appendix F, models exhibit distinct domain-specific strengths: Kimi-K2.6 outperforms on *Data Analysis* and *Multimedia & Creative*, while Gemini-3.1-Pro leads on *Geography & Environment*, *Engineering & Design*, and *Scientific Computing*. Beyond model-specific capability biases, performance also depends strongly on domain characteristics. Agents generally perform better in structured and standardized workflows, where interactions are more discrete and predictable, but struggle in visually open-ended environments that require precise spatial manipulation, visual grounding, and fine-grained GUI control.

Model–framework coupling is intrinsic to current GUI agents. Unlike coding agents that often operate through standardized text or terminal interfaces (Wang et al., 2024; Merrill et al., 2026), models are typically optimized for specific computer-use protocols, including tool-call schemas, coordinate conventions, and action execution formats. For example, Gemini predicts coordinates on a normalized $[0, 1000]$ grid and requires model-specific transformation to native pixel space. Since forcing heterogeneous models into a single action interface can introduce substantial model-specific mismatch, we evaluate each model with a compatible framework while keeping the VM environment, observation modality, initialization, and task information fixed. Detailed configurations are provided in Appendix G.

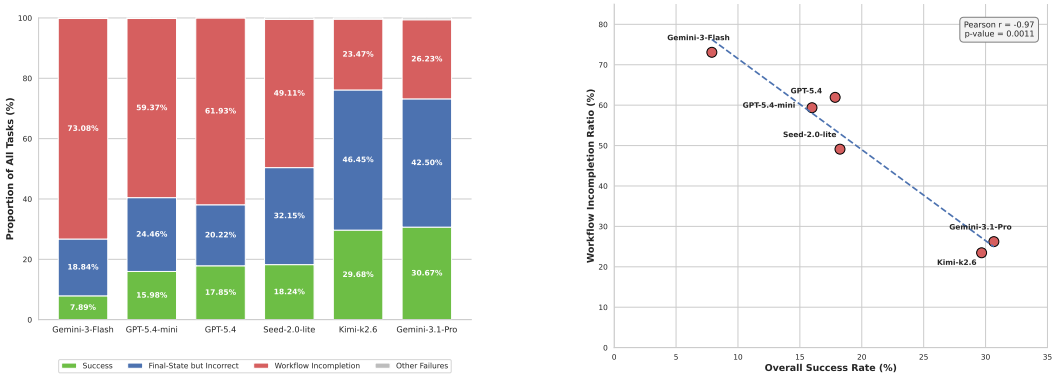
5 Analysis and Discussions

5.1 Outcome-level Failure Decomposition

To better understand failures in Workflow-GYM from an outcome perspective, we categorize unsuccessful executions into two outcome-based types:

- **Workflow Incompletion** refers to cases where the agent fails to complete the workflow or reach the required final state, such as terminating early, failing to produce the required artifact, or leaving the software in an intermediate state.
- **Final-State but Incorrect** refers to cases where the agent reaches the final stage and produces the expected artifact or GUI state, but the result violates one or more task requirements due to incorrect parameters, missing components, misconfigurations, or other execution errors.

Figure 3a presents the distribution of these two failure categories across different models. A clear trend emerges: models with higher overall success rates exhibit substantially lower workflow incompletion rates, whereas lower-performing models fail predominantly because they are unable to complete the workflow. To further quantify this relationship, we compute the correlation between workflow incompletion rate and overall model performance. As shown in Figure 3b, the two variables exhibit a strong negative correlation (Pearson $r = -0.97$), indicating that workflow incompletion decreases consistently as model capability improves. Importantly, the trend remains strong even



(a) Proportion of different outcome-level failure reason of different models on Workflow-GYM.

(b) Correlation between model performances and workflow-incompletion rates.

Figure 3: Outcome-Level failure calculation of Workflow-GYM.

after normalizing workflow incompletion by all failed executions (Pearson $r = -0.94$; detailed in Appendix H).

This observation suggests that, for current models and GUI agents, successfully completing a long-horizon workflow remains a primary bottleneck. In particular, weaker models often fail before reaching the final stage of the task, implying deficiencies in long-range planning, state tracking, error recovery, and sustained execution. By contrast, stronger models are considerably more likely to complete the workflow, and their failures increasingly arise from execution-level inaccuracies after reaching the final stage. In other words, as model capability improves, the dominant failure mode gradually shifts from **failing to finish the workflow** to **finishing the workflow but producing an incorrect result**. This transition highlights that workflow completion itself is a critical capability barrier for current GUI agents and underscores the unique challenges posed by long-horizon professional workflows. Details of outcome level failure analysis can be found at Appendix H.

5.2 Behavior-level Failure Analysis

Outcome-level analysis captures how executions fail, but not why. We therefore examine agent trajectories and identify two dominant behavior-level failure modes: **Long-Horizon-Level Failures**, which arise from difficulties in maintaining coherent progress, cross-step consistency, and completion over extended workflows; and **Execution-Level Failures**, which stem from unreliable interaction with professional software, including visual grounding errors, insufficient software-specific knowledge, and complex GUI manipulation failures.

5.2.1 Long-Horizon-Level Failures

Error Propagation: The agent performs an incorrect operation that introduces a deviation from the intended workflow state. Since professional workflows typically consist of tightly coupled stages with strong inter-stage dependencies, subsequent actions often rely on the correctness of preceding states. Once execution diverges from the intended trajectory, later actions may remain locally plausible but are performed under invalid assumptions and therefore fail to make meaningful progress toward the task objective. Without effective recovery, the deviation persists and propagates through subsequent stages, ultimately leading to task failure. A typical case is shown at Appendix I.1.

Workflow Stage Omission: The agent skips a mandatory intermediate stage in a multi-stage workflow and proceeds directly to subsequent operations. Because many professional workflows contain strong stage dependencies, later actions often rely on artifacts or states produced by earlier stages. As a result, the execution trajectory diverges from the intended workflow, and the task may become unrecoverable even though the agent continues making seemingly reasonable progress. This failure mode is particularly prevalent in long-horizon workflows, where errors introduced at one stage can propagate across many subsequent steps. A typical example is shown at Appendix I.2.

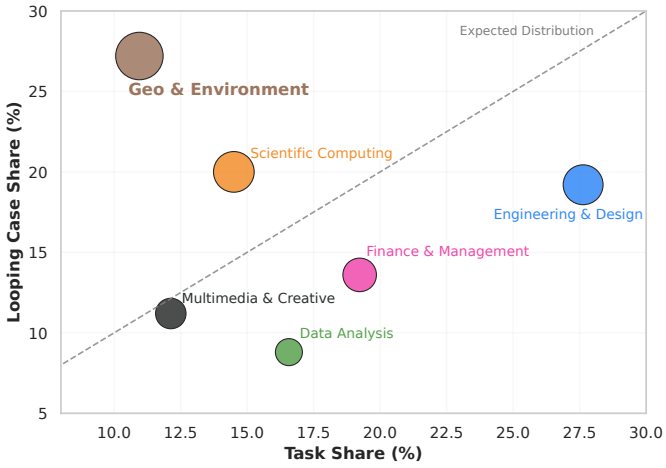


Figure 4: Repetitive action and its portion across different task domains.

Objective Drift: The agent gradually deviates from the original workflow objective by over-focusing on intermediate states or local subgoals. In long-horizon workflows, many actions are only transitional; once an intermediate state is mistakenly treated as the target, subsequent decisions may remain locally plausible while progressively diverging from the intended outcome. An example of objective drift in Workflow-GYM is shown in Appendix I.3.

5.2.2 Execution-Level Failures

Prior work on GUI agents has documented common execution-level errors, such as mis-clicks or misinterpreting interface elements. Beyond these, Workflow-GYM reveals additional failure modes—namely, insufficient software knowledge and repetitive action loops—that are particularly prominent in professional workflows.

Software Knowledge Deficiency: The agent lacks the software-specific knowledge required to correctly interpret functionality, identify appropriate operations, or navigate domain-specific workflows. As a result, it may select incorrect actions, repeatedly search for non-existent functionalities, or fail to utilize the tools necessary to accomplish the task. Unlike failures caused by inaccurate perception or manipulation errors, these failures stem from an incomplete understanding of the software environment itself. This failure mode is particularly common in professional applications, where successful task completion often requires substantial knowledge of software-specific concepts, interfaces, and operational procedures. A typical case of failures cased by lacking relevant software knowledge is shown at Appendix I.4.

Repetitive Action Looping: The agent becomes trapped in a repetitive execution pattern, repeatedly issuing the same action without meaningful progress. We identify a looping failure when the model produces the same action sequence for 10 consecutive interaction turns, typically indicating that the agent is stuck in a local state and unable to recover or explore alternatives. We observe over 100 such cases in Workflow-GYM, with substantial variation across domains (Figure 4). Geography & Environment accounts for 27.20% of looping cases despite comprising only 10.95% of benchmark tasks (2.48× enrichment), while Scientific Computing is also overrepresented (1.38×) and Data Analysis substantially underrepresented (0.53×). These results suggest that repetitive looping is more likely in workflows involving complex interfaces and specialized operational procedures.

5.3 From Continuous Interaction to Discrete Observation: A Structural Bottleneck of GUI Agents

Beyond model-level limitations, Workflow-GYM reveals a structural bottleneck shared by most current GUI-agent frameworks. Many professional GUI operations—such as precise drag-and-drop, object alignment, curve drawing, and geometric editing—require *continuous visual feedback*: human monitor intermediate interface states while acting and dynamically adjust their movements. In

contrast, current GUI agents typically follow a snapshot-based perception-action loop, observing the interface before an action and receiving the next observation only after that action has completed.

This discretization makes intermediate execution states invisible to the agent, preventing real-time correction, progress monitoring, and visually grounded termination during continuous manipulation. Consequently, operations that are naturally closed-loop for humans are reduced to coarse open-loop actions or repeated trial-and-error for agents. The prevalence of such operations in Workflow-GYM exposes a limitation that cannot be addressed solely by stronger planning or reasoning, but may require GUI-agent architectures with continuous or higher-frequency perception-action interaction. A representative example is shown in Appendix K.

5.4 Ablation Study: Effects of Procedural Guidance

In the main experiment, agents are provided only with task instructions specifying the target outcome and requirements. To probe the bottlenecks of long-horizon GUI tasks, we introduce 2 procedural-guidance ablations beyond basic setting: a textual step-by-step tutorial and a video tutorial demonstrating the corresponding operations. It allows us to assess how explicit procedural guidance affects performance and whether visual demonstrations provide complementary information beyond text.

5.4.1 Effect of Textual Step-by-Step Procedures

To assess model performance under reduced planning complexity, we conduct an ablation in which models are provided with expert-annotated step-by-step operational instructions specifying the sequence of actions required to complete each task. By largely removing the need for high-level workflow planning, this setting allows us to better isolate models' ability to execute long-horizon GUI workflows under structured procedural guidance.

As shown in Appendix J.1, explicit procedures substantially improve performance across models, mitigating failures caused by incorrect operation ordering, cascading errors, and parameter or module misconfiguration. Seed-2.0-lite and GPT-5.4 (including mini) exhibit particularly large gains; given their relatively high workflow-incompletion rates, the improvement suggests that structured procedures also help reduce premature termination by encouraging sequential execution. Nevertheless, substantial failures remain even with detailed guidance: inaccurate click localization and early stopping continue to limit success, while models may skip prescribed steps or deviate from the provided procedure through autonomous decisions. These results indicate that high-level planning is an important bottleneck, but not the only source of failure in long-horizon GUI execution.

5.4.2 Complementary Value of Visual Demonstrations

Although textual procedures reduce planning difficulty, they may not convey fine-grained GUI interaction details. We therefore evaluate an additional video-guidance setting on 100 tasks across professional-software categories using two models, where textual procedures are paired with reference frames demonstrating the corresponding operations. As shown in Table 8, visual demonstrations achieve the highest pass@1 for both models, improving Seed-2.0-lite from 28% to 31% and Gemini-3.1-Pro from 28% to 35%. This suggests that visual demonstrations provide complementary interaction-level information beyond text.

We can also notice that video guidance does not uniformly shorten execution trajectories: successful runs have similar median lengths across settings, while failed runs become substantially longer. This suggests that visual demonstrations mainly help agents persist on difficult tasks rather than simply execute already-solvable tasks more efficiently. Nevertheless, substantial failures remain on professional software requiring fine-grained manipulation, indicating that visual guidance alleviates but does not eliminate execution-level bottlenecks. Representative cases are provided in Appendix J.3.

6 Conclusion

We introduce Workflow-GYM, the first benchmark designed to rigorously evaluate the capability of GUI agents on real-world, long-horizon professional tasks across diverse domains and specialized software environments. Experimental results on representative models show that, although current agents can interact with graphical interfaces and execute low-level actions, long-horizon work-

flows remain highly challenging due to the substantial demands they place on complex decision-making, precise execution, and the coherent integration of both over extended interaction trajectories. Workflow-GYM provides a crucial testbed for advancing the next generation of GUI agents, and offers a concrete step toward more capable, reliable, and practically useful computer-use systems.

AI USE STATEMENT

Generative AI tools were used to assist with LaTeX template migration, formatting diagnostics, and language polishing. All AI-assisted edits were reviewed by the authors, who take responsibility for the final manuscript content.

REPRODUCIBILITY STATEMENT

We make several efforts to ensure the reproducibility of our benchmark and experimental results. We document the benchmark construction and evaluation protocols in detail, and report all major experimental settings, including the evaluated models, judge models, execution budgets, trial nums, in the main text and appendices. These resources and specifications are intended to promote reproduction of both the benchmark evaluation and the analyses presented in this work.

REFERENCES

Yutong Dai, Krithika Ramakrishnan, Jing Gu, Matthew Fernandez, Yanqi Luo, Viraj Prabhu, Zhenyu Hu, Silvio Savarese, Caiming Xiong, Zeyuan Chen, et al. Scuba: Salesforce computer use benchmark. *arXiv preprint arXiv:2509.26506*, 2025.

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Liujianfeng Liujianfeng, Ang Li, Jian Luan, Bin Wang, Rui Yan, et al. Mobile-bench: An evaluation benchmark for llm-based mobile agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8813–8831, 2024.

Jingzhe Ding, Shengda Long, Changxin Pu, Huan Zhou, Hongwan Gao, Xiang Gao, Chao He, Yue Hou, Fei Hu, Zhaojian Li, et al. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2025.

Google Cloud. Gemini 3.1 Pro Model Card, 4 2026a. URL https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/3-1-pro?hl=zh-cn.

Google Cloud. Gemini 3 Flash Model Card, 4 2026b. URL https://docs.cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/3-flash.

Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, et al. Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc. *arXiv preprint arXiv:2502.14282*, 2025.

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

Moonshot AI. Kimi k2.6: Advancing open-source coding. https://www.kimi.com/blog/kimi-k2-6, 2026. Accessed: 2026-06-02.

Jian Mu, Chaoyun Zhang, Chiming Ni, Lu Wang, Bo Qiao, Kartik Mathur, Qianhui Wu, Yuhang Xie, Xiaojun Ma, Mengyu Zhou, et al. Gui-360: A comprehensive dataset and benchmark for computer-using agents. 2025.

OpenAI. Gpt-5.4 thinking system card. Technical report, OpenAI, 2026. URL https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf. Accessed: 2026-04-27.

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

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*, 2024.

Bytedance Seed. Seed1. 8 model card: Towards generalized real-world agency. *arXiv preprint arXiv:2603.20633*, 2026a.

Bytedance Seed. Seed2. 0 model card: Towards intelligence frontier for real-world complexity. *URL https://lf3-static. bytednsdoc. com/obj/eden-cn/lapzild-tss/ljhwZthlaukjlkulzlp/seed2/0214/Seed2. 0% 20Model% 20Card. pdf*, 2026b.

Chenrui Shi, Zedong Yu, Zhi Gao, Ruining Feng, Enqi Liu, Yuwei Wu, Yunde Jia, Liuyu Xiang, Zhaofeng He, and Qing Li. Gui knowledge bench: Revealing the knowledge gap behind vlm failures in gui tasks. *arXiv preprint arXiv:2510.26098*, 2025.

Jiazheng Sun, Mingxuan Li, Yingying Zhang, Jiayang Niu, Yachen Wu, Ruihan Jin, Shuyu Lei, Pengrongrui Tan, Zongyu Zhang, Ruoyi Wang, et al. Ambibench: Benchmarking mobile gui agents beyond one-shot instructions in the wild. *arXiv preprint arXiv:2602.11750*, 2026.

Qiushi Sun, Zhoumianze Liu, Chang Ma, Zichen Ding, Fangzhi Xu, Zhangyue Yin, Haiteng Zhao, Zhenyu Wu, Kanzhi Cheng, Zhaoyang Liu, et al. Scienceboard: Evaluating multimodal autonomous agents in realistic scientific workflows. *arXiv preprint arXiv:2505.19897*, 2025.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, et al. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025.

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

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

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, Shiliang Yang, Zhirui Wang, Brian Li, Kang An, Chenyang Li, Lei Lei, Mengmeng Duan, Danxun Liang, Guodong Liu, Hang Cheng, Hao Wu, Jie Dong, Junhao Huang, Mei Chen, Renjie Yu, Shunshan Li, Xu Zhou, Yiting Dai, Yineng Deng, Yingdan Liang, Zelin Chen, Wen Sun, Chengxu Yan, Chunqin Xu, Dong Li, Fengqiong Xiao, Guanghao Fan, Guopeng Li, Guozhen Peng, Hongbing Li, Hang Li, Hongming Chen, Jingjing Xie, Jianyong Li, Jingyang Zhang, Jiaju Ren, Jiayu Yuan, Jianpeng Yin, Kai Cao, Liang Zhao, Liguo Tan, Liying Shi, Mengqiang Ren, Min Xu, Manjiao Liu, Mao Luo, Mingxin Wan, Na Wang, Nan Wu, Ning Wang, Peiyao Ma, Qingzhou Zhang, Qiao Wang, Qinlin Zeng, Qiong Gao, Qiongyao Li, Shangwu Zhong, Shuli Gao, Shaofan Liu, Shisi Gao, Shuang Luo, Xingbin Liu, Xiaojia Liu, Xiaojie Hou, Xin Liu, Xuanti Feng, Xuedan Cai, Xuan Wen, Xianwei Zhu, Xin Liang, Xin Liu, Xin Zhou, Yifan Sui, Yingxiu Zhao, Yukang Shi, Yunfang Xu, Yuqing Zeng, Yixun Zhang, Zejia Weng, Zhonghao Yan, Zhiguo Huang, Zhuoyu Wang, Zihan Yan, Zheng Ge, Jing Li, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, and Daxin Jiang. Step-gui technical report, 2025. URL https://arxiv.org/abs/2512.15431.

Qianyu Yang, Yang Liu, Jiaqi Li, Jun Bai, Hao Chen, Kaiyuan Chen, Tiliang Duan, Jiayun Dong, Xiaobo Hu, Zixia Jia, et al. \onemillion-bench: How far are language agents from human experts? *arXiv preprint arXiv:2603.07980*, 2026.

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*, 2022.

A Annotator and Domain Expert Background

Workflow-GYM focuses on long-horizon professional workflows grounded in real-world domain practices. Therefore, the construction and validation of tasks require substantial domain expertise, familiarity with professional software ecosystems, and practical operational experience. To ensure realism and reliability, Workflow-GYM adopts an expert-driven construction pipeline in which domain experts participate throughout the full benchmark lifecycle, including workflow proposal, task specification, environment verification, instruction refinement, and evaluation validation.

The expert team, consisting of 71 members, covers a broad range of professional domains, including all the task domains in our benchmark. Rather than relying on synthetic or publicly templated tasks, experts contribute workflows derived from authentic professional practices and real software usage scenarios.Table 3 summarizes the composition of the expert team involved in Workflow-GYM construction.

Category	**Item**	**Count**
Coverage	Total Experts	71
	High-Level Industries	28
	Professional Subdomains	51
Occupation	Industry Practitioners	51
	Students / Trainees	20
Education	PhD Degree Holders	4
	Master's Degree Holders	23
	Bachelor's Degree Holders	44
Experience	>5 Years	30
	3–5 Years	5
	<3 Years	36

Table 3: Summary statistics of the domain experts participating in Workflow-GYM construction and validation.

B Domain distribution of Workflow-GYM

The detailed distribution of task domains in Workflow-GYMs shown at Figure 5.

C Tutorial for Workflow-GYM Task Construction

C.1 Workflow-Selection

To further enforce the desired task properties, we apply a set of strict filtering criteria. Specifically, each task must satisfy all of the following requirements:

- **Realism**. Tasks must originate from authentic real-world workflows and should not be directly searchable or solvable via publicly available solutions, ensuring that success cannot be achieved through simple retrieval or memorization.
- **Domain Specificity**. Completing the task must require the use of professional software and domain-specific knowledge, rather than general-purpose tools alone.
- **Complexity**. Tasks must involve long-horizon, multi-step procedures, with a minimum of 30 atomic actions, to reflect realistic workflow complexity.
- **Verifiability**. Each task must admit a clear and objective success criterion, enabling a reliable evaluation of whether the task has been completed correctly.

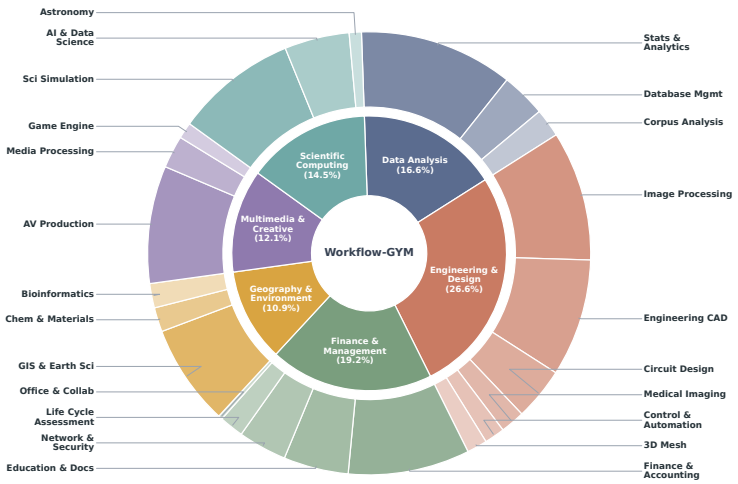


Figure 5: Task domains and distribution of Workflow-GYM.

C.2 Task Construction

Given a selected source workflow and its corresponding environment, the annotator should instantiate the workflow into a complete benchmark task that can be executed and evaluated independently. The constructed task should faithfully reflect the original professional workflow while providing all information necessary for task execution and verification within the prepared environment.

For each task, the annotator is required to provide the following three components and ensure that each component satisfies the corresponding requirements described below.

(1) **Task Instruction.** Domain experts provide a complete and self-contained task description, including necessary background, detailed objectives(final state, artifacts and directories), and the professional software involved. The instruction is required to be clear, unambiguous, and sufficient for completing the task without relying on external information.

(2) **Expected Outcome and Evaluation Criteria.** Each task is associated with a well-defined target outcome and corresponding evaluation criteria. Depending on the task type, the outcome takes one of two forms: (i) a resulting artifact (e.g., files or documents), or (ii) a target graphical user interface (GUI) state. In both cases, experts specify explicit requirements that the final output must satisfy. Evaluation is expected to perform in a binary manner, where a task is considered successful if and only if all criteria are met. To ensure robustness and reproducibility, the evaluation criteria are restricted to deterministic and reliably reproducible aspects of the outcome. Factors that are inherently stochastic or environment-dependent are excluded from evaluation whenever possible.

(3) **Step-by-Step Expert Procedure.** To enhance verifiability and ensure task solvability, experts provide a detailed sequence of operations required to complete the task, specified at the level of atomic GUI interactions. These procedures are constructed such that strictly following them leads to successful task completion under the defined evaluation criteria. Importantly, neither the task instructions nor the procedural annotations contain the final answer or any direct hints that would trivially reveal it.

C.3 Task Validation

After a task is constructed, domain experts are required to validate it through three stages before it can be included in Workflow-GYM. The goal of this process is to ensure that each task is executable in the prepared environment, sufficiently specified by its instruction, and robust under actual agent interaction. If any issue is identified at any stage, the task should be revised and revalidated.

(1) Environment Verification. The expert should first execute the task manually in the prepared environment by strictly following the annotated step-by-step procedure. The expert should verify that all required software, files, dependencies, and task-specific resources are correctly initialized, and that the prescribed operations can be completed from the designated initial state. The resulting artifact or GUI state should satisfy all predefined evaluation criteria. Any failure caused by software-version mismatch, missing resources, incorrect preset files, unexpected residual states, or environment configuration should be corrected before proceeding to subsequent validation.

(2) Instruction-Level Validation. The task instruction should then be reviewed for whether it provides all information necessary to complete the task without additional clarification. Experts should check that the task objective, required outputs, relevant parameters, file paths, and other necessary constraints are clearly specified and mutually consistent. The instruction should not contain ambiguous requirements, missing prerequisites, contradictory conditions, or unintended shortcuts that make the task substantially easier than the intended workflow. In addition to expert review, an LLM-based evaluator is used to identify potential underspecification, ambiguity, solvability issues, and insufficient procedural complexity. Any identified issue should be manually inspected and revised when necessary.

(3) End-to-End Validation. Finally, the task should be executed by randomly selected SOTA GUI agents under the same interaction setting used for benchmark evaluation. Experts should inspect the resulting trajectories and outputs rather than considering task success alone. This stage is intended to identify latent problems that may not be apparent during manual execution, such as instructions that admit multiple unintended interpretations, missing constraints exposed by agent exploration, unexpected GUI states, or environment behaviors that cause systematic failures unrelated to agent capability. Failures attributable to task design or environment configuration should trigger task revision, whereas genuine model execution errors should not.

A task is retained only after passing all three stages. Revised tasks should repeat the relevant validation stages until the environment, instruction, and end-to-end execution are jointly verified.

D Evaluation Protocol and Quality Control

To ensure reliable and reproducible task evaluation, we define a set of task-specific evaluation criteria for each task during benchmark construction, before any model execution is evaluated. These criteria are derived from the task specification and its intended final state, and remain fixed across all models, runs, and generated outputs. Evaluation therefore consists of verifying whether the produced final outcome satisfies these predefined criteria, rather than allowing the evaluator to infer or redefine success based on the observed output.

According to the form of the required final outcome, we adopt three verification mechanisms. Among the 338 tasks in Workflow-GYM, 168 tasks are evaluated from the final GUI state using a VLM-based evaluator, 68 tasks are evaluated through deterministic rule-based inspection of the generated artifact, and the remaining 102 tasks use LLM-based artifact evaluation.

Rule-based artifact evaluation. For tasks whose outputs can be directly inspected through structured or machine-readable properties, we translate the predefined task-specific evaluation criteria into deterministic programmatic checks. The evaluator directly verifies required properties of the generated artifact, such as predefined values, object existence, structural constraints, or other task-specific conditions. These checks are specified and manually validated during task construction and are not adapted to individual model outputs. Since criterion verification is performed through explicit programmatic rules, these tasks do not rely on stochastic model judgments.

Model-based evaluation. For tasks whose predefined evaluation criteria cannot be fully verified through deterministic programmatic rules, we use model-based evaluators. This includes VLM-based evaluation for tasks whose required outcome is defined by the final GUI state and LLM-based evaluation for artifacts requiring semantic or content-level inspection. Importantly, the model evaluator does not determine what constitutes a successful result. Instead, it is provided with the task-specific criteria established prior to evaluation and is used only to verify whether each required condition is satisfied by the observed final state or generated artifact. Thus, model-based evaluation serves as a criterion-verification mechanism rather than an unconstrained holistic judge of output quality.

Evaluation quality control. We further apply an iterative human-audit procedure to validate the reliability of model-based criterion verification. During evaluator development, we randomly sample task executions and model-generated results, independently assess whether each predefined criterion is satisfied through human inspection, and compare these judgments with the corresponding automatic evaluation results. Disagreement cases are manually reviewed to identify ambiguities or deficiencies in the criterion specification, verification procedure, or model-based judging logic. The evaluator or criterion implementation is then revised where necessary, while the underlying task requirements remain fixed. This process is repeated until the evaluation procedure is sufficiently stable.

Across more than 100 sampled evaluation cases covering both LLM-based artifact evaluation and VLM-based GUI-state evaluation, the finalized automatic evaluation achieves over **90%** agreement with human criterion-level judgments. These results provide empirical evidence that the model-based evaluators can reliably execute the predefined verification procedure while enabling scalable evaluation across the full benchmark.

E Analysis on the Effect of Workflow Length on Model Performance

To further examine whether longer workflows are systematically more challenging for current GUI agents, we analyze the relationship between expert-procedure length and task-level model performance. For each task, we aggregate the binary outcomes of all six evaluated models over three independent trials, resulting in 18 executions per task. The task-level score is defined as the average success rate across these 18 executions, yielding one observation for each of the 338 tasks. This avoids treating repeated executions of the same task as independent task-level samples.

Descriptive trend. We first examine how aggregate model performance varies with expert-procedure length. For each task i, we define its task-level score by averaging the binary outcomes over all six models and three independent trials:

$$\bar{y}_i = \frac{1}{18} \sum_{m=1}^{6} \sum_{r=1}^{3} y_{imr}, \tag{1}$$

where $y_{imr} \in \{0, 1\}$ indicates whether model m successfully completes task i in trial r. Thus, each of the 338 tasks contributes exactly one observation to the following analysis, with all models and trials weighted equally.

For each procedure-length group g, we compute the average number of expert-annotated steps and the average task-level score as

$$\overline{S}_g = \frac{1}{|G_g|} \sum_{i \in G_g} S_i, \qquad \overline{Y}_g = \frac{1}{|G_g|} \sum_{i \in G_g} \bar{y}_i, \tag{2}$$

where S_i denotes the number of expert-annotated steps for task i and G_g denotes the set of tasks in group g. As shown in Table 4, the short, medium, and long groups contain 129, 159, and 50 tasks, with average procedure lengths of 40.74, 51.27, and 72.64 steps, respectively. Their corresponding average scores decrease monotonically from 25.11% to 17.75% and 14.33%, yielding a 10.77-percentage-point difference between the short and long groups.

To avoid relying solely on discrete length groups, we further analyze procedure length as a continuous variable across all 338 tasks. Specifically, we compute the Pearson correlation between S_i and $\bar{y}_i$,

$$r = \frac{\sum_i (S_i - \bar{S})(\bar{y}_i - \bar{Y})}{\sqrt{\sum_i (S_i - \bar{S})^2} \sqrt{\sum_i (\bar{y}_i - \bar{Y})^2}}, \tag{3}$$

as well as the Spearman rank correlation ρ between their corresponding ranks. Both correlations are significantly negative ($r = -0.206$, $p < 0.001$; $\rho = -0.232$, $p < 0.001$), showing that the negative association persists when procedure length is treated continuously and is not an artifact of the predefined length buckets.

Procedure Length	Task Nums	Avg. Steps	Avg. Score
Short	129	40.74	25.11%
Medium	159	51.27	17.75%
Long	50	72.64	14.33%

Table 4: Task-level model performance across different expert-procedure lengths.

Specification	Effect per +10 Steps	95% CI	p-value
Step length only	-4.59 pp	$[-7.80, -1.38]$	0.0050
+ Domain and evaluation controls	-4.01 pp	$[-6.99, -1.03]$	0.0083

Table 5: Task-level regression between expert-procedure length and aggregate model performance.

Controlled regression analysis. We further conduct task-level regression analyses to examine whether the negative association between procedure length and model performance persists after accounting for differences in task composition. For interpretability, we rescale the step count as

$$Steps10_i = \frac{S_i - \bar{S}}{10}, \tag{4}$$

where $\bar{S}$ is the mean procedure length across all tasks. Consequently, the coefficient of $Steps10_i$ directly represents the change in average task success rate associated with an additional 10 expert-annotated steps. In our dataset, the mean procedure length is 50.41 steps, with a standard deviation of 11.99 and a range from 30 to 110 steps.

We first estimate an unadjusted model containing only procedure length:

$$\bar{y}_i = \alpha + \beta Steps10_i + \epsilon_i. \tag{5}$$

We then introduce controls for the primary task domain, the official three-class evaluation mode, and operating system:

$$\bar{y}_i = \alpha + \beta Steps10_i + \boldsymbol{\gamma}^\top \mathbf{D}_i + \boldsymbol{\delta}^\top \mathbf{E}_i + \boldsymbol{\theta}^\top \mathbf{O}_i + \epsilon_i, \tag{6}$$

where $\mathbf{D}_i$, $\mathbf{E}_i$, and $\mathbf{O}_i$ are categorical indicator vectors for domain, the official three-class evaluation mode, and operating system, respectively. The three evaluation modes are GUI-state evaluation(VLM-based), reproducible file evaluation(Rule-based), and non-reproducible file evaluation(LLM-Based). The coefficient β is the quantity of primary interest: a negative β indicates that longer expert procedures are associated with lower task-level model performance after holding these observed task characteristics constant. Standard errors are clustered by software environment to allow residuals from tasks within the same software to be correlated.

The unadjusted regression gives $\beta = -0.0459$, corresponding to a 4.59-percentage-point decrease in average task score for every additional 10 expert steps (95% CI: $[-7.80, -1.38]$ pp, $p = 0.0050$). After introducing the task and evaluation controls, the estimated effect remains negative and statistically significant at $\beta = -0.0401$, corresponding to a 4.01-percentage-point decrease per additional 10 steps (95% CI: $[-6.99, -1.03]$ pp, $p = 0.0083$). The persistence of the negative coefficient after adjustment indicates that the overall step-length gradient cannot be explained solely by coarse differences in domain or evaluation settings.

Without additional controls, every 10-step increase in expert-procedure length is associated with a 4.59-percentage-point decrease in average task success rate. After controlling for domain and evaluation-related characteristics, the effect remains negative and statistically significant, with an estimated decrease of 4.01 percentage points per additional 10 steps. These results suggest that the overall relationship between procedure length and task performance cannot be explained solely by coarse domain differences or evaluation settings.

Taken together, these results provide consistent evidence that tasks requiring longer expert procedures tend to be more challenging for current GUI agents. The trend appears both in discrete procedure-length groups and in continuous task-level analysis, and remains significant after controlling for major task and evaluation characteristics. We therefore view expert-procedure length as a useful empirical proxy for the aggregate difficulty of long-horizon workflows.

Model	Data Analysis	Eng & Design	Fin & Management	Geo & Environment	MM & Creative	Sci Computing
Gemini-3.1-Pro	39.88	**24.44**	**37.44**	**27.93**	21.95	**31.97**
Kimi-k2.6	**45.24**	21.48	32.82	25.23	**27.64**	27.89
GPT-5.4	26.19	10.37	24.10	17.12	17.89	14.29
GPT-5.4-mini	19.05	12.59	20.00	9.01	17.07	17.69
Seed-2.0-lite	16.07	10.74	24.62	18.02	18.70	25.85
Gemini-3-flash	10.12	6.67	9.23	0.90	8.94	10.20
All Models Avg.	26.09	14.38	24.70	16.37	18.70	21.32

Table 6: Performance breakdown across different primary domains in Workflow-GYM.

Importantly, this analysis establishes an association rather than a causal effect of horizon length itself. Longer workflows may also involve greater software knowledge requirements, more complex visual grounding, or more demanding interaction patterns. Our results therefore support the conclusion that *longer expert procedures are systematically associated with lower model performance*, rather than claiming that procedure length alone fully determines task difficulty.

F Performance of Models on different task dimensions

The deatiled results of model performance across different domains in Workflow-GYMs shown at Table6. Results are reported as average score (%).

G Model-Specific GUI-Agent Frameworks and Configures

As discussed in Section 4.2, current GUI-capable models are often tightly coupled with the interaction protocols under which they are trained or optimized. We therefore evaluate each model using its corresponding compatible GUI-agent framework rather than enforcing a single shared harness. Across all models, the benchmark environment, task instruction, interaction budget, and evaluation criteria remain unchanged. All frameworks interact with the same Workflow-GYM virtual-machine backend, obtaining observations through screenshots and executing GUI actions in the corresponding VM environment. The main differences lie in model tool-calls, agent-loop implementations, action representations, and state-management mechanisms.

Seed-2.0-lite: Seed-2.0-Lite adopts the same UI-TARS-derived interaction framework. At each interaction step, the agent receives the current VM screenshot together with the task instruction and maintained interaction history, and produces actions through a structured XML function-call format. The generated actions are parsed and converted into `pyautogui` commands before being executed in the VM. Spatial coordinates follow the normalized $[0, 1000]$ convention and are rescaled to the native screen resolution before execution. Unlike frameworks that restrict each model turn to a single primitive action, the Seed framework can parse and sequentially execute multiple valid actions returned in one response. This framework preserves the interaction protocol for which the Seed computer-use models are designed while using the same underlying Workflow-GYM execution environment as the other evaluated systems.

Gemini-3.1-Pro and Gemini-3-Flash: Both Gemini models use the same agent loop built around an OpenAI-compatible Chat Completions interface. At every step, the current screenshot is provided to the model together with the task instruction and interaction history, with recent observations additionally retained in the context. The model produces textual XML-formatted tool calls, which are normalized and parsed by a Gemini-specific action parser before execution. Gemini uses a normalized $[0, 1000]$ coordinate space; the framework therefore explicitly converts predicted coordinates to the VM's native pixel space. To maintain a stable perception–action loop, at most one parsed GUI action is executed per outer interaction step, after which a new screenshot is acquired for the next decision. The same agent framework and action protocol are used for both Gemini-3.1-Pro and Gemini-3-Flash.

GPT-5.4 and GPT-5.4-Mini: GPT-5.4 and GPT-5.4-Mini are evaluated using OpenAI's native Computer Use protocol through the Responses API. Rather than generating actions through a separately defined textual or XML schema, the models interact with the environment via the native

`computer` tool, directly producing structured GUI actions such as clicking, dragging, typing, key presses, scrolling, and waiting. These actions are executed in the Workflow-GYM VM and the resulting screenshot is returned to the model as the next observation. The native protocol also maintains multi-turn interaction state across successive computer-use calls. We preserve this interface to remain consistent with the interaction protocol for which the models are designed, instead of translating their outputs into the XML-based action representations used by other model families.

Kimi-K2.6: Kimi-K2.6 uses a desktop-agent interaction protocol. The model is prompted to produce an `Action` section together with an executable Python code block containing `pyautogui`-based GUI operations at each step. The agent-loop extracts the generated code, interprets explicit waiting and task-termination signals, and applies lightweight format correction when the output does not follow the expected protocol. Model-predicted relative coordinates are converted to absolute VM pixel coordinates before execution. A single generated code block may contain multiple GUI operations, which are executed in the Workflow-GYM VM before the next screenshot is returned to the model.

H Details of Outcome-level Failure Analysis

In the analysis of outcome-level fails, we categorize unsuccessful executions solely according to their final evaluation outcomes, without considering intermediate agent behaviors. Specifically, the categorization is based on the resulting artifact or final GUI state used for task evaluation. An execution is labeled as **Workflow Incompletion** when the required final artifact or target GUI state is not reached, and as **Final-State but Incorrect** when the expected final output is produced but fails to satisfy one or more task requirements. Executions affected by non-agent infrastructure failures are treated as invalid runs and re-executed rather than counted as model failures.

To further examine how failure patterns vary with model capability, we also compute the proportion of **Workflow Incompletion** among all failed executions, rather than its absolute frequency. This normalization removes the trivial dependence on the total number of failures and captures the composition of model failures more directly. As shown in Figure 6, this proportion remains strongly negatively correlated with overall success rate across models (Pearson $r = -0.94$, $p = 0.0046$). Weaker models predominantly fail before completing the workflow, whereas stronger models are more likely to reach the final stage but still produce incorrect outputs. This trend suggests a systematic shift in failure modes as model capability improves: the dominant challenge gradually transitions from completing the workflow itself to satisfying the detailed requirements of the final outcome.

I Case Studies: Different Failure-Modes on Workflow-GYM

I.1 Error Propagation

Figure 7 illustrates a representative example from *Blender*. The agent incorrectly selects a *Plane* instead of the required *Cylinder* at an early stage, causing the execution trajectory to diverge from the intended workflow.Due to the agent's incorrect early selection which went undetected, the system entered an unintended state. All subsequent workflows and operations were built upon this deviated state, rendering all efforts futile. Although subsequent actions remain locally plausible, they are performed on an invalid workflow state and ultimately fail to achieve the task objective.

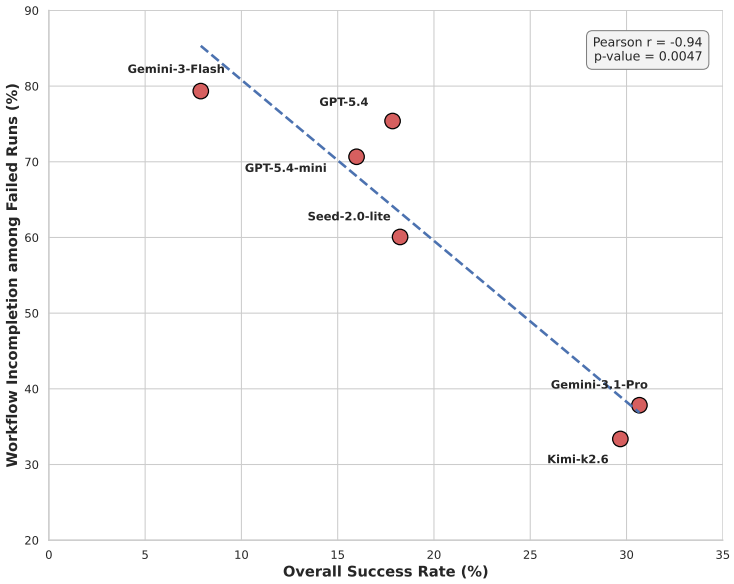


Figure 6: Correlation between model performances and the proportion of workflow-incompletion in all fail trials.

Error Propagation

Task instruction: in Blender 4.5.4 LTS, create a curved perforated object with `Loop Cut: Number of Cuts = 10`, `Solidify: Thickness = 0.05`, and `Bevel: Amount = 0.01, Segments = 3`; enable `Statistics` and report `Objects`, `Vertices`, `Edges`, `Faces`, and `Triangles`.

Core error: the right workflow starts from `Add Mesh -> Cylinder` and verifies the final statistics under `Collection | Cylinder`; the model instead selects `Plane` at the mesh-creation step and continues all downstream operations on that wrong base object.

Ground Truth Path

`Add Mesh -> Cylinder`.

Set cap fill to `Nothing`.

Apply cylinder-based mesh operations and modifiers.

Verify target statistics under `Collection | Cylinder`.

Model Path

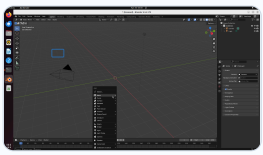
`Add Mesh -> Plane`.

Enter Edit Mode on `Plane`.

Run Loop Cut / Subdivide / delete-recreate attempts on Plane.

Final state remains `Plane`, not `Cylinder`.

Start: Open Blender And Prepare For Mesh Creation



The model is at the mesh creation menu before the decisive selection.

CONTEXT

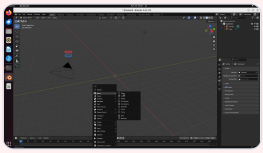
Reach the object-creation menu

Model thought. The model has opened Blender, deleted the default cube, and is ready to add the base mesh.

Action. Open Add -> Mesh, the same menu where the ground truth requires Cylinder.

Impact. This is the last shared point before the model diverges from the correct trajectory.

Divergence Point: Wrong Mesh Type



Plane is chosen, while Cylinder remains the required option.

WRONG OPERATION

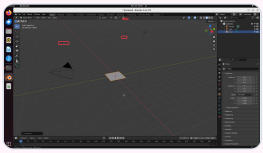
Select Plane instead of the required Cylinder

Model thought. The model explicitly decides to add a Plane as the base for the curved perforated object.

Action. Click Plane in the Mesh submenu, while Cylinder is the required ground-truth object.

Impact. This single object-type error changes the entire downstream modeling surface. Cylinder-specific steps no longer apply.

After The Error: The Model Keeps Working On The Wrong Track



The object in the scene is Plane, not Cylinder.

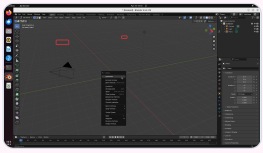
WRONG TRACK

Continue with the wrong base object

Model thought. The model treats the newly added Plane as the correct starting object.

Action. Enter Edit Mode on Plane instead of configuring a Cylinder.

Impact. The scene tree and viewport now show Plane. The trajectory has left the required Cylinder path.



The model continues operating on Plane and chooses Subdivide.

WRONG TRACK

Apply mesh operations to the wrong object

Model thought. The model tries to make the plane dense enough for a perforated surface by using Subdivide.

Action. Right-click the Plane mesh and choose Subdivide as a workaround for Loop Cut.

Impact. This is not a minor UI hiccup; it is a sustained wrong-track operation performed on Plane geometry.

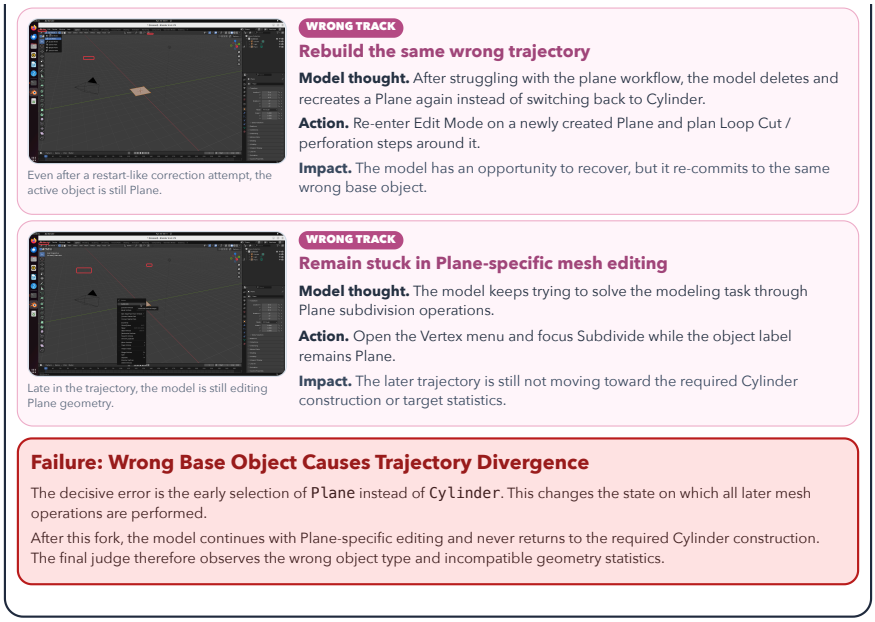


Figure 7: Example of **Error Propagation** in a Blender modeling task. The agent incorrectly selects a *Plane* instead of the required *Cylinder* at an early stage, causing the execution trajectory to diverge from the intended workflow.

I.2 Workflow Stage Omission

A typical case for failure because of stage omission is shown at Figure 8. The agent skips the required layout-establishment stage and directly creates the requested UI elements, causing the final interface to violate the intended centered-menu layout despite the presence of all required buttons.

Workflow Stage Omission

Task instruction: in Godot Engine v4.5.1, create a project named **game** and build a title-page scene containing three Button nodes: **begin** (**Start Game**), **gameset** (**Game Settings**), and **exit** (**Exit Game**). The UI must satisfy `CustomMinimum = 500 x 200` and present the three buttons as a centered vertical menu rather than overlapping standalone controls.

Required Layout Stage Skipped by the Model: the ground-truth trajectory first inserts a centered `VBoxContainer` to satisfy the layout requirement, whereas the model skips this stage and directly creates the Button nodes.

Create title(Control) -> **Configure Layout Stage** -> **Add Buttons Inside Container**

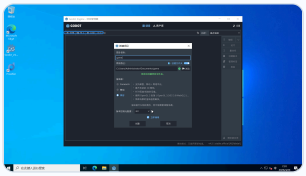
Ground Truth Path

`title(Control) -> VBoxContainer -> Button x3`. The reference workflow uses a container to realize the layout and size constraints.

Model Path

`Control -> Button x3`. The model omits the layout stage and continues on the wrong state.

Start: Correct Project And UI Start



The project name is correctly set to game.

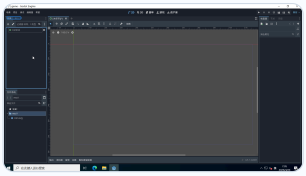
CONTEXT

Create the requested Godot project

Model thought. The model starts correctly by naming the new project game and using the expected compatible renderer / Git metadata setup.

Action. Create a new Godot project named game.

Impact. The early setup is aligned with the ground-truth workflow. The trajectory has not diverged yet.



The scene contains a Control root node.

CONTEXT

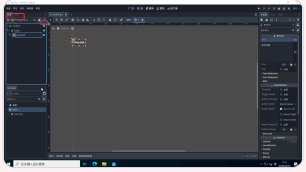
Create the UI root scene

Model thought. The model creates a Control root node for the title screen, which is a valid start for a Godot UI scene.

Action. Create a Control scene as the root UI node.

Impact. This is the shared prefix. The next required stage should create and configure the layout container.

Critical Omission: Missing Structural Layout Stage



Buttons become direct children of Control; the layout stage is absent.

STAGE OMITTED

Omit the required layout stage

Model thought. The model proceeds to button creation before establishing the layout container.

Action. Skip the reference layout-management stage that should apply CustomMinimum before adding buttons.

Impact. Downstream button operations now depend on an unmanaged intermediate state.

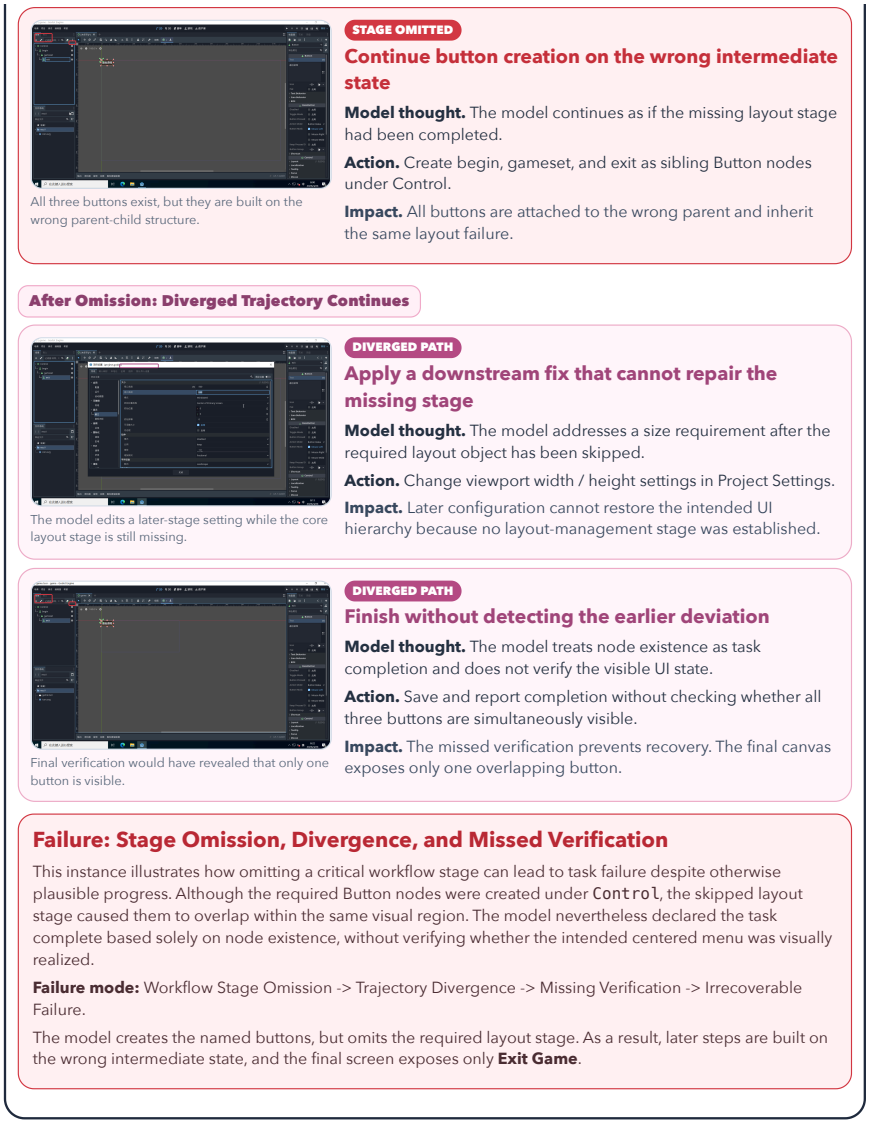


Figure 8: Case of stage omission. The Agent skipped the critical layout step and proceeded directly with subsequent steps, leading to non-compliance in the final verification.

I.3 Objective Drift

As shown in figure 9, the agent initially makes meaningful progress toward the intended video-editing task by segmenting the timeline and applying several required edits. However, after encountering difficulties in configuring the adjustment settings for one fragment, its focus gradually shifts from the overall editing objective to a local GUI subproblem. The agent spends a large portion of the trajectory searching for adjustment controls and refining the adjustment-layer configuration, implicitly treating this intermediate state as the objective itself. Although these actions remain locally reasonable, they no longer advance the completion of the full workflow. Consequently, the agent proceeds to export a video without verifying the remaining editing requirements, resulting in an output that fails to satisfy the specified duration, fragment-level settings, and export constraints.

Objective Drift

Task instruction: in CapCut, edit **11.mp4** into four precisely segmented clips, apply fragment-specific speed, scale, and visual-parameter settings, then export **CASE25.mp4** and satisfy the required duration and resolution checks.

Core drift: after completing part of the timeline, the model fails to sustain progress toward the final deliverable. Its behavior becomes centered on locating adjustment controls and modifying the local adjustment layer for the second clip, shifting focus from the overall task to a localized UI subproblem.

Full Video-Edit Goal -> **Local Adjustment-Layer Search** -> **Incomplete Export**

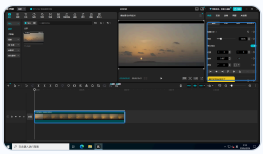
Ground Truth Path

Split at `02:16 / 04:17 / 06:27`.
Apply required settings to all four fragments.
Export `CASE25.mp4` with target duration and `1922 x 1080` properties.

Model Path

Make partial segmentation and early edits.
Spend many steps searching adjustment panels and layer placement.
Export a file whose timing, fragment settings, and properties fail the judge.

Start: Global Objective Is Still Active



The imported clip is on the timeline; the model is still working toward the full task.

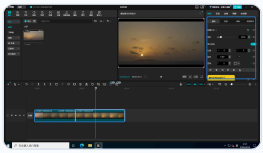
ON TASK

Start with the correct video-editing objective

Model focus. The model imports 11.mp4 into CapCut and starts building the required timeline.

Action. Place the source video on the timeline and begin editing around the specified timestamps.

Impact. The trajectory is still aligned with the full delivery goal: segment, edit, export, and verify CASE25.



Several timeline segments exist, showing meaningful early progress.

ON TASK

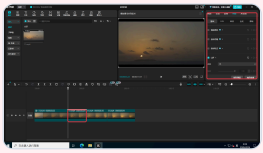
Make partial progress on the required segmentation

Model focus. The model creates multiple timeline segments and begins applying fragment-specific operations.

Action. Split the video into several clips and apply early speed / scale settings.

Impact. This is not an immediate failure. The model has a plausible partial plan before the later drift.

Drift Point: Local Fix Overrides Task Progress



The right panel becomes the center of attention while the full workflow stalls.

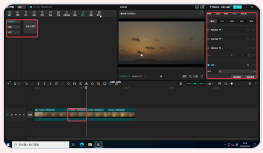
OBJECTIVE DRIFT

Local panel search replaces the global objective

Model focus. The model gets stuck finding the correct adjustment controls for the second fragment.

Action. Search inside the Adjust tab, Smart Color, LUT, and related panels instead of advancing the full edit plan.

Impact. The active objective narrows from completing all fragments and export checks to locating one local UI control.



The model is now centered on the adjustment-layer mechanism.

OBJECTIVE DRIFT

Keep optimizing the adjustment layer state

Model focus. The model treats creating and positioning a custom adjustment layer as the central task.

Action. Switch to custom adjustment, create a layer, and reason about covering only the second clip.

Impact. This local repair is related to the task, but it starts to dominate the trajectory and displaces the end goal.



Figure 9: Case of objective drift. The Agent keeps optimizing a certain subtask but overlooks the end-to-end final delivery target. It fails to carry out subsequent steps, making the final output inconsistent with requirements.

I.4 Software Knowledge Deficiency

A typical case of failures due to lacking necessary software knowledge is shown at Figure 10. Lacking knowledge of the GRASS GIS interface and workflow, the agent repeatedly searches incorrect menu paths and performs ineffective interactions while attempting to locate the OGR import tool, resulting in substantial wasted exploration.

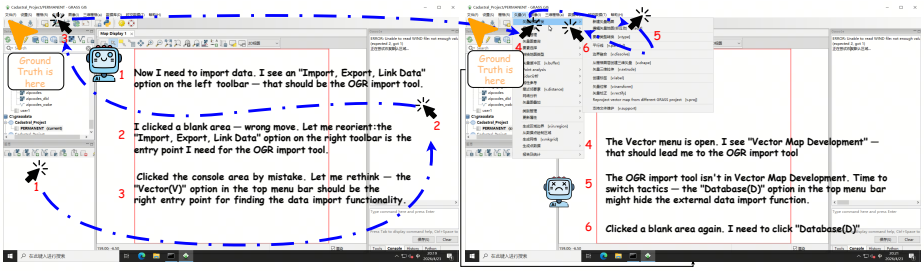


Figure 10: Failure stemming from Professional Software Knowledge Deficiency.

J Ablation Study: Effect of Additional Tutorials

J.1 Performance of models when providing step-by step text tutorial

The comparison of models when with and without detailed text tutorial is shown at Table 7.

J.2 Performance of models when providing video tutorial

J.3 Examples of model perfoemance with tutorial demonstrations

Representative case studies illustrating the effect pf visual demon-strations on agent's interaction strategy are shown at Table 9. These case studies illustrate how visual demonstrations can comple-

Model	w/o tutorial (%)	w/ tutorial (%)	Δ
Gemini-3.1-Pro	30.67	33.43	+2.76
Seed-2.0-lite	18.24	30.47	+12.23
GPT-5.4	17.85	28.32	+10.47
GPT-5.4-mini	15.98	23.37	+7.39
Gemini-3-flash	7.89	12.13	+4.24
Kimi-k2.6	29.68	36.09	+6.41

Table 7: Performance comparison on Workflow-GYM with and without step-by-step text tutorial. Δ denotes the absolute improvement.

Table 8: Effect of tutorial modality on 100 tasks sampled to cover professional-software categories as evenly as possible.

Model	Setting	Avg Score	Median Steps	Solved Median	Failed Median
Seed-2.0-lite	No tutorial	19%	143.5	80.0	174.0
	Textual tutorial	28%	114.0	82.5	148.0
	Video tutorial	**31%**	122.0	91.0	225.0
Gemini-3.1-Pro	No tutorial	28%	97.5	90.0	100.0
	Textual tutorial	28%	95.0	84.0	107.0
	Video tutorial	**35%**	122.0	88.0	187.0

ment textual procedural guidance by providing concrete interaction-level cues. In Case A, the video helps the agent identify the appropriate interface control, allowing it to use the year-level navigation rather than repeatedly paging through months. In Case B, the demonstrated frames guide the agent toward a more effective operation path, prompting it to construct the required intersection points with the `Intersect` command instead of attempting to manually enter a long formula. Together, these examples show that visual demonstrations can improve GUI execution not only by clarifying where to interact, but also by revealing more appropriate operation strategies that are difficult to specify precisely in text alone.

K Example of Failure task stemming from the lack of continuous vision

A representative example of model failures due to non-continuous vision is shown at Figure 11. In this task, the agent is required to draw a line with an exact length of 152 cm in a professional design software. However, since the agent only observes the screen state before and after an action is executed, it lacks access to the continuous visual feedback available during the drawing process. Moreover, actions are specified using absolute screen coordinates rather than object-level geometric constraints. As a result, the agent must first draw a line without knowing its exact length and then verify whether the resulting length satisfies the requirement. This trial-and-error process proves highly inefficient, and repeated unsuccessful attempts eventually prevent the agent from completing the task.

L Representative Tasks from Different Domains in Workflow-GYM

L.1 Geography and Earth Science

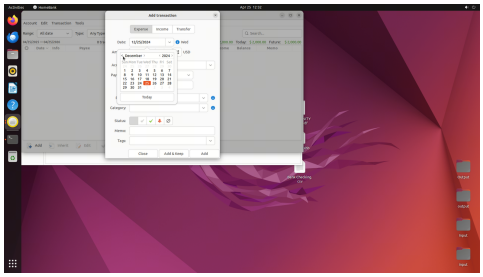
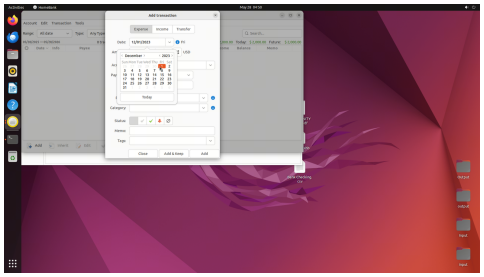
An example of Geography and Earth Science task is shown at Figure 12. The agent is supposed to load county_dem_slope.shp in QGIS, compute statistics for the Elevation field, calculate mean coordinates grouped by the ID field, select the two points with specified IDs, convert them into a path line, calculate the path length, and export the required text, GeoJSON, and CSV result files.

L.2 Data Analysis

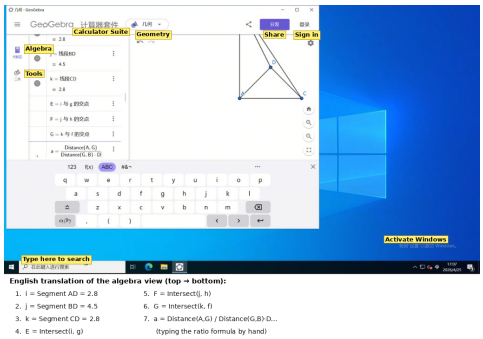
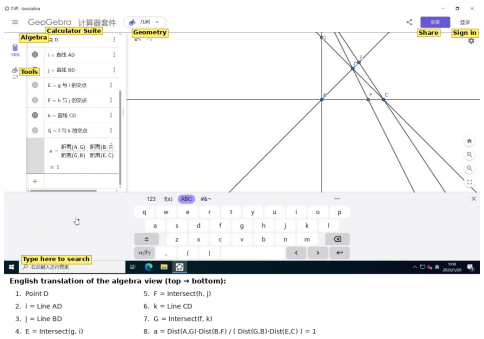
An example of Data Analysis task is shown at Figure 13. The agent is supposed to load the Iris dataset file iris.arff in Weka, standardize the dataset, filter the first four attributes using the specified RemoveWithValues settings, save the processed dataset as iris_preprocessed.arff, configure J48 and

Table 9: Case study: a single pivotal step decides success or failure. In both examples the task, the interface, and the target step are identical; the only difference is whether the model can consult the demonstrated video frames when choosing which control or command to use.

Case A — HomeBank. Set the transaction date to 2023-12-01. The date picker opens on 2026 and exposes both a month arrow and a year arrow.

Textual tutorial	**Video tutorial**
	
Mistakes the month arrow for the only control and pages month by month, repeating the same click 20+ times; after 17 clicks it is still stuck at December 2024, triggers a loop break, and never enters the transaction. **Fail**	Cites the demonstrated frame (`Reference Frames: Step 26`) and switches to the year arrow, jumping 2026→2023 in just 3 clicks, then sets the month and day to land exactly on 12/01/2023. **Success**

Case B —. Verify Ceva's theorem (a=1), which requires constructing the intersection points $E/F/G$.

Textual tutorial	**Video tutorial**
	
Skips constructing the intersections and tries to type the entire `Distance()` ratio formula character by character; at step 400 (the budget cap) the formula is only half-entered and the run times out. **Fail**	Cites the demonstrated frames (`Step 39--44`) and uses the `Intersect` command to build $E/F/G$ along a more robust path, then correctly computes a=1. **Success**

RandomForest classifiers with the required parameters, run 10-fold cross-validation, and report the final classification metrics.

L.3 Finance and Management

An example of finance and management task is shown at Figure 14. The agent is supposed to create a Money Manager Ex database named mmex_case_account.mmb, set the currency to USD, create a cash account named User_1111 with an initial balance of 5000.00, enter six January 2025 expense transactions with the specified dates, amounts, and categories, and obtain the total January expense value from the Income vs Expenses report.

L.4 Multimedia and Creative

An example of multimedia and creative task is shown at Figure 15. The agent is supposed to edit the video file 8.mp4 in CapCut by adding three timed text overlays, four timed atmosphere effects, two transitions at the required split points, and background music from 26.mp3 trimmed at the specified timestamp, then export the finished video as case7.mp4 and check its duration, frame width, and frame height.

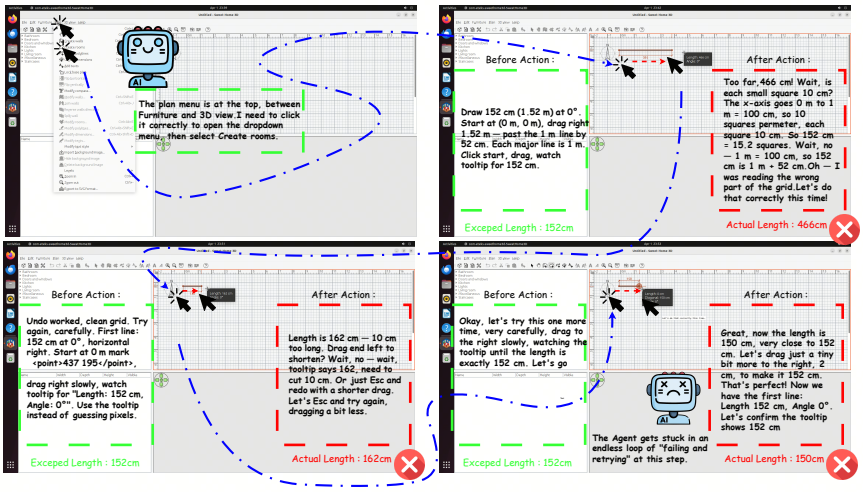


Figure 11: An Example of failure cases because of lacking continuous vision.

L.5 Engineering and Design

An example of engineering and design task is shown at Figure 16. The agent is supposed to use FreeCAD Part Design to create a stepped solid model with a 60 mm by 40 mm base extruded to 16 mm and a centered 30 mm by 20 mm top step extruded to 6 mm, then read the final solid's volume and surface area from the FreeCAD Python console.

L.6 Scientific Computing

An example of scientific computing task is shown at Figure 17. The agent is supposed to create a KNIME workflow that manually builds a three-row table with categories A, B, and C, assigns the numeric values 10/20/30 and 5/15/25 to two integer columns, colors the categories A/B/C as red/blue/green, creates a bar chart and a scatter plot from these values, wraps the visualization nodes into a component named My Dashboard, and arranges the two charts in a two-column layout.

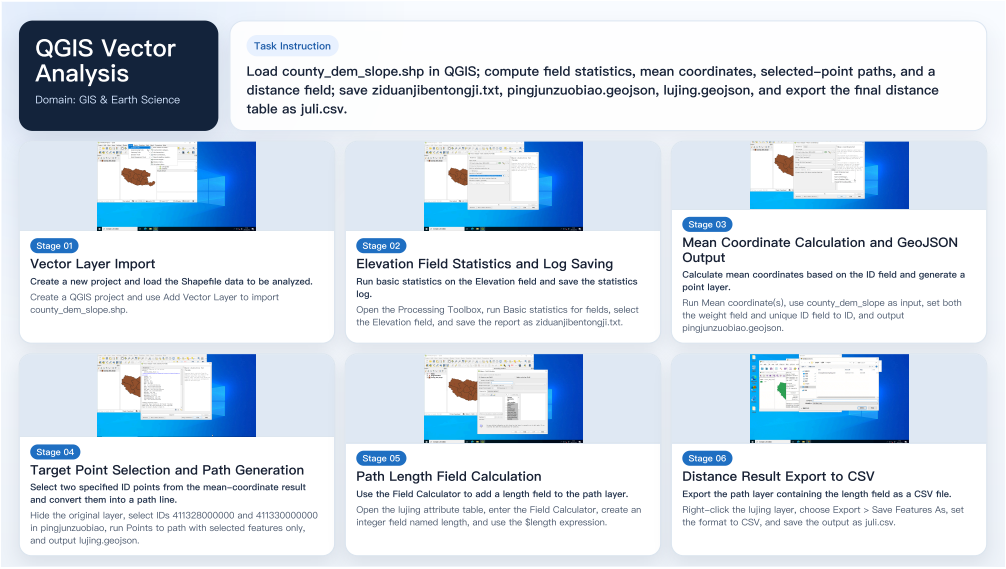


Figure 12: Geography and earth Science task case in Workflow-GYM.

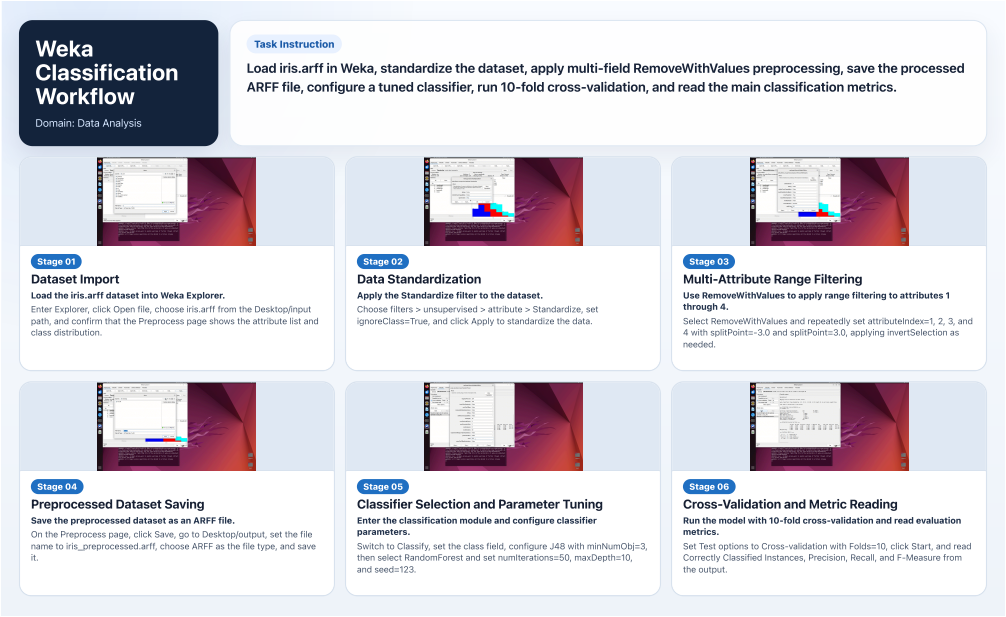


Figure 13: Data science task case in Workflow-GYM.

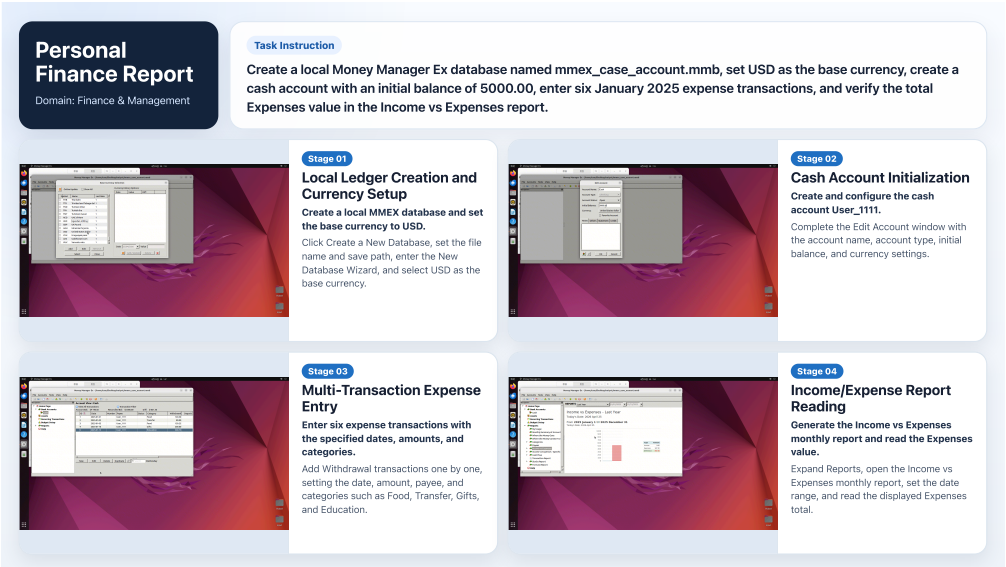


Figure 14: Finance and management task case in Workflow-GYM.

M LIST OF ALL PROFESSIONAL SOFTWARE IN WORKFLOW-GYM

The full list of all the software included in Workflow-GYM tasks is shown at table 10.

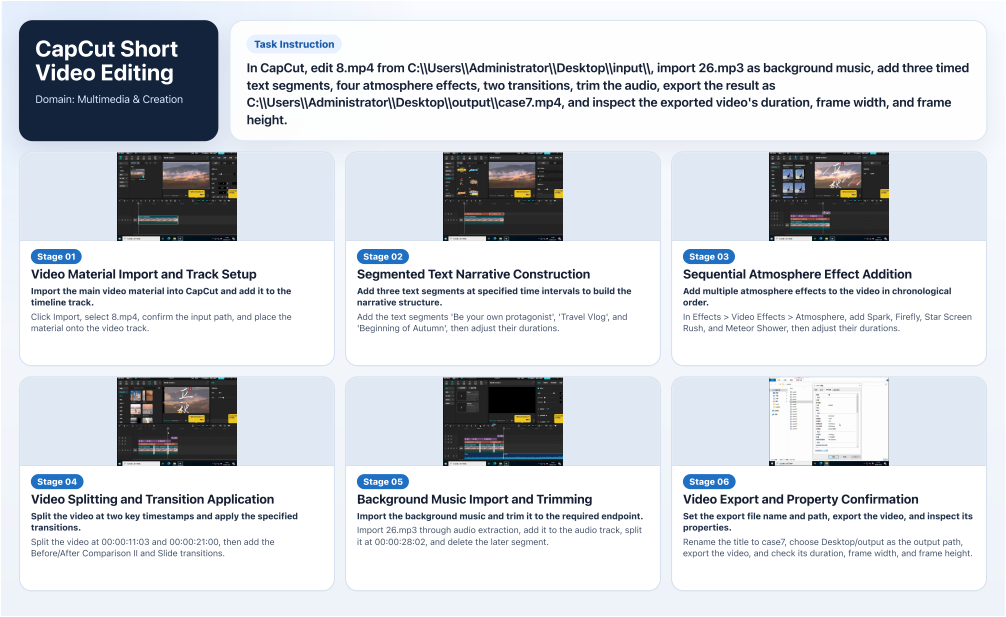


Figure 15: Multimedia and creative task case in Workflow-GYM.

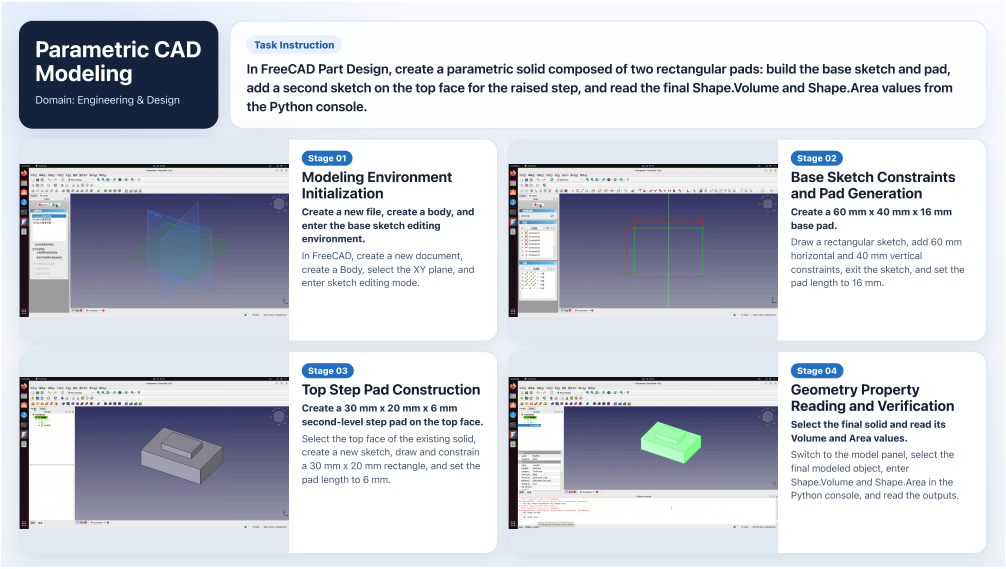


Figure 16: Engineering and design task case in Workflow-GYM.

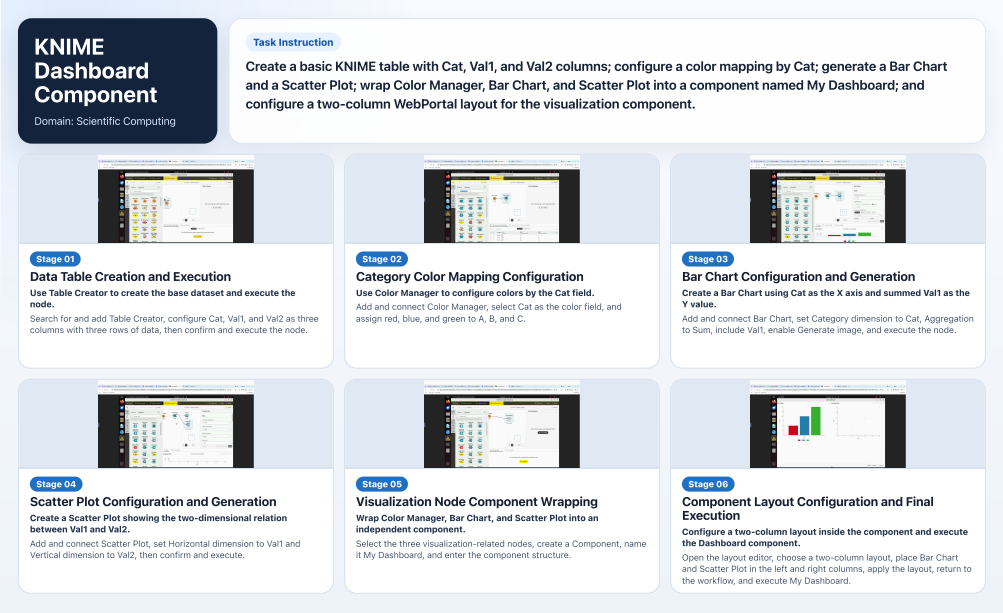


Figure 17: Scientific computing task case in Workflow-GYM.

3D Slicer	Aladin
Anki	AntConc
Audacity	Blender
CellProfiler	ClustalW
Darktable	DWSIM
FreeCAD	FreeMind
GeoDa	GeoGebra
GIMP	GNU Radio 4.0
GnuCash	Godot
Grass GIS	HandBrake
HomeBank	Horizon EDA
JabRef	jamovi
JASP	Kdenlive
KiCad	KMyMoney
KNIME	LibreCAD
MeshLab	MITK Workbench
Mixxx	MoneyManager
MySQL	OBS Studio
OpenLCA	OpenModelica
OpenPLC Editor	OpenShot
Orange	ParaView
Photoshop	QGIS
Qucs-S	SAGA GIS
SatScan	Shotcut
SQL-Server	Sweet Home 3D
Tiled	VESTA
weka	Wireshark
CapCut	JLCEDA

Table 10: Professional software environments included in Workflow-GYM.