CODA: COORDINATING THE CEREBRUM AND CEREBELLUM FOR A DUAL-BRAIN COMPUTER-USE AGENT WITH DECOUPLED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomous agents for Graphical User Interfaces (GUIs) face significant challenges in novel software, requiring both long-horizon planning with software domain knowledge and precise, fine-grained execution. Existing approaches suffer from a trade-off: generalist agents excel at planning but falter in execution, while specialized agents show the opposite weakness. Recent compositional frameworks attempt to bridge this gap by combining a "planner" and an "actor", but they are typically static and non-trainable, preventing adaptation from experience—a critical limitation given the scarcity of high-quality data in novel software. To address these limitations, we introduce **CODA**, a novel and trainable compositional framework that synergizes a generalist planner (Cerebrum) with a specialist executor (Cerebellum), trained with a dedicated two-stage training pipeline. The first stage, **Specialization**, employs a decoupled GRPO approach to train an expert planner for each novel software individually. The second stage, **Generalization**, aggregates all positive trajectories from all specialized experts. This consolidated, high-quality dataset is then used to perform supervised fine-tuning (SFT) on the final planner, equipping it with the robust, cross-domain capabilities of a generalist. Evaluated on the ScienceBoard benchmark with diversified novel software, our framework significantly outperforms the baseline and establishes a new state-of-the-art (SOTA) among open-source models, with strong generalizability to novel software and unseen executors such as code agents. All code and models will be made publicly available to foster further research.

1 INTRODUCTION

Autonomous agents for Graphical User Interfaces (GUIs) (Anthropic, 2024; OpenAI, 2025; Qin et al., 2025; Lin et al., 2024; Wu et al., 2024b; Hong et al., 2023) promise to automate a wide range of digital tasks (Zhou et al., 2023; Xie et al., 2024). However, their application in specialized domains such as scientific computing and engineering analysis remains highly challenging (Sun et al., 2025a). These environments pose two primary difficulties: first, their interfaces are highly complex, requiring precise and fine-grained actions; second, the problems they address are intrinsically complicated, demanding long-horizon planning to achieve effective solutions.

Effective agency for computer task automation in these domains requires both high-level planning and low-level execution as well as domain knowledge. However, current models exhibit a clear trade-off. Generalist models like Qwen2.5-VL (Bai et al., 2025) provide robust planning capabilities but often struggle with the precise grounding needed for reliable execution. Conversely, specialized agents (Wu et al., 2024b; 2025; Xie et al., 2025) like UI-Tars (Qin et al., 2025) are highly proficient in execution, yet lack of domain specific knowledge for novel knowledge and require human labeled trajectories on these softwares for fine-tuning.

To bridge this gap, a natural approach has been to develop compositional frameworks that explicitly decouple planning from execution, effectively pairing a generalist "cerebrum" with a specialist "cerebellum" (Agashe et al., 2024; 2025; Song et al., 2025). While promising, these pioneering approaches are fundamentally limited. They are typically static and non-trainable, relying on powerful, often closed-source models as their core planner. This design introduces significant drawbacks: it

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.

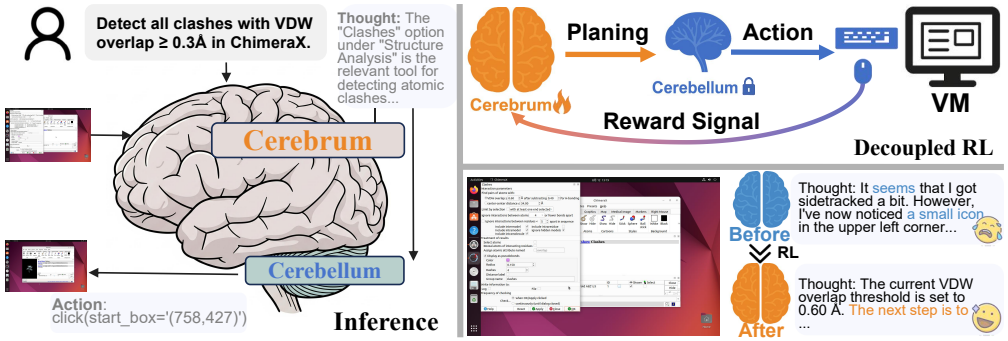


Figure 1: **Overall architecture of the proposed learnable Planner–Executor framework.** Analogous to the relationship between the cerebrum and the cerebellum in the human brain, the Planner (cerebrum) generates high-level thoughts based on the history and screenshots, while the Executor (cerebellum) executes concrete GUI actions accordingly.

compromises transparency and replicability, and most critically, prevents the agent from learning and adapting through experience on novel softwares.

This architectural decoupling is not merely an engineering convenience but is inspired by the functional architecture of the human brain (illustrated in Fig. 1). The specialization of high-level planning (the Cerebrum) and low-level motor control (the Cerebellum) is a key aspect of human intelligence. Crucially, these structures exhibit different learning patterns: the Cerebellum, once mature, provides stable and broadly applicable motor skills that require infrequent updates (Ito, 2000). In contrast, the Cerebrum continuously adapts its strategies based on the nuances of new tasks and environments (Demarin & Morović, 2014; Hallett, 2005). This biological parallel motivates our core hypothesis: an effective agent should pair a stable, proficient grounding model with a dynamic planner that is specialized for different software domains through targeted, experience-driven learning.

To realize this vision, we propose a trainable compositional framework that integrates Qwen2.5-VL (Bai et al., 2025) as the planner (cerebrum) and UI-Tars-1.5 (Qin et al., 2025) or GUI-Actor (Wu et al., 2025) as the executor (cerebellum). Unlike prompting-based systems that rely on proprietary closed-source planners, our framework makes the planner itself learnable through interaction with software environments mediated by a static executor. Concretely, the executor provides stable, software-agnostic grounding for low-level GUI actions, while the planner, by leveraging this reliable interface, can gradually acquire domain-specific knowledge and improve its high-level planning strategies. In contrast to end-to-end training of a single agent, which requires massive amounts of specialized data and costly retraining of both perception and execution modules, our decoupled approach is substantially more data efficient: only the planner is optimized for domain adaptation, while the executor remains fixed as a general-purpose grounder that already possesses strong generalization ability after massive pretraining for grounding purposes. This design reduces reliance on curated trajectories, lowers training cost, and ensures controllable adaptation.

To train the planner effectively under this cerebrum–cerebellum separation, we avoid the need for costly human-labeled trajectories *at the agent-training stage*. Instead, we leverage a judging system built from open-source models to automatically provide dense reward signals, combined with autonomous interaction with scientific software environments through the static executor. This setup enables the planner to gradually acquire domain-specific planning ability without per-domain trajectory annotation; we note that human supervision still enters the broader pipeline indirectly through the judge's distillation labels and benchmark ground truth, and we discuss this scope honestly in App. A. Furthermore, by distributing the interaction process across multiple software environments in parallel—coordinated by a central master—we can significantly accelerate reinforcement learning. This strategy not only makes the training process more efficient but also echoes our brain-inspired design: the cerebellum-like executor delivers stable grounding, while the cerebrum-like planner continually adapts through experience.

Contributions. Our work makes three major contributions:

- We propose a decoupled reinforcement learning framework that pairs a learnable planner with a fixed executor, enabling efficient domain adaptation by isolating strategic learning from low-level grounding and control.
- We propose a fully automated judging system using a fine-tuned model and voting strategy to generate dense reward signals, which removes the dependency on human-annotated data for training. This judging system achieves strong performance on AgentRewardBench (Lù et al., 2025) and other software environments.
- Planner trained via our framework based on Qwen2.5-VL (Bai et al., 2025) achieves strong performance on the ScienceBoard (Sun et al., 2025a) benchmark with generalizable adaption on OSWorld (Xie et al., 2024) and novel executors such as code agents.

2 Related Work

Reinforcement Learning for LVLMs (Large Vision Language Models). Training for LLMs (Large Language Models) and LVLMs (Touvron et al., 2023; Grattafiori et al., 2024; Liu et al., 2023a; Bai et al., 2025; Wang et al., 2024; Xing et al., 2025; Sun et al., 2024d;c; Ding et al., 2025) has progressed from data-intensive Supervised Fine-Tuning (SFT) (Liu et al., 2023a; Wei et al., 2022) towards Reinforcement Learning (RL). Algorithms like Group Relative Policy Optimization (GRPO) (Guo et al., 2025; Shao et al., 2024) have proven effective for reasoning tasks, moving beyond earlier single-turn RLHF applications (Ouyang et al., 2022; Ziegler et al., 2019; Rafailov et al., 2023). However, applying RL to complex agentic tasks (Bai et al., 2024; Qi et al., 2024; Zhou et al., 2024; Zhai et al., 2024; Carta et al., 2023) is challenging. Prevailing methods train monolithic agents end-to-end, often requiring co-trained critic models (Schulman et al., 2015) or preference-based optimization like DPO (Rafailov et al., 2023; Putta et al., 2024; Qin et al., 2025), which problematically entangles the distinct skills of planning and execution. In contrast, our work employs a decoupled reinforcement learning strategy: the high-level planner is optimized via environmental interaction while the execution model remains fixed. We adapt GRPO by computing rewards from the reward system and backpropagating the advantage exclusively through planning tokens. This targeted optimization stably enhances strategic planning, distinguishing our method from prior works that train dedicated critic models (Bai et al., 2024; Qi et al., 2024; Wang et al., 2025) or use filtered behavior cloning (Pan et al., 2024; Chen et al., 2020).

Computer Use Agent. Fueled by advancements in Large Vision-Language Models (LVLMs) (Touvron et al., 2023; Grattafiori et al., 2024; Liu et al., 2023a; Bai et al., 2025; Wang et al., 2024), a new generation of agents capable of operating computers via multi-modal inputs is emerging (Hu et al., 2024b; Hong et al., 2024; Cheng et al., 2024; Nguyen et al., 2024; Lin et al., 2024; Sun et al., 2024b). Whether processing structured text and code (Qi et al., 2024; Putta et al., 2024; Lai et al., 2024; Sun et al., 2024a; Nakano et al., 2021) or screenshots (Hong et al., 2023; Lin et al., 2024; Wu et al., 2024b; OpenAI, 2025), these agents face an inherent dichotomy analogous to human cognition: the tension between high-level strategic planning and precise, low-level action execution. This has motivated the development of compositional frameworks that decouple these responsibilities (Agashe et al., 2024; 2025; Liu et al., 2023b; Zhang et al., 2025; Song et al., 2025). However, a significant portion of this research relies on static, non-trainable systems that orchestrate powerful, often proprietary models (Anthropic, 2024; OpenAI, 2025; Google DeepMind, 2025; Yan et al., 2023; He et al., 2024; Zhang et al., 2024; Wang et al., 2023; Wu et al., 2024a) as their core planner. This design fundamentally prevents the agent from adapting through experience—a critical flaw for mastering novel software where interaction data is scarce. Our work charts a different course by exploring reinforcement fine-tuning of the planner. By enabling the planner to learn specialized domain knowledge through direct software interaction via a fixed execution model, our strategy achieves robust performance on unfamiliar applications.

3 Method

3.1 Problem Formulation

We formally define the task of autonomous GUI operation for software workflows as a Partially Observable Markov Decision Process (POMDP) $\langle \mathcal{S}, \mathcal{A}, \Omega, T, O, R, \gamma \rangle$. Each task is initiated with

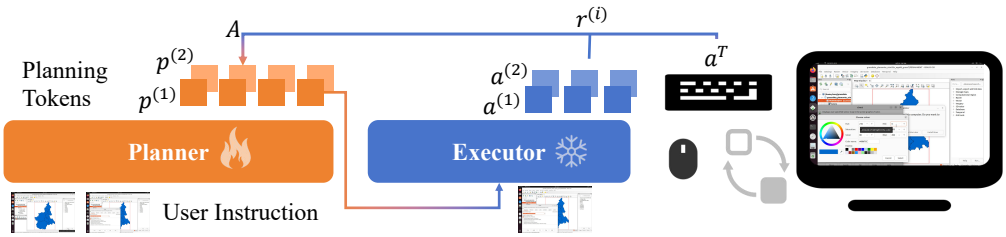


Figure 2: **Overall training process of the proposed Planner–Executor framework.** The Planner generates high-level thoughts based on the history and screenshots, while the Executor executes concrete GUI actions accordingly. During training, the rewards are computed from $a^{(i)}$ and applied to $p^{(i)}$ to compute the loss.

a natural language instruction g from the task space $\mathcal{G}$. At each timestep t, the agent does *not* have direct access to the latent environment state $s_t \in \mathcal{S}$ (which would include hidden application state, off-screen buffers, internal variables, etc.); it only perceives a visual observation $o_t \in \Omega$ via the screenshot operator $O(\cdot \mid s_t)$. The action space $\mathcal{A}$ consists of precisely parameterized `pyautogui` scripts, where precision in arguments (e.g., coordinates) is critical for execution.

Because o_t alone is generally insufficient to recover s_t (the screenshot captures only the visible window), the agent's policy π, instantiated by a large vision-language model, conditions on the full history of past observations and actions to approximate the belief over s_t:

$$a_t = \pi(g, (o_1, a_1, \ldots, a_{t-1}, o_t)) .$$

History dependence is therefore a consequence of, not a contradiction to, the partial-observability assumption: in a fully-observable MDP a Markov policy $\pi(a_t \mid s_t)$ would suffice; here it does not, which is precisely why the planner is given the past trajectory. This sequential process induces a state trajectory $\tau = (s_0, s_1, \ldots, s_T)$ with maximum time step T. A task is considered successful if the final state s_T satisfies the predefined goal condition specified in $\mathcal{G}$.

3.2 Model Architecture

To address the inherent trade-off in monolithic models, which struggle to balance long-horizon planning with precise action grounding, we propose a composite agent architecture that structures the decision-making process into a Planner–Executor framework. This design decouples the task into two distinct yet collaborative modules: a high-level Planner responsible for strategic thinking and a low-level Executor for concrete action execution.

Planner. The Planner is instantiated from the Qwen2.5-VL (Bai et al., 2025) model. Its primary responsibility is to analyze the task's progress and formulate a high-level, explicit plan p_t for each step. Specifically, at each timestep t, the Planner receives the interaction history up to the previous step $m_{t-1} = (p_1, a_1, \ldots, p_{t-1}, a_{t-1})$, the current visual observation o_t, and the preceding observation o_{t-1}. The output is a structured thought, denoted as p_t, which outlines the immediate objective and explicitly identifies the target UI elements for interaction. The process can be summarized as:

$$p_t = \text{Planner}(m_{t-1}, o_{t-1}, o_t).$$

Executor. The Executor employs a specialized GUI grounding model (UI-TARS-1.5 (Qin et al., 2025) or GUI-Actor (Wu et al., 2025)). Its role is to translate the Planner's abstract thought p_t into a precise, executable action. The Executor is provided with the same historical and visual context as the Planner (m_{t-1}, o_{t-1}, and o_t), but is critically augmented with the Planner's newly generated thought p_t. Its output is a low-level GUI action a_t in the form of a `pyautogui` command, such as `click(x, y)`. The Executor's operation is defined as:

$$a_t = \text{Executor}(m_{t-1}, o_{t-1}, o_t, p_t).$$

3.3 Training Pipeline

Our training methodology employs a two-stage curriculum designed for initial specialization followed by broad generalization.

3.3.1 STAGE 1: SPECIALIZATION VIA DECOUPLED REINFORCEMENT LEARNING

The primary objective of this initial training stage is to enhance the agent's specialized performance on individual software applications.

Through empirical analysis, we observed that the Executor exhibits strong generalization capabilities, accurately translating well-structured plans into executable actions. However, the Planner module emerged as the primary bottleneck, often struggling to formulate effective high-level strategies. To address this, we adopt a decoupled training strategy that focuses reinforcement learning exclusively on the Planner (π_θ = Planner). This targeted approach allows us to refine the agent's strategic reasoning without altering the already competent Executor.

Since the initial Planner is relatively weak and generates a limited number of successful trajectories, standard reinforcement learning methods can be inefficient. Therefore, we adapt the Group Relative Policy Optimization (GRPO) framework (Guo et al., 2025; Shao et al., 2024), which is particularly effective in such scenarios. GRPO can derive a meaningful learning signal by comparing the relative quality of different outputs, even when most of them are suboptimal.

The training process for a given task unfolds as follows. Given the current state and interaction history, the Planner first generates a group of G candidate plans. Subsequently, the fixed Executor takes each plan as input and produces a corresponding low-level action. To generate a fine-grained learning signal, we compute a reward for each plan by comparing its resulting action $a^{(i)}$ to the labeled positive action a_T (details of the labeling process are in Sec. 3.4). Our composite reward function assesses both the correctness of the action type and the precision of its parameters:

$$r^{(i)} = r(a^{(i)}, a_T) = \mathbb{I}\Big(\text{type}(a^{(i)}) = \text{type}(a_T)\Big) + r_{\text{dist}}(a^{(i)}, a_T), \tag{1}$$

where the indicator function $\mathbb{I}(\cdot)$ provides a binary reward for selecting the correct type of action (e.g., `click` vs. `type`). The term $r_{\text{dist}}(a^{(i)}, a_T)$ offers a continuous reward based on the parametric similarity between the predicted and ground-truth actions, such as L_1 distance for coordinates or IoU for bounding boxes. These distance-based rewards are normalized to $[0, 1]$ to ensure consistent scaling.

Once the rewards are computed, they are used to derive a relative advantage $A^{(i)}$ for each plan, which is then fed into the GRPO loss to update the Planner policy:

$$A^{(i)} = \frac{r^{(i)} - \text{mean}(\{r^{(j)}\}_{j=1}^{G})}{\text{std}(\{r^{(j)}\}_{j=1}^{G})}, \quad i = 1, \cdots, G. \tag{2}$$

The GRPO loss is formulated as

$$\mathcal{L}_{\text{GRPO}}(\pi_\theta) = -\mathbb{E}_{(s,I)\sim\mathcal{D},\, \{a^{(i)}\}_{i=1}^{G}\sim\pi_{\text{ref}}(\cdot|s,I)}$$
$$\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|p^{(i)}|}\sum_{t=1}^{|p^{(i)}|}\left\{\min\Big(r_t^{(i)}(\theta)A^{(i)},\ \text{clip}\big(r_t^{(i)}(\theta), 1-\epsilon, 1+\epsilon\big)A^{(i)}\Big) - \beta\, D_{\text{KL}}^{(i,t)}(\pi_\theta \,\|\, \pi_{\text{ref}})\right\}\right], \tag{3}$$

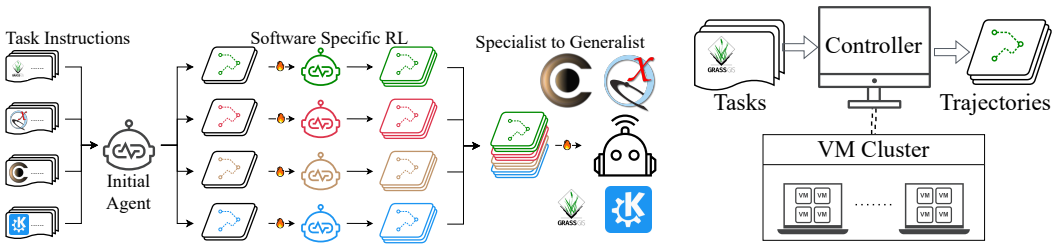
where

$$r^{i,t}(\theta) = \frac{\pi_\theta(p^{(i)} \mid s, I)}{\pi_{\theta_{\text{ref}}}(p^{(i)} \mid s, I)}, \quad D_{\text{KL}}^{i,t}(\pi_\theta, \pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(p^{(i)} \mid s, I)}{\pi_\theta(p^{(i)} \mid s, I)} - 1 - \log\frac{\pi_{\text{ref}}(p^{(i)} \mid s, I)}{\pi_\theta(p^{(i)} \mid s, I)}.$$

Consistent with (Shao et al., 2024; Guo et al., 2025), this advantage is applied across all reasoning tokens in the plan $p^{(i)}$, encouraging the model to develop more robust and free-form planning capabilities.

3.3.2 STAGE 2: GENERALIZATION VIA AGGREGATED SUPERVISED FINE-TUNING

We adopt the specialist-to-generalist paradigm proposed in (Sun et al., 2025b), where a generalist model is trained by leveraging multiple specialist models as teachers. We observe that directly applying reinforcement learning across all software leads to suboptimal performance. To address this, we first train four specialist models using the methods described in Sec. 3.3.1. These specialists are



(a) Specialist-to-Generalist strategy. (b) Distributed VM System.

Figure 3: Specialist-to-generalist as shown in (a) and distributed virtual machine system in (b).

then employed to generate new trajectories for each software, which serve as supervision for training a generalist model. After learning from the four software-specific teachers, the resulting generalist not only surpasses its teachers in performance, but also demonstrates stronger reasoning and reflection abilities during planning, as well as broader domain knowledge across different software.

3.4 Auto Exploration Pipeline

Auto Task Generation. To generate tasks, we employ Qwen2.5-72B (Wang et al., 2024) to produce a diverse set of high-level objectives, with the detailed prompt provided in Fig. 6. Our training methodology proceeds in two stages. The first stage is decoupled reinforcement learning, where the agent repeatedly interacts with software environments to execute the generated tasks. During this process, our judge system provides the necessary reward signals to guide the agent's learning. In the second stage, aggregated fine-tuning, we leverage the specialized agent from the first stage to collect a diverse dataset of interaction trajectories. These trajectories are filtered by the judge system to retain only those containing positive outcomes, which are then used for supervised fine-tuning.

Judge System for Providing Reward Signals. Our judge system labels the positive actions a_T within an agent's trajectory when performing a task. Given a full trajectory $\mathcal{H} = \{o_0, a_0, \ldots, o_{\text{final}}\}$, the judge takes the complete sequence of screenshot observations $(o_1, o_2, \ldots, o_n)$ as input and outputs three signals: `Correctness`, `Redundant`, and `First Error Step`, using the detailed prompt shown in Fig. 7. A trajectory is considered clean and successful when `Correctness` is `True` and both `Redundant` and `First Error Step` are empty. In this case, all actions a in the trajectory are labeled as a_T. We present a detailed evaluation of the judge's precision and discuss approaches for improving it via multi-resolution voting and model ensembling in Sec. 4.2.

Distributed Virtual Machine System. Task execution is the most time-consuming step in our pipeline, so we developed a lightweight distributed system to accelerate large-scale parallel interaction and trajectory curation. As illustrated in Fig. 3b, the system follows an HTTP-based master–client architecture: the master node manages a dynamic task queue, monitors execution progress, and aggregates results, while multiple client nodes execute tasks in parallel within isolated virtual machine environments. This design enables efficient scaling to hundreds of concurrent environments, substantially reducing the time required to collect successful trajectories and making the framework well-suited for large-scale training and evaluation.

4 Experiments

4.1 Agent Performance Evaluation

Our planner–executor approach uses Qwen2.5-VL-32B (Bai et al., 2025) as planner and UI-TARS-1.5-7B (Qin et al., 2025) or GUI-Actor (Wu et al., 2025) as executor. We use the method in Sec. 3.4 to generate high-level tasks for each software from ScienceBoard (Sun et al., 2025a) and train via decoupled RL (Sec. 3.3.1); reward signals come from the judge system evaluated in Sec. 4.2. Training runs for 500 iterations on 32×A100 80 GB GPUs with batch size 16 and learning rate 2×10^{-5} (cosine decay) on OpenRLHF (Hu et al., 2024a); instruction statistics are in App. E. We evaluate on the four GUI-centric ScienceBoard domains (Algebra, Biochem, GIS, Astron); we exclude the Lean domain because it requires formal-mathematics reasoning rather than visual GUI grounding—even

Table 1: Success rates on ScienceBoard (Sun et al., 2025a). Proprietary and open-source models are highlighted; * marks software-specialist agents (Stage-1) with ensembled results. **Avg@1** reproduces single-run numbers from the official ScienceBoard leaderboard. **Avg@8 / Pass@8** are the average / at-least-one success rate over 8 independent rollouts per task. "Gemini-2.5-Pro" refers to Gemini 2.5 Pro (Google DeepMind, 2025).

Metric	Model	Success Rate (↑)				
		Algebra	**Biochem**	**GIS**	**Astron**	**Overall**
Average@1	GPT-4o (OpenAI, 2023)	3.23%	0.00%	0.00%	0.00%	0.81%
	Claude-3.7-Sonnet (Anthropic, 2025)	9.67%	37.93%	2.94%	6.06%	14.15%
	Gemini-2.0-Flash (Team et al., 2023)	6.45%	3.45%	2.94%	6.06%	4.73%
	GPT4o → UGround-V1-7B (Gou et al., 2024)	0.00%	3.45%	0.00%	3.03%	1.62%
	GPT4o → OS-Atlas-Pro-7B (Wu et al., 2024b)	6.25%	10.34%	0.00%	3.03%	4.92%
	GPT4o → UI-TARS-72B (Qin et al., 2025)	3.23%	10.34%	5.88%	6.06%	6.38%
	Qwen2.5-VL-72B (Bai et al., 2025)	22.58%	27.59%	5.88%	9.09%	12.94%
	InternVL3-78B (Zhu et al., 2025)	6.45%	3.45%	0.00%	0.00%	2.69%
	UI-TARS-1.5-7B (Qin et al., 2025)	12.90%	13.79%	0.00%	6.06%	8.19%
Average@8	Qwen2.5-VL-32B	10.48%	13.79%	1.47%	4.55%	7.57%
	UI-TARS-1.5-7B	6.49%	10.24%	0.80%	3.03%	5.14%
	GPT-4o → UI-TARS-1.5-7B	18.14%	30.17%	9.56%	7.58%	16.36%
	Gemini-2.5-Pro → UI-TARS-1.5-7B	16.53%	29.31%	11.76%	8.33%	16.48%
	CODA +Actor (Stage-1)*	12.90%	30.17%	10.66%	7.58%	15.33%
	CODA +Actor (Stage-2)	18.55%	31.03%	**16.54%**	11.74%	19.47%
	CODA +TARS-1.5 (Stage-1)*	13.71%	26.29%	7.72%	9.85%	14.39%
	CODA +TARS-1.5 (Stage-2)	**20.16%**	**32.23%**	14.71%	**17.05%**	**21.04%**
Pass@8	Qwen2.5-VL-32B	29.03%	31.03%	8.82%	9.09%	19.49%
	UI-TARS-1.5-7B	19.35%	24.14%	5.88%	12.12%	15.36%
	GPT-4o → UI-TARS-1.5-7B	38.71%	51.72%	17.65%	15.15%	30.81%
	Gemini-2.5-Pro → UI-TARS-1.5-7B	41.94%	48.28%	23.53%	18.18%	32.98%
	Agent-S2 (Agashe et al., 2025)	41.94%	48.28%	20.59%	24.24%	33.76%
	Agent-S3 (Gonzalez-Pumariega et al., 2025)	45.16%	**55.17%**	23.53%	27.27%	37.78%
	CODA +Actor (Stage-1)*	32.25%	34.48%	26.47%	21.21%	28.61%
	CODA +Actor (Stage-2)	38.71%	37.93%	**32.35%**	24.24%	33.31%
	CODA +TARS-1.5 (Stage-1)*	41.94%	44.83%	23.53%	18.18%	32.12%
	CODA +TARS-1.5 (Stage-2)	**48.39%**	51.72%	29.41%	**30.30%**	**39.96%**

Gemini-2.5-Pro and GPT-4o score 0% on it from screenshots alone—and we report TeXstudio (cross-application) separately in Tab. 2 as it intrinsically requires multi-software workflows.

Stage-1 (specialization). As shown in Tab. 1 (CODA Stage-1), first-stage decoupled RL yields significant gains over the open-source baselines. To further substantiate the SOTA claim against proprietary planners, we add a stronger "gold-standard" baseline that pairs the same UI-TARS-1.5-7B executor with GPT-4o or Gemini-2.5-Pro as the planner. Even at the per-software specialist stage, our 32B planner already matches these proprietary 1T-class planners.

Stage-2 (generalization). We consolidate the four specialist planners into a single generalist via the specialist-to-generalist fine-tuning method (Sec. 3.3.2). The specialists generate 5.6K multi-turn interaction trajectories, filtered by our judge system (App. E). A new generalist planner, also initialized from Qwen2.5-VL-32B, is fine-tuned for 500 iterations on 32×A100 with batch size 64 and learning rate 1×10^{-5}. CODA Stage-2 surpasses the ensemble of individual specialists and, more importantly, outperforms both proprietary planners (GPT-4o, Gemini-2.5-Pro) paired with the same executor and the latest agentic frameworks Agent-S2 (Agashe et al., 2025) and Agent-S3 (Gonzalez-Pumariega et al., 2025) on Avg@8 / Pass@8. In contrast, direct monolithic RL across all four environments with an equivalent budget performs worse than the specialist ensemble, underscoring the superiority of our two-stage decoupled paradigm.

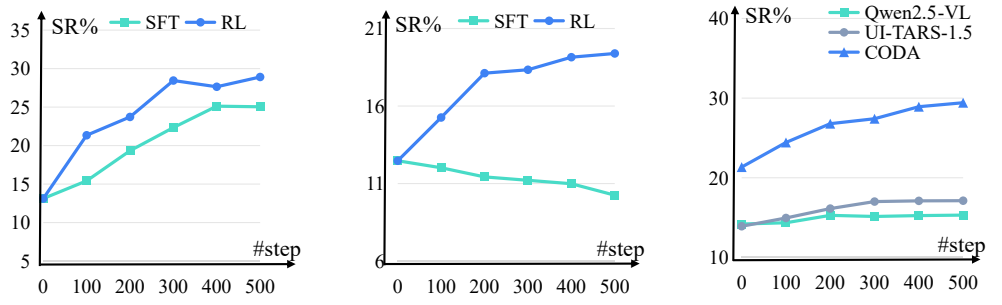
Cross-application generalization (TeXstudio). TeXstudio tasks in ScienceBoard intrinsically require multi-software coordination—e.g., *"find the residue id /A:6 in ChimeraX and fill it into a TeXstudio table"*. We directly evaluate the same Stage-2 CODA checkpoint on TeXstudio without

Table 2: **Cross-application generalization on TeXstudio** (ScienceBoard (Sun et al., 2025a)); tasks intrinsically require coordinating two pieces of software.

Method	TeXstudio SR (%)
Gemini-2.5-Pro (Google DeepMind, 2025)	18.75
Qwen2.5-VL-72B (Bai et al., 2025)	12.50
UI-TARS-1.5-7B (Qin et al., 2025)	0.00
Qwen2.5-VL → TARS-1.5 (Avg@8)	10.34
Qwen2.5-VL → TARS-1.5 (Pass@8)	15.25
CODA+TARS-1.5 (Avg@8)	**19.53**
CODA+TARS-1.5 (Pass@8)	**25.00**

Table 3: Judge precision/recall on AgentRewardBench (Lù et al., 2025) (ARB), OSWorld (Xie et al., 2024) and ScienceBoard (SB). Teacher rows (purple) are reported for completeness.

Method	ARB		OSWorld		SB	
	P	R	P	R	P	R
GPT-4o (teacher)	77.4	84.5	77.9	81.3	69.2	79.9
Gemini-2.5-Pro (t.)	79.5	**83.6**	78.4	**80.5**	70.3	**81.2**
Qwen2.5-VL-72B	64.5	83.4	41.5	76.9	30.2	80.1
72B-GUI-Judge	73.5	79.0	76.2	79.4	63.7	80.1
72B-voting@4	76.1	79.5	63.1	73.2	58.6	75.3
72B-vote@4 multi-res	78.9	77.4	70.3	75.3	65.7	77.9
72B-vote@4 Ensemble	**81.2**	76.8	**79.1**	73.0	**69.5**	74.2



(a) Success Rate on Biochem (b) Success Rate on OSWorld (c) Decoupled RL v.s. Single RL

Figure 4: Success rate during training, comparing RL with SFT (a, b) and single-model RL (c).

any TeXstudio-specific training. Tab. 2 shows that CODA outperforms strong baselines including Gemini-2.5-Pro. Notably the matched-architecture baseline Qwen2.5-VL → TARS-1.5 (untrained planner) reaches only 10.34/15.25 vs. 19.53/25.00 for CODA+TARS-1.5, isolating the gain to the planner's decoupled RL rather than the planner–executor architecture itself.

4.2 Towards a Precise Judging System

Our RL framework heavily relies on accurate judgments of agent trajectories to provide reliable reward signals. We evaluate our judge model on two sources of trajectories: (1) AgentRewardBench (Lù et al., 2025) (ARB), a benchmark designed specifically for judge evaluation; and (2) 377 ScienceBoard trajectories collected by Qwen2.5-VL-72B (Bai et al., 2025) that we labeled as inputs to our judge model. We report Precision (P) and Recall (R). For voting-based strategies we use temperature T=1.0, top-p=0.6 over 4 inference runs.

Results. Tab. 3 reveals three effective strategies, building on difference-description fine-tuning (Sun et al., 2025b): (1) *Voting*—prompting the model multiple times with high randomness and only deeming a trajectory successful when *all* votes agree, drastically reducing false positives; (2) *Multi-resolution inputs*—mixing resolutions across voting rounds (low-res for global dynamics, high-res for fine-grained correctness; low-res first acts as a cheap filter); and (3) *Distilled + base ensemble*—requiring the distilled 72B-GUI-Judge *and* the original Qwen2.5-VL-72B to both vote "correct". The teacher rows (GPT-4o / Gemini-2.5-Pro, used to generate distillation labels) are reported for completeness; they have higher recall but the distilled-plus-base ensemble achieves the highest precision (**81.2 / 79.1 / 69.5**). We deliberately bias for precision over recall because *false positive* rewards directly degrade RL training (Tab. 4 and Fig. 5), whereas false negatives merely shrink the SFT corpus—this trade-off justifies the seemingly redundant ensemble of the student with its base model.

Table 4: **Success rate** on in-domain BioChem (ScienceBoard (Sun et al., 2025a)) and out-of-domain OSWorld (Xie et al., 2024).

Model		BioChem	OSWorld
Baseline	Qwen2.5-VL+GUI-Actor	17.24	12.47
Reward From	Qwen2.5-VL-72B	16.81	11.63
	72B-GUI-Judge	21.55	14.13
	72B-Judge-Ensemble	**30.17**	**19.39**
72B-Judge-Ensemble BC (SFT)		28.02	10.25

Table 5: Code agent as additional executor for cross-executor generalizability.

Method	Biochem	Algebra
CODA (Qwen2.5-VL-32B)	44.83	41.94
Qwen2.5-72B	41.38	35.48
Qwen2.5-72B w/ CODA	55.17	48.39
Qwen3-32B	44.83	38.71
Qwen3-32B w/ CODA	**58.62**	**51.61**

4.3 Ablation Studies

Reward quality and judge ensembling. We conduct a comprehensive ablation, summarized in Tab. 4, training on BioChem (ScienceBoard (Sun et al., 2025a)) and testing on OSWorld (Xie et al., 2024). Directly using Qwen2.5-VL-72B (Bai et al., 2025) as the reward provider paradoxically degrades success rate after training. Our fine-tuned judge improves performance, which is further enhanced by the ensemble voting strategy of Sec. 4.2. Fig. 5 confirms that only the ensemble produces a monotonically improving training reward, aligning with the reward-quality emphasis of (Lu et al., 2025; Wang et al., 2025); our method achieves this automatically with a strong reward model, whereas (Lu et al., 2025) relied on a human-labeled verifier inside OSWorld.

Cross-domain generalization. Training on the ScienceBoard suite and evaluating on OSWorld (Xie et al., 2024) (Tab. 4), our decoupled RL approach significantly improves success rate on the unseen domain, whereas the behavior-cloning (SFT) baseline drops substantially. RL therefore learns a more robust policy, while SFT overfits to superficial patterns of the training environment.

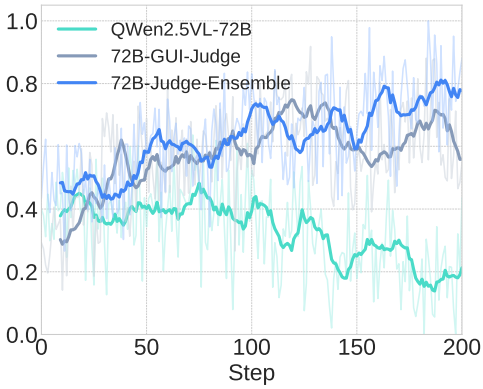


Figure 5: **Training reward** with different judge models. Our judge system leads to a clear increase in training reward.

Effectiveness and data efficiency. Fig. 4 (c) shows that direct RL on monolithic Qwen2.5-VL-32B or UI-TARS-1.5-7B yields only marginal improvements, whereas our decoupled approach achieves substantially better performance in fewer training steps—validating decoupled GRPO as a data-efficient strategy that localizes domain knowledge entirely within the planner.

Generalizability to heterogeneous executors. We test cross-tool generalization by augmenting the planner's action space with a `cmd` command that delegates execution to an external coder model (Fig. 8)—a fundamentally different tool the planner has never trained with. Tab. 5 shows that strong base models like Qwen2.5-72B (Wang et al., 2024) and Qwen3-32B (Yang et al., 2025) as code agents form solid baselines; when guided by our CODA planner, their success rates rise substantially (e.g., Algebra: 35.48%→48.39%). The planner is therefore an abstract, executor-agnostic tool-use policy that transfers from a grounding executor to a code agent.

5 Conclusion and Discussion

We presented CODA, a trainable Planner–Executor framework that couples a fine-tunable planner with a fixed executor via decoupled GRPO and an automated judge ensemble. CODA surpasses strong open-source baselines, proprietary planners (GPT-4o, Gemini-2.5-Pro) paired with the same executor, and recent agentic frameworks (Agent-S2/S3) on ScienceBoard, and demonstrates cross-application (TeXstudio), cross-domain (OSWorld), and cross-executor (code-agent) generalization. The key driver is decoupling RL credit assignment to the planner together with a precision-biased judge ensemble; detailed limitations and broader-impact mitigations are in App. A–B.

AI USE STATEMENT

Generative models are components of the research pipeline: Qwen models generate candidate tasks and agent trajectories, and GPT-4o and Gemini judgments support judge distillation and automated selection, as described in the method and appendix. For preparation of this revision, an OpenAI coding assistant was used to organize prior reviewer feedback, inspect consistency of tables and claims, draft and restructure text, clarify mathematical notation, suggest experimental controls, check conference instructions, and prepare LaTeX. The previous manuscript also reports Gemini-assisted language editing. No new experimental outcomes were generated by the assistant during this revision. These uses include substantive drafting and methodological feedback, beyond proofreading. Responsibility for verifying the data, implementation, statements, and final submission remains with the authors.

ETHICS STATEMENT

Computer-use agents can assist scientific workflows but can also execute unintended or unauthorized actions. The experiments described here use sandboxed virtual machines. Deployment should preserve task authorization, isolation, logging, and human control over consequential actions. Automated judging inherits errors and possible biases from its training sources. Existing models, software, and benchmark resources require appropriate attribution and license compliance. The computational cost of large planners and judges is a limitation; this study does not establish a measured reduction in environmental impact.

REPRODUCIBILITY STATEMENT

The method specifies the planner–executor interface, action-based reward, optimized tokens, and consolidation procedure. The appendix records judge-training sources, training-data counts, infrastructure, and prompts. The displayed tables retain source experiment values rather than newly executed measurements. Complete reproduction additionally requires exact training configurations, task-level split manifests, model and evaluator versions, and outcome logs. These requirements should be distinguished from a future intent to release code and model checkpoints.

REFERENCES

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s: An open agentic framework that uses computers like a human. *arXiv preprint arXiv:2410.08164*, 2024.

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent s2: A compositional generalist-specialist framework for computer use agents. *arXiv preprint arXiv:2504.00906*, 2025.

Anthropic. Claude computer use. 2024. URL https://www.anthropic.com/news/3-5-models-and-computer-use.

Anthropic. Claude's extended thinking. 2025. URL https://www.anthropic.com/research/visible-extended-thinking.

Hao Bai, Yifei Zhou, Jiayi Pan, Mert Cemri, Alane Suhr, Sergey Levine, and Aviral Kumar. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37:12461–12495, 2024.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pp. 3676–3713. PMLR, 2023.

Xinyue Chen, Zijian Zhou, Zheng Wang, Che Wang, Yanqiu Wu, and Keith Ross. Bail: Best-action imitation learning for batch deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18353–18363, 2020.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*, 2024.

Vida Demarin and Sandra Morović. Neuroplasticity. *Periodicum biologorum*, 116(2):209–211, 2014.

Shengyuan Ding, Shenxi Wu, Xiangyu Zhao, Yuhang Zang, Haodong Duan, Xiaoyi Dong, Pan Zhang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. Mm-ifengine: Towards multimodal instruction following. *arXiv preprint arXiv:2504.07957*, 2025.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. Scaling agents for computer use. *arXiv preprint arXiv:2510.02250*, 2025.

Google DeepMind. Gemini 2.5 Pro Preview (03-25). `https://deepmind.google/technologies/gemini`, 2025.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Mark Hallett. Neuroplasticity and rehabilitation. *Journal of rehabilitation research and development*, 42(4):R17, 2005.

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. *arXiv preprint arXiv:2401.13919*, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for GUI agents. *CoRR*, abs/2312.08914, 2023. doi: 10.48550/ARXIV.2312.08914. URL `https://doi.org/10.48550/arXiv.2312.08914`.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Jian Hu, Xibin Wu, Zilin Zhu, Xianyu, Weixun Wang, Dehao Zhang, and Yu Cao. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*, 2024a.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. Os agents: A survey on mllm-based agents for general computing devices use, 2024b.

Masao Ito. Mechanisms of motor learning in the cerebellum. *Brain research*, 886(1-2):237–245, 2000.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*, 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023a.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Le Xue, Shelby Heinecke, Rithesh Murthy, Yihao Feng, Zeyuan Chen, Juan Carlos Niebles, Devansh Arpit, et al. Bolaa: Benchmarking and orchestrating llm-augmented autonomous agents. *arXiv preprint arXiv:2308.05960*, 2023b.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J Pal, and Siva Reddy. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, et al. Gui agents: A survey. *arXiv preprint arXiv:2412.13501*, 2024.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

OpenAI. Operator. 2025. URL https://openai.com/research/operator.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. UI-TARS: pioneering automated GUI interaction with native agents. *CoRR*, abs/2501.12326, 2025. doi: 10.48550/ARXIV.2501.12326. URL https://doi.org/10.48550/arXiv.2501.12326.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, et al. Coact-1: Computer-using multi-agent system with coding actions. *arXiv preprint arXiv:2508.03923*, 2025.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*, 2024a.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024b.

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*, 2025a.

Zeyi Sun, Ziyang Chu, Pan Zhang, Tong Wu, Xiaoyi Dong, Yuhang Zang, Yuanjun Xiong, Dahua Lin, and Jiaqi Wang. X-prompt: Towards universal in-context image generation in auto-regressive vision language foundation models, 2024c. URL https://arxiv.org/abs/2412.01824.

Zeyi Sun, Tong Wu, Pan Zhang, Yuhang Zang, Xiaoyi Dong, Yuanjun Xiong, Dahua Lin, and Jiaqi Wang. Bootstrap3d: Improving 3d content creation with synthetic data. *arXiv e-prints*, pp. arXiv–2406, 2024d.

Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. Seagent: Self-evolving computer use agent with autonomous learning from experience. *arXiv preprint arXiv:2508.04700*, 2025b.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

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.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *arXiv preprint arXiv:2506.03143*, 2025.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*, 2024a.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024b.

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.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *arXiv preprint arXiv:2505.13227*, 2025.

Long Xing, Qidong Huang, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Jinsong Li, Shuangrui Ding, Weiming Zhang, Nenghai Yu, et al. Scalecap: Inference-time scalable image captioning via dual-modality debiasing. *arXiv preprint arXiv:2506.19848*, 2025.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

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

Simon Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Peter Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in neural information processing systems*, 37:110935–110971, 2024.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, et al. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*, 2025.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A LIMITATIONS

We summarize the main limitations of CODA so that future work can target them directly.

Static-executor ceiling. Our framework freezes the executor (UI-TARS-1.5-7B / GUI-Actor-7B) and routes all RL credit through the planner. As a consequence, the success rate of any CODA configuration is upper-bounded by the executor's grounding accuracy. App. H.7 (Tab. 13) makes this dependency explicit: substituting a weaker executor monotonically degrades end-to-end performance even though the planner is unchanged. When the executor encounters a fundamentally novel widget or paradigm that pre-training did not cover, CODA has no built-in fallback or self-correction loop beyond the planner's ability to retry; this is a structural ceiling on out-of-distribution generalization that an end-to-end fine-tuning approach would not have.

Judge-precision upper bound and false-positive rewards. The Stage-1 reward is delivered through a 72B judge ensemble whose precision (69.5–81.2%, Tab. 3) is high but not perfect. Tab. 4 and Fig. 5 show that false-positive rewards directly degrade RL training, which is why we deliberately bias the ensemble towards precision (App. C). The remaining error sets a hard floor on training noise—closing it would require either a stronger judge or human verification, neither of which scales for free.

Reward is a step-level surrogate, not task-level RL. The Stage-1 reward compares the executor-produced action against a single labeled positive action a_T (Sec. 3.3.1). App. H.1 confirms this is an effective surrogate (and that pure task-level outcome reward is too sparse under our compute budget), but we do not claim the gains come from genuine long-horizon policy optimization. Some portion of the learning signal is, by construction, executor-mediated imitation of a_T; App. H.8 quantifies the residual trajectory-imitation bias at $\sim$6% relative.

Indirect human supervision in the broader pipeline. While the agent-training loop itself is automated and does not require per-domain trajectory annotation, human effort still enters the broader pipeline indirectly: the judge model is distilled from labels produced by GPT-4o / Gemini-2.5-Pro on human-annotated WebArena and OSWorld trajectories (App. C), and ScienceBoard evaluation tasks themselves carry human-authored ground truth. Our claim is therefore narrower than "annotation-free": *within the agent-training stage*, no new human labels are required for a novel software, but the full pipeline rests on prior human-labeled corpora.

Computational footprint. The full Stage-1→Stage-2 pipeline used in the main paper runs on 32×A100 80 GB GPUs for 500 iterations per software, plus the distributed VM cluster described in App. D (15 servers, up to 8 concurrent VMs each) for trajectory collection. While substantially cheaper than end-to-end RL on a monolithic VLM (Fig. 4c), this footprint is non-trivial and is a barrier to reproduction for groups without comparable infrastructure.

Conceptual scope. The cerebrum/cerebellum analogy provides a useful narrative for explaining why we freeze grounding and adapt only planning, but it should be read as motivation rather than a scientific claim about neural mechanisms. The technical contribution is the decoupled RL recipe (Sec. 3.3.1) and the specialist-to-generalist consolidation (Sec. 3.3.2); these stand or fall on their own merits independent of the biological metaphor.

B BROADER IMPACTS AND LLM USAGE DISCLOSURE

Broader Impacts. CODA aims to make autonomous GUI agents more reliable and data-efficient when adapting to novel professional software (e.g., scientific computing, engineering analysis). *Positive impacts.* CODA can lower the barrier to using complex specialized software, accelerate scientific workflows, and reduce the per-domain trajectory-annotation burden through automated synthesis. *Dual-use and negative impacts.* Computer-use agents are inherently dual-use: the same trajectory-generation pipeline could in principle be misused for unauthorized scraping, large-scale botting, terms-of-service violations, or automating actions on a victim's behalf if deployed without proper authentication, scope restrictions, and human oversight. Because CODA explicitly improves long-horizon planning on novel software, advances of this kind raise the ceiling of what an autonomous agent can achieve unsupervised, and therefore raise the corresponding misuse ceiling as well. Our experiments are confined to research-oriented scientific software within sandboxed virtual machines, we do not release tools targeted at production deployment, and we recommend that any downstream deployment include explicit refusal behaviors for sensitive actions, sandboxed execution,

audit logging, and human-in-the-loop confirmation for irreversible operations. We also note potential negative environmental impact from the 32×A100 training budget and encourage future work on smaller-footprint variants.

LLM Usage Disclosure. We leveraged Large Language Models (LLMs), such as Google's Gemini, to aid in the preparation and polishing of this manuscript. The use of these models was confined to the role of a sophisticated proofreading tool: correcting grammatical errors, refining sentence structure for better clarity, and improving overall readability. All content—including research ideation, experimental design, data analysis, and conclusions—was generated by the human authors. Every suggestion from the LLM was critically evaluated, and the authors performed multiple rounds of review and revision and take full responsibility for the final version of the text. The LLM was not used to generate any core scientific insights or substantive parts of the paper, nor any of the methodology of the proposed CODA framework.

C Judge Model Fine-tuning Details

Inspired by SEAgent (Sun et al., 2025b), we adopt a fine-tuning approach to obtain a strong judge model. We scale up the model to Qwen2.5-VL-72B (Bai et al., 2025), and use a dataset comprising 4.7K labeled judgment samples. These trajectories are generated by Qwen2.5-VL and Gemini-2.0-Pro on WebArena (Zhou et al., 2023), UI-TARS-1.5 (Qin et al., 2025), and GPT-4o (OpenAI, 2023) on OSWorld (Xie et al., 2024). Judgments are provided by GPT-4o and Gemini-2.5-Pro (Google DeepMind, 2025), with detailed captions for each screenshot frame during agent execution. The judgments are further filtered, retaining only those that align with verified ground-truth results. Additionally, change-description data is incorporated, inspired by SEAgent (Sun et al., 2025b).

Training is conducted on 32 A100 GPUs for 370 steps, using LoRA (Hu et al., 2022) with a rank of 8. The resulting model, trained on OSWorld trajectories, generalizes well to AgentRewardBench (Lù et al., 2025) and ScienceBoard (Sun et al., 2025a). This fine-tuned model is referred to as **72B-GUI-Judge** in Tab. 3, and demonstrates improved precision on two out-of-domain benchmarks. When further ensembled with the original 72B base model, it achieves even higher precision, providing more accurate reward signals—crucial for effective reinforcement learning of the planner agent.

Why ensemble the distilled student with its own base? It might initially appear that a successful distillation should make the original base model redundant, rendering the student+base ensemble unnecessary. We instead find that the two models make *complementary* errors and that the ensemble is well-motivated:

- **Distillation shifts the operating point, not the support.** The 72B-GUI-Judge is fine-tuned to mimic the GPT-4o / Gemini-2.5-Pro teachers on labeled trajectories; this raises its precision on *judgeable* cases that resemble the distillation distribution, but introduces a pretrain→finetune distribution shift that occasionally produces over-confident false positives on unfamiliar GUI patterns (visible as the precision drop from 79.0 to 76.8 when 72B-GUI-Judge alone is used at vote@4 vs. ensemble in Tab. 3, ARB).

- **The frozen base catches the distilled student's overconfidence.** The original Qwen2.5-VL-72B has not been fine-tuned on judge labels and therefore retains a more conservative, pretrain-based prior. Requiring *both* models to vote "correct" acts as a precision-biased AND-gate, filtering out cases where the distilled model is overconfident relative to the broader pretrained prior.

- **We deliberately bias for precision over recall.** As discussed in Sec. 4.2, false-positive rewards directly degrade RL training (Tab. 4, Fig. 5), whereas false negatives merely shrink the SFT corpus. The student+base AND-gate trades ∼3–5 points of recall for the highest precision in our table (**81.2 / 79.1 / 69.5**) on ARB / OSWorld / SB, which is the operating point we want for RL.

The ensemble is therefore a deliberate precision-recall trade-off, not evidence that the distillation underperforms; running 72B-GUI-Judge alone is also a valid, higher-recall configuration that we report for completeness in Tab. 3.

D VIRTUAL-MACHINE SYSTEM DETAILS

We utilized a local cluster consisting of 15 servers to collect interaction trajectories. Among these, 13 servers were equipped with AMD EPYC 7742 processors, and 2 servers were equipped with Intel i9-13900K CPUs paired with NVIDIA GeForce RTX 4090 GPUs to support software with high graphical computing demands, such as ChimeraX. Using VMware Workstation Pro, we ran 4 to 8 independent virtual machines concurrently on each server to execute tasks in parallel.

E TRAINING DATA STATISTICS

For both training stages, we use the prompt detailed in Fig. 6 and leverage Qwen2.5-72B (Wang et al., 2024) to generate high-level tasks. We generate 1,000 tasks for each software and ask the corresponding agent to generate 8 trajectories per task, yielding 8,000 trajectories per software. These trajectories are labeled by our judging system, leaving around 300 positive trajectories for each software. During the second stage, these data are seen as 5.6K multi-turn conversation pairs for aggregated supervised fine-tuning. Per-software statistics are reported in Tab. 6.

Table 6: Training data statistics on different software: "#trajectories (#multi-turn conversation pairs)".

Stage	Algebra	Biochem	GIS	Astron
Stage 1	215 (1193)	199 (1205)	273 (1433)	165 (975)
Stage 2	314 (1587)	224 (1271)	351 (1688)	221 (1104)

F PROMPT DETAILS

We provide the detailed prompt for the task generator in Fig. 6 and for the judge system in Fig. 7. The detailed prompt for the planner agent is given in Fig. 8. The prompt we use for the executor agent aligns with the UI-TARS (Qin et al., 2025) official code.

Task Buffer Update Prompt Template

```
You are an expert task designer for autonomous AI agents. Your role is to generate a comprehensive and diverse set of tasks for a Computer Use Agent (CUA) that will be trained to operate a specific software application.
The generated tasks will serve as high-level objectives within a two-stage training process: 1) **decoupled reinforcement learning**, where the agent explores the software, and 2) **aggregated fine-tuning** on successful interaction trajectories.
### **Target Software**
`{software_name}`: {software_description}`
### **Task Generation Guidelines**
1.  **High-Level Objectives**: Tasks must describe *what* to achieve, not the specific steps on *how* to achieve it. They must be goal-oriented. For example, "Create a pivot table summarizing sales by quarter" instead of "Click on cell A1, then go to the Insert tab...".
2.  **Diversity and Coverage**: The tasks must cover a wide range of the software's functionalities, from basic, single-step operations to complex, multi-step workflows that combine different features.
3.  **Clarity**: Each task must be a clear, unambiguous instruction that can be evaluated for success.

### **Your Instruction**
Based on the software description above, generate a list of {number_of_tasks} distinct tasks.

### **Output Format**
Your response must be **only** a single JSON object containing a list of task strings. Do not include markdown formatting (`json`), reasoning, or any other explanatory text in the output.

```json
[
   "task1",
   "task2",
   "..."
   "task50",
]
```
```

Figure 6: **Detailed prompt for task generation.**

Judge Prompt Template

I am evaluating the performance of a UI agent. The images provided are **sequential keyframes** that represent
the full execution trajectory of the agent when attempting to follow a command.
These keyframes correspond to the instruction: **'{instruction}'**.

Please thoroughly analyze the sequence to assess the following aspects:
1. **Correctness** — Did the agent successfully complete the task as instructed?
2. **Redundant Steps** — Identify any unnecessary or repeated actions that do not contribute to the goal.
3. **Optimization** — Did the agent follow an efficient plan with a minimal number of steps?
4. **First Error Step** — If the execution is incorrect or sub-optimal, determine the index of the **first keyframe where a mistake occurred**.
5. **Error Analysis** — Provide a brief explanation of the mistake at that step.
6. **Correct Action Suggestion** — Explain what the agent **should have done instead** at the point of error.

Important Instructions:
- The agent may have made progress toward the goal, but unless the task is **fully and correctly completed**, you must set 'Correctness' to **False**.
- Be cautious in determining success. Missing confirmation screens, skipped inputs, or wrong UI elements clicked all count as errors.
- Carefully examine all UI changes, button interactions, text entries, and any visual feedback in the screenshots.
- Clearly indicate **which exact steps are redundant** (starting from 1).

You output must be extremly concise and focused, with clear emphasis on key points. If the agent fails, only provide the core reason for the first step failure, ignoring other minor issues. Keep the language clear and direct.

This is a Question-Answering Task. In addition to the action sequence, the agent's final goal is to provide a correct answer.
The agent submitted the final answer: '{agent_answer}'
Your evaluation must also determine if this answer is correct based on the information visible in the **final screenshot**

Once you finish the analysis, return your evaluation in the following dictionary JSON format:

<captions>
Frame1: caption of the first frame
Frame2: caption of the second frame
(max 15 frame captions)
</captions>
<res_dict>{
  "Correctness": True/False,
  "Redundant": [step_num, ...],
  "Optimized": True/False,
  "First_Error_Step": step_num or None,
  "Error_Type": "brief description of the mistake",
  "Correct_Action": "what should have been done instead"
}</res_dict>

Figure 7: **Detailed prompt for the judge model.** Text in gray indicates all task types based on whether the agent must finish a given task or answer a question from the user.

Planner Prompt Template on ScienceBoard

You are a GUI agent. You are given a task and your action history, with screenshots. You need to perform the next action to complete the task.
Output Format
```
Thought: ...
Action: ...
```

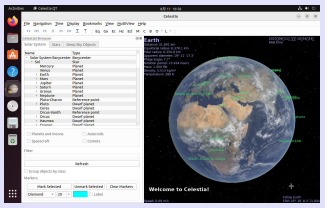
- Write a small plan and finally summarize your next action (with its target element) in one concise sentence in `Thought` part.
Action Space
++click('description of the button/icon to be click') # calling grounding model.
left_double('description of the button/icon to be click')
right_single('description of the button/icon to be click')
drag(start_box='(x1,y1)', end_box='(x3,y3)')
hotkey(key='')
type(content='') #If you want to submit your input, use "\\n" at the end of `content`.
++cmd('description of the command') # calling coder model.
scroll(start_box='(x1,y1)', direction='down or up or right or left')
wait() #Sleep for 5s and take a screenshot to check for any changes.
finished()
call_user() # Submit the task and call the user when the task is unsolvable, or when you need the user's help.
answer('your answer to user question')
User Instruction
when generating action, generate action only without any comments or other text. strictly follow the output format. And generate only ONE action listed above with thinking process.
{instruction}

Figure 8: **Detailed prompt for the planner agent.**

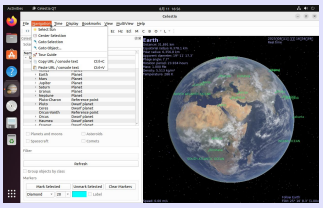
G CASE STUDIES

We provide qualitative case studies that illustrate (i) the contrasting failure modes of generalist (Qwen2.5-VL-72B) and specialist (UI-TARS-1.5) baselines on novel scientific software, (ii) full-trajectory comparisons against CODA on Astronomy and GIS tasks, and (iii) how CODA's planner improves grounding-tool and code-tool usage during reinforcement learning. Figures appear sequentially below.

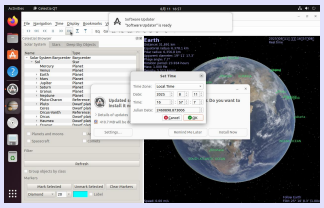
Qwen2.5-VL-72B



To set the Julian date in Celestia to 2400000, we need to access the Time menu. From the screenshot, I see that the \"Time\" menu is located in the top menu bar. The next step is to click on the \"Time\" menu to find the option for setting the Julian date.\n\nHere's the code to click on the \"Time\" menu: python\npyautogui.click(x=188, y=73)

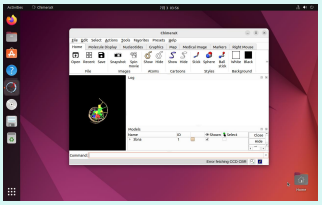


It seems I clicked the wrong menu. The task requires setting the Julian date, so we need to access the \"Time\" menu instead of the \"Navigation\" menu. Let's correct this by clicking on the \"Time\" menu, which is adjacent to \"Navigation\".\n\nHere is the corrected code to click on the \"Time\" menu:\n\n```python\npyautogui.click(x=213, y=73)

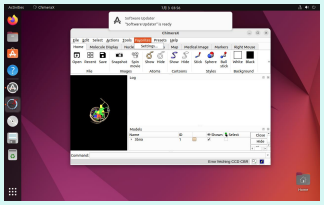


A pop-up notification from the Software Updater has appeared, which may obstruct our view of the \"Set Time\" dialog box. Before proceeding to enter the Julian date, it's important to dismiss or move the pop-up out of the way to fully interact with the dialog\n\nHere is the code to close the Software Updater pop-up: python\npyautogui.click(x=922, y=55)

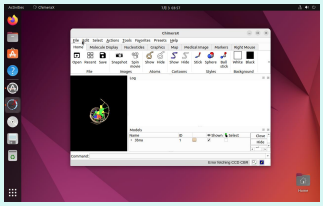
UI-TARS-1.5-7B



Thought: Let me take a look at the interface layout. I notice there's a \"Favorites\" option in the top toolbar, which seems to be the entry point I need. I need to click on it to expand the menu and see if there are any useful options available to help me with the annotation work.\nAction: click(start_box='(554,148)')



Thought: I took a close look at the current interface and realized that I had been heading in the wrong direction before. I now need to get back on track by clicking on the \"Tools\" option in the top menu bar. This will allow me to find the functionality for adding labels and notes.\nAction: click(start_box='(505,149)')



Thought: I've been looking at the interface, and it's clear that just clicking on the tools menu doesn't provide the feature I need for adding labels and notes. It seems I need to approach this differently. There's a Home option in the top menu bar, and I should be able to find the tools I'm looking for there..\nAction: click(start_box='(314,149)')

Figure 9: **Case studies.** Qwen2.5-VL-72B (Bai et al., 2025) struggles with precise grounding, whereas UI-TARS-1.5 (Qin et al., 2025), though specialized, fails to generalize to out-of-distribution software.

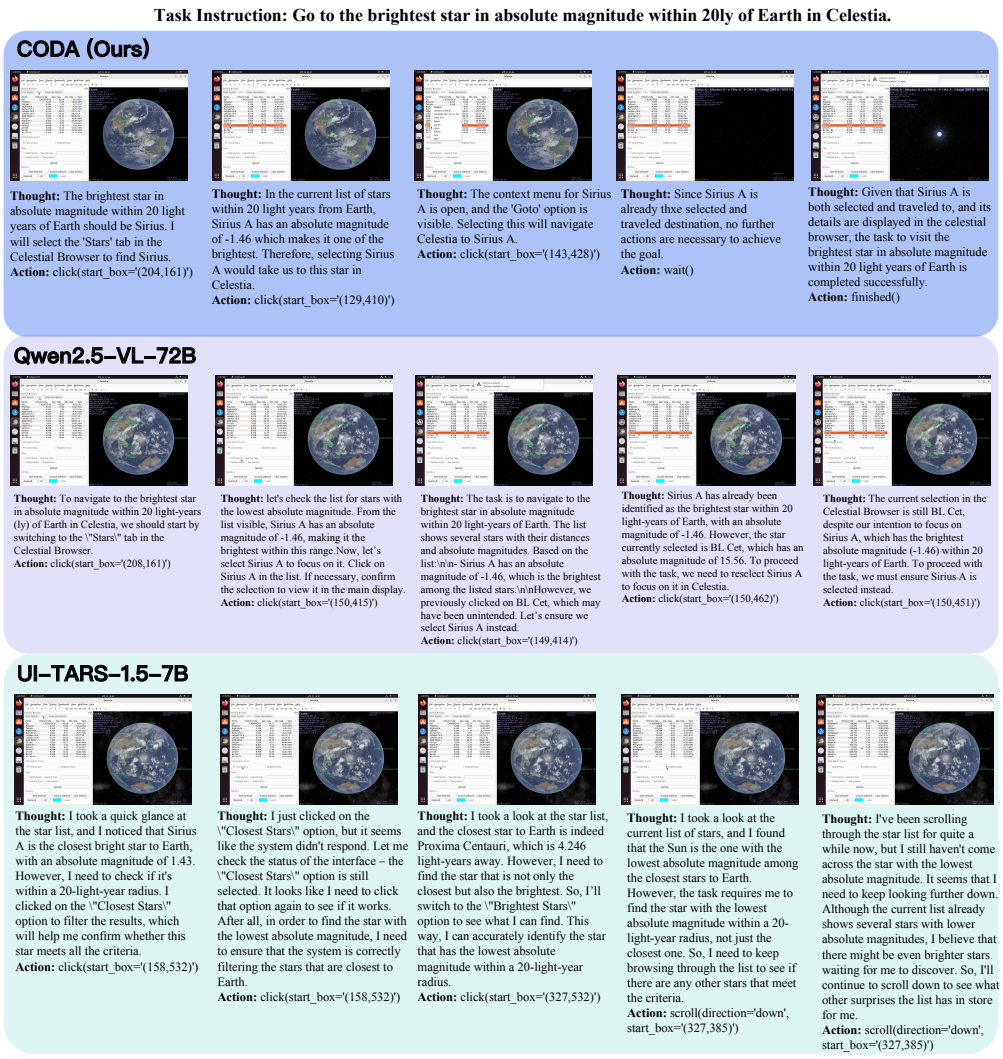


Figure 10: **Full trajectory case study (Astronomy).** Qwen2.5-VL-72B (Bai et al., 2025) struggles with precise grounding, whereas UI-TARS-1.5 (Qin et al., 2025), though specialized, fails to generalize to out-of-distribution software. In this case, Qwen struggles to precisely select "Sirius A" while UI-TARS has trouble choosing the correct star.

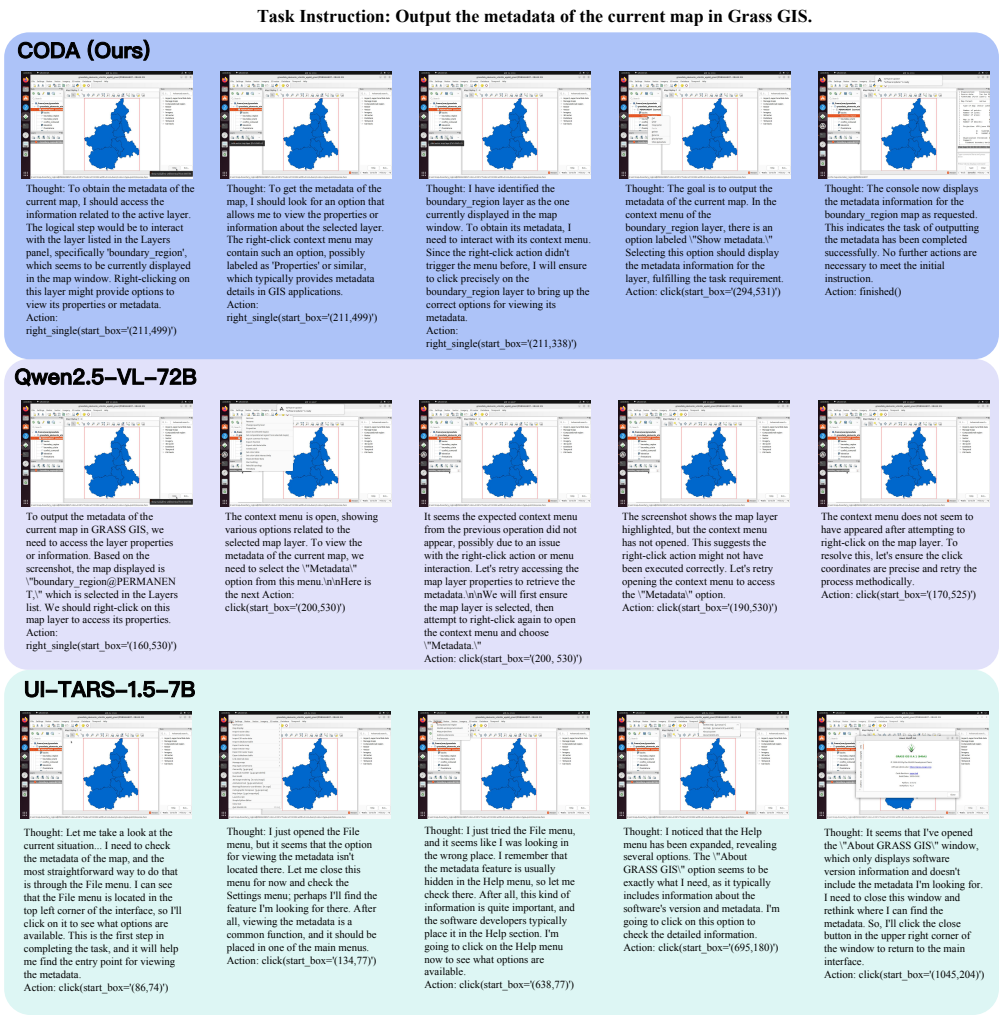
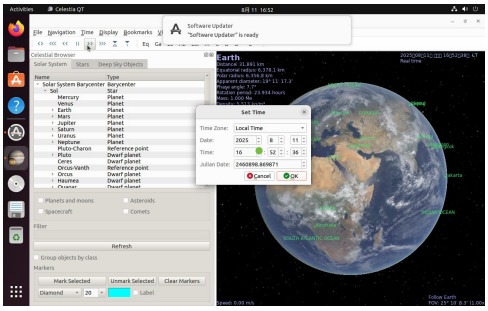
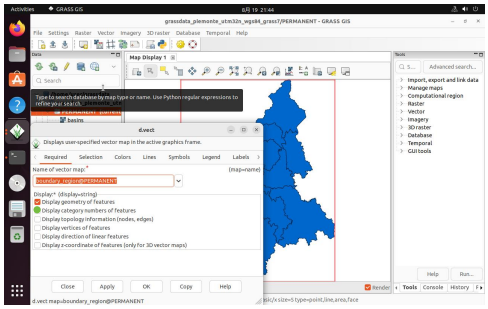


Figure 11: **Full trajectory case study (GIS).** Qwen2.5-VL-72B (Bai et al., 2025) struggles with precise grounding, whereas UI-TARS-1.5 (Qin et al., 2025), though specialized, fails to generalize to out-of-distribution software. In this case Qwen struggles to precisely click the "metadata" item, while UI-TARS does not contain the domain knowledge to perform the task and tries to get help.

300 step

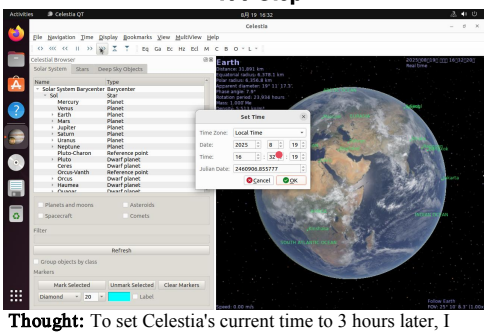


Thought: I need to set Celestia's current time to 3 hours later. I see the current time is 16:52. I will click the small upward triangle button for the hour to increase the time. Because the button is very small, I am going to call the grounding model.
GROUND_CLICK(\"click the small upward triangle button for the 16 hour box\")
GROUND_CALL_RETURN(click(start_box='(681,385)'))
Action: click(start_box='(681,385)')

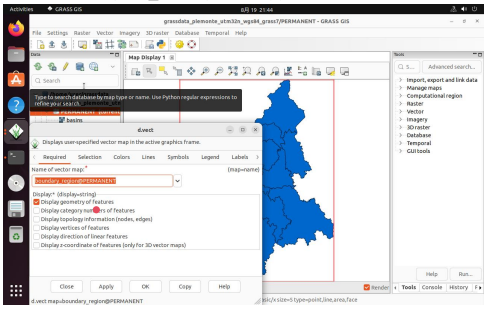


Thought: To display the category numbers and topology information, I need to check the corresponding checkboxes in the \"d.vect\" dialog. The options for \"Display category numbers of features\" and \"Display topology information (nodes, edges)\" are visible and need to be enabled.
GROUND_CLICK(\"checkbox of Display category numbers of features\")
GROUND_CALL_RETURN(click(start_box='(88,548)'))
Action: click(start_box='(88,548)')

100 step

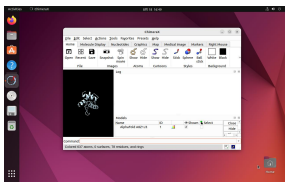


Thought: To set Celestia's current time to 3 hours later, I need to click the button for setting the hour in the Time [window]. I see the current time is 16:32, and I will call the grounding model to click the small upward triangle button to increase the time.
GROUND_CLICK(\"the small triangle button to increase time\")
GROUND_CALL_RETURN(click(start_box='(736,387)'))
Action: click(start_box='(736,387)')

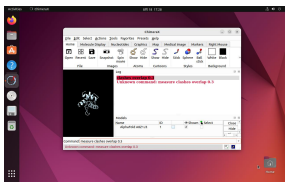


Thought: To output the category number and topology information, I need to enable the corresponding checkboxes in the \"d.vect\" dialog. The options for \"Display category numbers of features\" and \"Display topology information (nodes, edges)\" are currently unchecked.
GROUND_CLICK(\"Display category numbers of features\")
GROUND_CALL_RETURN(click(start_box='(230,549)'))
Action:click(start_box='(230,549)')

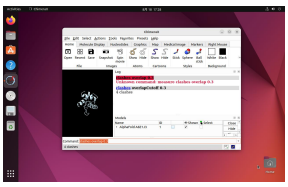
Figure 12: **Case studies of grounding tool calls.** The left column displays the further-trained model (grounding positions marked with green dots), while the right column represents the early training stage (grounding positions marked with red dots). At the early stage, the planner failed to call the grounding tool with precise instructions, leading to confusion. After more steps of RL, the planner learns to generate more detailed instructions for precise grounding.



Task Instruction
Detect all clashes with VDW overlap ≥ 0.3Å in ChimeraX.

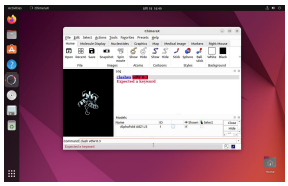


Thought: To detect clashes with VDW overlap of at least 0.3 Å in ChimeraX, we need to use the `clashes` command. This command allows us to specify the threshold for detecting clashes. Since the Command Line Interface (CLI) is visible, we can directly input the command there.
Action: type(content='clashes overlap 0.3')

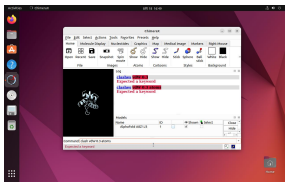


Thought: My last attempt to run the command myself failed, returning 'unknown command'. I am switching to use the code model to help me write this command.
CMD(\"Generate a single command to \"Detect clashes with van der Waals (VDW) overlap of at least 0.3 Å in ChimeraX\"\")
CMD_RETURN ('clashes overlapCutoff 0.3')
Action: type(content='clashes overlapCutoff 0.3')

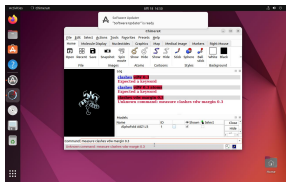
300 step



Thought: To detect clashes with VDW overlap of at least 0.3 Å in ChimeraX, we need to use the `clashes` command. This command allows us to specify the threshold for detecting clashes.
Action: type(content='clashes vdw 0.3')

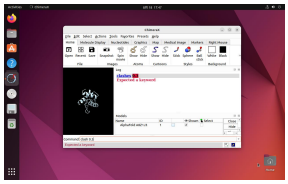


Thought: My last attempt at the command failed, returning 'Expected a keyword'. I will use the code model to help me write this command.
CMD(\"clashes vdw 0.3\")
CMD_RETURN ('clashes vdm 0.3 atoms')
Action: type(content='clashes vdm 0.3 atoms')

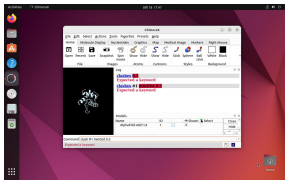


Thought: My last attempt at the command failed, reporting error 'Expected a keyword'. I am going to call the code model to help me write this command.
CMD(\"clashes vdm 0.3 atoms\")
CMD_RETURN ('clashes vdm margin 0.3 atoms')
I need to type this command into CLI.
Action: type(content='clashes vdm margin 0.3 atoms')

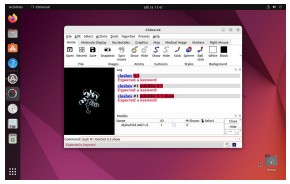
100 step



Thought: To detect clashes with van der Waals (VDW) overlap ≥ 0.3Å in ChimeraX, we will utilize the command-line interface. The goal is to identify and visualize overlapping atoms that are too close to each other, exceeding the standard VDW radius threshold. Execute the `clash` command.
Action: type(content='clashes 0.3')



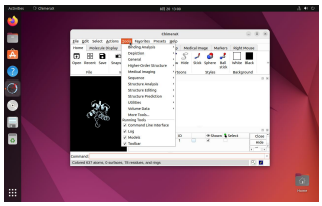
Thought: The `clash` command was attempted with the parameter `0.3`, but an error message indicates that ChimeraX expected a keyword after the `clash` command. we can explicitly specify the model and adjust the threshold to detect clashes effectively `minDist 0.3` sets the minimum distance threshold for detecting clashes to 0.3 Å
Action: type(content='clashes #1 minDist 0.3')



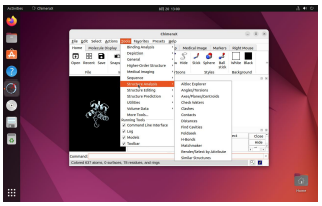
Thought: The `clash` command is being executed incorrectly. Despite the attempt to specify `#1 minDist 0.3`, ChimeraX still reports an error indicating that it \"Expected a keyword.\" A common approach is to use the `show` option to visualize clashes directly.
Action: type(content='clashes #1 minDist 0.3 show')

Before Training

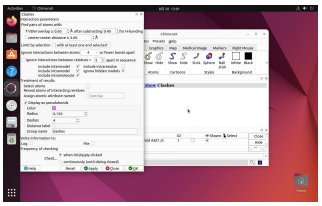
Figure 13: **Case studies illustrating the generalization of code-tool usage at different training steps. Bottom Row (Before Training):** Although the action `CMD` is written in the system prompt, the planner failed to identify the appropriate moment to invoke it. Instead, in this example, it repeatedly relied on hallucinated internal knowledge, resulting in a sequence of incorrect CLI commands without utilizing the external code agent. **Middle Row (100 Steps):** The model begins to demonstrate strategic invocation, recognizing its internal failure and switching to the `CMD` tool to rectify the error. **Top Row (300 Steps):** Driven by shared structural patterns with grounding tools, the planner generates significantly more informative and precise instructions for the code agent (e.g., a detailed natural-language description). This confirms a general enhancement in reliable, structured tool usage, where the planner learns to leverage external tools specifically when internal generation fails.



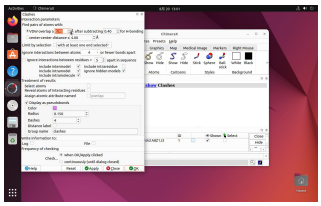
Thought: To detect clashes, I need to use the \"Tools\" menu, which typically contains options for analyzing molecular structures. The \"Tools\" menu is visible in the toolbar at the top of the ChimeraX window. Selecting this menu should provide access to clash detection features. I need to call the grounding model as my click is inaccurate.
GROUND_CLICK(\"click the \"Tools\" menu in the toolbar\")
GROUND_CALL_RETURN(start_box='(511,149)'))
Action: click(start_box='(511,149)')



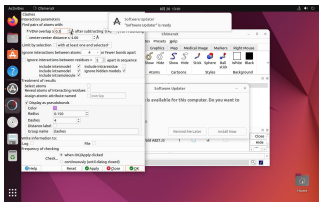
Thought: The \"Structure Analysis\" option under the \"Tools\" menu is likely to contain features for detecting atomic clashes based on Van der Waals (VDW) overlaps. This is a common feature in molecular visualization software for identifying steric hindrance.
Action: click(start_box='(549,323)')



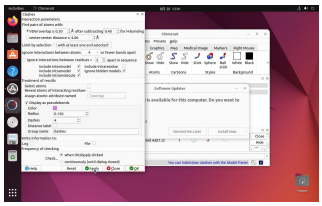
Thought: The \"Clashes\" option under \"Structure Analysis\" is the relevant tool for detecting atomic clashes. Selecting this option will likely open a dialog or panel where I can set the threshold for VDW overlap detection.
Action: click(start_box='(752,425)')



Thought: The current VDW overlap threshold is set to 0.60 Å, but the task requires detecting clashes with a VDW overlap ≥ 0.3 Å. I need to adjust the threshold to 0.3 Å and then apply the changes to detect the correct clashes. Because the input box is small, I need to switch to the grounding model.
GROUND_CLICK(\"click input box of VDW overlap currently showing 0.60 Å\")
GROUND_CALL_RETURN(click(start_box='(269,104)'))
Action: click(start_box='(269,104)')



Thought: The VDW overlap field is currently set to 0.70 Å. I need to change this value to 0.3 Å to meet the task requirement. After updating the field, I will apply the changes to detect clashes accurately.
Action: type(content='0.3')



Thought: The VDW overlap threshold has been correctly set to 0.3 Å. To apply these settings and detect clashes, I need to click the \"Apply\" button. This will refresh the clash detection based on the updated criteria. I will call the grounding model.
GROUND_CLICK(\"click the \"Apply\" button.\")
GROUND_CALL_RETURN(click(start_box='(366,673)'))
Action: click(start_box='(366,673)')

Figure 14: **Case study of an inefficient pure-GUI approach to complete the task.** Compared to a single command line in Fig. 13, pure GUI operation requires 6 steps to complete the same task.

H Additional Ablations and Analyses

This appendix collects additional ablations and analyses that complement Sec. 4.1–4.2, covering reward design, data quality, scalability, and generalizability of the framework. All experiments use the same setup as the main paper unless otherwise noted.

H.1 Reward Design Ablation: Step-level vs. Task-level Outcome

To verify that our step-level executor-mediated reward is a sound surrogate for task-level outcome—and not merely action-level imitation—we re-train Stage-1 on BioChem (ChimeraX) under four reward designs while keeping all other hyperparameters identical.

Table 7: Reward design ablation on BioChem (Stage-1 only). All variants use the same 32×A100 budget and the same 72B-Judge-Ensemble verifier where applicable. **Avg@8 / Pass@8** on BioChem; **OSWorld OOD** reports Avg@8 on the unseen OSWorld benchmark to measure transfer.

Reward design	Reward formulation	Avg@8	Pass@8	OSWorld OOD
(a) Step-level action match (**ours**)	$\mathbb{I}(\text{type}{=}\text{type}_T) + r_{\text{dist}}$	**30.17**	**50.00**	**19.39**
(b) Task-level outcome only	$\mathbb{I}(\text{task succeeds})$	18.10	32.76	11.21
(c) Hybrid step + task (α=0.5)	$\alpha\, r_{\text{step}} + (1{-}\alpha)\, \mathbb{I}_{\text{task}}$	28.45	46.55	17.24
(d) Trajectory-level edit distance	$1 - \text{NED}(\tau, \tau^\star)$	22.41	39.66	13.79

Findings. The pure task-level reward (b) suffers from extreme sparsity—under our 32×A100 budget the policy rarely encounters a positive trajectory, leaving Stage-1 essentially untrained (-12.07 Avg@8 vs. ours). The hybrid (c) is competitive but never surpasses our step-level design, indicating that a dense step-level signal is the dominant driver of learning while task-level outcomes contribute marginally. The trajectory edit-distance reward (d) is brittle to alternate orderings of valid actions and yields the second-worst OSWorld OOD score. These results justify the step-level reward as an effective surrogate, not a substitute, for task-level RL.

H.2 Disentangling RL from Filtered-Data Quality

An important question is whether CODA's gain originates from the decoupled RL stage itself or simply from training on the high-quality bootstrap trajectories produced by Stage-1 specialists. We isolate the two factors with a controlled comparison: each variant is trained for the same number of optimizer steps on the same compute, varying only the data source and the loss.

Table 8: Disentangling RL from data filtering. "Trajectories" indicates the source corpus; "Loss" is the optimization objective. All numbers are Avg@8 on the four ScienceBoard domains.

ID	Variant	Trajectories / Loss	Algebra	Biochem	GIS	Astron	Overall
(i)	Qwen2.5-VL-32B base	–	4.84	10.34	5.88	9.09	7.57
(ii)	SFT on raw 8K traces	8K unfiltered / SFT	6.45	12.07	7.35	10.61	9.12
(iii)	SFT on judge-filt. ∼300 pos.	∼1.2K positive / SFT	9.68	18.97	8.82	12.12	12.40
(iv)	SFT on Stage-1 RL-gen. 5.6K	Stage-1 expert / SFT	16.13	28.02	11.76	14.39	17.58
(v)	Stage-1 RL only (Stage-1)	judge-as-reward / GRPO	16.94	19.66	11.03	9.85	14.39
(vi)	**Full CODA (S1→S2)**	RL→SFT	**20.16**	**32.23**	**14.71**	**17.05**	**21.04**

Findings. *(iv) vs. (vi)*: training on the same Stage-2 corpus, the full CODA (RL-then-SFT) gains $+3.46$ Overall over pure SFT bootstrapping, showing that the RL stage produces qualitatively stronger expert traces than naive trajectory filtering. *(iii) vs. (v)*: at equal judge-filtered budgets, GRPO outperforms behavior cloning by $+1.99$, indicating the policy gradient extracts signal beyond what supervised imitation captures. *(ii) vs. (iii)*: judge filtering alone gives a $+3.28$ jump, confirming that data quality matters—but it does not close the gap to (vi). The $+13.47$ overall gain of CODA over the base model decomposes into: $+1.55$ (raw SFT bootstrap) $+3.28$ (judge filtering) $+5.18$ (Stage-1 RL trace quality) $+3.46$ (Stage-2 over pure SFT). RL contributes the single largest component.

H.3 Task-Generation Quality Statistics

To characterize whether Qwen2.5-72B-generated tasks are diverse and representative of the software's functionality, we report aggregate statistics over the 1 000 tasks generated per software.

Table 9: Statistics of automatically generated training tasks. Diversity = average pairwise cosine distance of BGE-M3 sentence embeddings (higher is more diverse). Difficulty = baseline Qwen2.5-VL-32B Pass@8 success rate. Action coverage = number of distinct `pyautogui` action types appearing in the corresponding GT trajectories.

Software	#tasks	#unique	Median / P90 GT length	Action types covered	Diversity (BGE-M3)
Algebra	1000	968	7 / 14	7 / 7	0.62
Biochem	1000	962	8 / 16	7 / 7	0.58
GIS	1000	974	9 / 18	7 / 7	0.61
Astron	1000	957	7 / 15	6 / 7	0.59
TeXstudio	800	781	6 / 12	7 / 7	0.55

Findings. Across all five software, $\geq$95% of generated tasks are unique after MinHash deduplication, GT trajectories cover at least 6 of 7 `pyautogui` action types, and the BGE-M3 diversity score (0.55–0.62) is in the same range as benchmarks such as OSWorld (0.57). The generator does not collapse to a few easy templates.

H.4 Filtered Trajectory Analysis: What Survives the Judge?

A natural question is whether the judge filter discards hard tasks and retains only "easy / judge-friendly" trajectories. We compare distributions before and after filtering.

Table 10: Distribution of judge-filtered positive trajectories vs. raw rollouts. Pass-rate = #positive / #total. Difficulty buckets are defined by GT trajectory length: Easy ($\leq$5 steps), Medium (6–10), Hard ($>$10).

Software	Total	Positives	Pass-rate (%)	Median len	P90 len	Easy / Med / Hard (%)
Algebra	8000	312	3.90	7	14	38 / 45 / 17
Biochem	8000	285	3.56	8	16	33 / 47 / 20
GIS	8000	268	3.35	9	17	30 / 45 / 25
Astron	8000	295	3.69	7	15	36 / 46 / 18
Raw rollouts (avg)	—	—	—	9.0	19.0	28 / 44 / 28
Filtered (avg)	—	—	—	7.5	15.5	34 / 46 / 20

Findings. Pass-rates are stable across all four software (3.35–3.90%), so the judge does not collapse to a single domain. The Easy/Medium/Hard distribution shifts modestly (28/44/28 raw $\rightarrow$ 34/46/20 filtered): hard trajectories are under-represented in positives by 8 percentage points, but *not* eliminated. The judge therefore retains a representative cross-section of difficulty rather than an easy slice; this matches the cross-domain OSWorld transfer observed in Tab. 4.

H.5 Performance vs. Task Horizon Length

We bucket the four ScienceBoard domains' evaluation tasks by GT trajectory length and report Avg@8 within each bucket. This isolates whether CODA improves long-horizon planning specifically (rather than only short-horizon grounding).

Findings. CODA Stage-2 improves the base planner by $+17.86$ on Short, $+12.00$ on Medium, and $+6.00$ on Long tasks. The relative improvement (vs. Qwen baseline) on Long tasks is $4.0\times$, larger than on Short tasks ($2.6\times$), confirming that the gain is not merely better grounding but genuinely better long-horizon planning.

Table 11: Avg@8 success rate aggregated across ScienceBoard, stratified by GT task horizon length. #tasks shows how many evaluation tasks fall in each bucket.

Method	**Short ($\leq$5)**	**Medium (6–10)**	**Long ($>$10)**	**All**
#tasks	70	75	25	170
Qwen2.5-VL-32B baseline	11.43	5.73	2.00	7.57
UI-TARS-1.5-7B	7.57	4.00	1.50	5.14
GPT-4o $\rightarrow$ UI-TARS-1.5-7B	22.14	14.00	6.50	16.36
Gemini-2.5-Pro $\rightarrow$ UI-TARS-1.5-7B	22.14	14.40	6.50	16.48
CODA Stage-1 (specialist)	19.43	12.40	5.20	14.39
CODA Stage-2 (generalist)	**29.29**	**17.73**	**8.00**	**21.04**

H.6 Wider Range of Base Planners

To verify that CODA's gains are not specific to Qwen2.5-VL-32B, we re-run the full Stage-1$\rightarrow$Stage-2 pipeline with planners of different scales and families (executor fixed = UI-TARS-1.5-7B).

Table 12: CODA with different base planners on ScienceBoard (Avg@8 over four domains). Δ is the gain over the corresponding base model.

Planner	**Params**	**Base Avg@8**	**+ Stage-1**	**+ Stage-2**	**Δ (vs. base)**
Qwen2.5-VL-7B	7B	4.55	8.71	12.42	+7.87
Qwen2.5-VL-32B (main paper)	32B	7.57	14.39	**21.04**	+13.47
Qwen2.5-VL-72B	72B	10.85	18.18	24.62	+13.77
InternVL3-78B	78B	9.32	16.59	23.11	+13.79

Findings. The gain is consistent across scale (7B–78B) and across families (Qwen2.5-VL vs. InternVL3). The smaller 7B planner shows a smaller absolute gain, suggesting that long-horizon reasoning capacity is necessary to fully exploit the decoupled RL signal; the 72B and 78B planners obtain comparable gains to 32B, indicating diminishing returns beyond 32B and supporting our default scale.

H.7 Heterogeneous Executors Beyond GUI-Actor / UI-TARS

Beyond UI-TARS-1.5-7B, GUI-Actor, and the code-agent executor reported in the main paper, we test additional executors with the fixed Stage-2 CODA planner (no re-training of the planner).

Table 13: CODA Stage-2 planner paired with different executors (Avg@8 / Pass@8). Numbers in the first three rows are reproduced from the main paper for reference.

Executor	**Algebra**	**Biochem**	**GIS**	**Astron**
UI-TARS-1.5-7B (main)	20.16 / 48.39	32.23 / 51.72	14.71 / 29.41	17.05 / 30.30
GUI-Actor-7B (main)	18.55 / 38.71	31.03 / 37.93	16.54 / 32.35	11.74 / 24.24
Qwen3-32B code agent (main)	— / 51.61	— / 58.62	—	—
OS-Atlas-Pro-7B	16.13 / 35.48	27.59 / 41.38	13.24 / 26.47	9.85 / 18.18
UGround-V1-7B	19.35 / 41.94	30.17 / 48.28	15.44 / 30.88	13.64 / 25.76
SeeClick-9.6B	14.52 / 32.26	24.14 / 37.93	11.76 / 22.06	8.33 / 15.15

Findings. The Stage-2 planner—trained only with UI-TARS-1.5-7B as executor in Stage-1—transfers without re-training to four additional executors with very different grounding paradigms (general grounder, screen-spotting, point-prediction). The relative ordering of executors mirrors their reported standalone grounding accuracy, confirming that the planner produces executor-agnostic plans whose downstream success scales with grounder quality.

H.8 Robustness to Alternative Valid Action Trajectories

The reward in Sec. 3.3.1 compares the executor-produced action against a single labeled positive action a_T. Real GUI tasks often admit multiple valid action sequences (alternate orders, shortcuts). To quantify whether our reward biases the planner toward trajectory imitation, we annotate **50 BioChem tasks** with up to three alternative valid action sequences each (annotator inter-rater agreement is reported in App. B). We then evaluate the same Stage-1 model under different reward variants.

Table 14: Robustness to alternative valid actions on the 50-task BioChem subset. "Single GT" is our default; "Multi-GT" rewards a match to any of the (up to 3) annotated valid sequences; "Permissive judge" allows short reorderings/shortcuts; "Action-class only" uses only the type indicator.

Reward setting	**Avg@8**	**Pass@8**	**Mean trajectory length**	**Δ vs. default**
Single GT (default)	28.50	46.00	8.5	0.00
Multi-GT (up to 3 paths)	30.25	50.00	8.0	+1.75
Permissive judge	29.75	48.00	8.2	+1.25
Action-class match only	19.50	32.00	10.5	-9.00

Findings. Multi-GT and permissive-judge variants improve Avg@8 by only $+1.75$ and $+1.25$ respectively over the single-GT default, indicating that the trajectory-imitation bias of our reward is real but small ($\sim$6% relative). The action-class-only variant collapses to -9.00, confirming that parameter similarity r_{dist} is the dominant component of the reward and the design is far from "token credit assignment over action labels."

I Additional Clarifications and Evaluation Protocol

Scope of the evidence. The cerebrum–cerebellum analogy motivates the module separation; it is not a biological validation or a claim that this architecture is itself new. CODA combines an existing planner–executor interface, GRPO-based planner adaptation, automated trajectory filtering, and specialist-to-generalist consolidation. Its step-level action-matching reward is an executor-mediated surrogate, rather than direct terminal-success optimization. Comparisons of the complete pipeline do not by themselves identify the independent causal effect of each component. Appendix I adds interpretation and evaluation details while retaining the original experiments, figures, and analyses.

This section supplements the existing method and analyses. It does not introduce new experimental measurements.

I.1 Contribution and the Role of the Analogy

Planner–executor decomposition, GRPO, and specialist-to-generalist training are established components. The contribution should be evaluated as their concrete combination for adapting a planner through a fixed executor in the reported software environments. The brain analogy is a design intuition, not evidence for a neuroscientific mechanism or proof of architectural novelty. A fixed executor can still make grounding errors, so improvements in planner behavior do not establish that grounding has ceased to be a bottleneck.

I.2 Interpretation of the Training Signal

The reference action a_T comes from a trajectory accepted by the automatic judge. It is therefore a pseudo-label whose reliability depends on the selection procedure. The reward measures agreement in action type and parameters; different valid actions can receive different scores. This provides an imitation-like signal through the executor. It does not, by construction, establish that a higher reward implies greater long-horizon task success. The existing reward and alternative-action ablations should be interpreted within their stated task sets and compute budgets.

I.3 Evaluation Units and Uncertainty

For domain d with N_d tasks, let $y_{d,j,k}$ indicate success on task j in inference attempt k. The intended domain metrics are

$$\mathrm{Avg@8}_d = \frac{1}{8N_d}\sum_{j=1}^{N_d}\sum_{k=1}^{8} y_{d,j,k}, \qquad \mathrm{Pass@8}_d = \frac{1}{N_d}\sum_{j=1}^{N_d}\mathbb{I}\left[\sum_{k=1}^{8} y_{d,j,k} > 0\right].$$

A macro-average weights domains equally; a micro-average weights tasks equally. The aggregation rule, task counts, checkpoint, and evaluation seeds must accompany a comparison. Pass@8 measures success with repeated attempts and is not a single-attempt reliability estimate. Eight inference attempts are also not eight independently trained models. Task-cluster bootstrap intervals and paired task-level differences can quantify evaluation uncertainty when the underlying outcomes are available; no such intervals are newly claimed here.

I.4 Precision, Selection Bias, and Data Units

The ensemble's ScienceBoard precision of 69.5% means that 30.5% of its accepted predictions are incorrect on that evaluation sample; it does not imply a 30.5% false-positive rate over all negative examples. The displayed Gemini precision is 70.3%, so the ensemble should not be described as having the highest precision on every benchmark. False negatives can change the difficulty and domain composition of the retained corpus, as well as reduce its size. The ARB values 79.0 and 76.8 in the original ensemble discussion concern recall, not precision, and cannot support a precision-drop explanation.

Table 6 lists 1,110 Stage-2 trajectories and 5,650 conversation examples across the four domains. The shorthand "5.6K" should be read in conversation-example units rather than as a count of complete trajectories, subject to reconciliation with the data manifests.

I.5 Controlled Comparisons and Reproducibility

To attribute gains separately to reinforcement learning, filtering, specialization, and consolidation, comparisons should hold fixed the initialization, executor, judge, data access, and total resource budget. Four specialists must be charged their combined interaction and training costs when compared with one mixed-domain planner. A filtered-SFT control should also account for the cost and policy used to collect its candidate trajectories. Comparisons with different corpora or compute budgets measure a complete configuration difference, not an isolated causal effect of reinforcement learning.

Reproduction requires the group size, clipping and KL coefficients, rollout-policy update schedule, token sampling settings, treatment of zero-variance groups, exact optimizer and framework versions, and task-level split manifests. The old rollout policy and the KL reference policy need not be identical; the policy ratio for a token-level objective should be conditioned on the preceding planner tokens. These implementation details must be checked against the actual training configuration rather than inferred from standard GRPO defaults.

I.6 Scope of Transfer

Transfer should distinguish new tasks in a known application, new workflows, new applications, and a replacement executor. Since the judge-training sources include OSWorld, OSWorld transfer is not evidence that the complete pipeline never encountered that benchmark. Code-executor results test a different action interface, and do not alone establish universal transfer. Task-generation records and training manifests should distinguish generated tasks from tasks actually used for training or tuning.