LEARN FROM WEAKNESSES: STUDENT-AWARE DOMAIN SPECIALIZATION FOR SMALL COMPUTER-USE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Small computer-use agents (CUAs) are attractive where cost, latency, or privacy matter, but they lag far behind large models in domain-specific software. We find that scaling up generic domain data yields limited gains, as small students differ in their weaknesses even within the same software. Existing failure-driven methods do adapt to the student, but they draw the reference for correct behavior from the student itself, leaving the failures it cannot yet solve uncorrected. We introduce LEARNWEAK, an annotation-free framework that makes specialization student-aware by contrasting the student with a stronger teacher, which can be a black-box API model. At the task level, it learns from *correctable failures*, tasks that the teacher solves but the student fails, and synthesizes practice that targets the missing skills. At the state level, it replays the student along the teacher's successful trajectories and corrects only the part of its action that deviates, depending on whether the failure stems from planning or execution. Training on only a few dozen trajectories per domain, LEARNWEAK improves EvoCUA-8B and OpenCUA-7B by 11.6 and 11.1 points on average across eight OSWorld domains, lets the 8B student surpass its 32B teacher on two domains where it initially trailed by more than 10 points, and outperforms existing data construction methods, including failure-driven training, under the same data budget. Our analyses further show that specialization data is student-dependent, as data built from another student's failures can even underperform the unadapted model. Together, these results suggest that specializing small CUAs hinges less on scaling data than on diagnosing and correcting the failures of the particular student.

1 INTRODUCTION

Computer-use agents (CUAs) have advanced rapidly. While large proprietary models such as Claude Sonnet 4.6 (Anthropic, 2026) and GPT-5.4 (OpenAI, 2026) lead in capability, small CUAs such as EvoCUA (Xue et al., 2026a) and OpenCUA (Wang et al., 2025) are more practical to deploy, remaining viable on edge devices (Xu et al., 2026; Lin et al., 2025) and in privacy-sensitive settings (Erdogan et al., 2024; Zharmagambetov et al., 2025). However, they still fall far behind large models in domain-specific software (Li et al., 2025a; Yang et al., 2025b; Xie et al., 2024).

Domain specialization is a natural remedy (Sun et al., 2025c; Liu et al., 2026; Sun et al., 2025b; Chen et al., 2025), commonly pursued by scaling up collected or synthesized domain data (Wang et al., 2025; Xie et al., 2025; Sun et al., 2025a; Yang et al., 2025a; He et al., 2025b). Yet scaling static data yields diminishing returns (Xue et al., 2026a), and our analysis indicates that which data is used matters more than how much. Under the same data budget, existing datasets improve the model substantially less than data targeting its failures, and enlarging a human-collected corpus adds little (§4.3). Much of such data covers workflows the model already handles or offers no correct reference: even among tasks generated by exploring the target software, only about one in six is solved by a stronger teacher but failed by the student. Moreover, the workflows a model fails on differ across models, so data built from one model's failures can even hurt another model (§5.1).

This suggests that specialization should be student-aware, letting the failures of the model being specialized, the *student*, determine what it practices and how it is corrected. Recent work already

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.

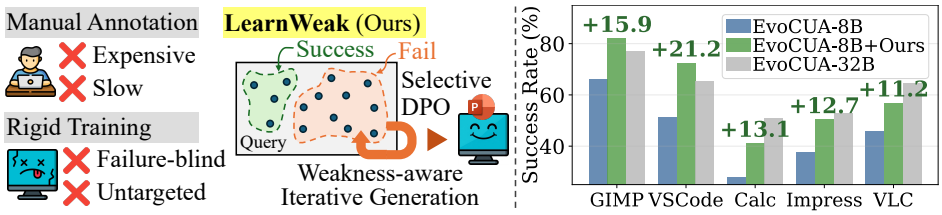


Figure 1: Conceptual illustration of LEARNWEAK, performance gains after domain specialization.

lets failures shape training, adapting curricula via online reinforcement learning (Sun et al., 2025c; Xue et al., 2026b) or correcting where a failed rollout deviates from a successful one (Xue et al., 2026a). Yet not every failure is equally useful, and three gaps remain. (i) The reference comes from the agent itself, so failures on tasks it never solves go uncorrected and cannot be told apart from infeasible tasks. (ii) Feedback to task generation mostly adjusts difficulty rather than targeting the underlying skill. (iii) Correction is poorly matched to the student: a stronger teacher can supply the reference (Lyu et al., 2025), but imitating it imposes a diverging reasoning distribution (Lauffer et al., 2025), on-policy distillation requires white-box teacher access (Agarwal et al., 2024), and few methods distinguish planning from execution failures (Gou et al., 2025; Agashe et al., 2025).

We introduce LEARNWEAK, which closes these gaps by contrasting the student with a stronger teacher on the same tasks and at the same states. At the task level, it learns from *correctable failures*, tasks that the teacher solves but the student fails. These carry a verified reference even when the student has never succeeded, while tasks that neither model solves, including infeasible or ambiguous ones, are excluded. The failures are summarized into skill-level weakness reports that drive the synthesis of practice tasks targeting the missing skills. At the state level, the student is replayed along the teacher's successful trajectory, and its action is compared with the teacher's on the same screen. This reveals whether the student chose the wrong operation or applied the right one incorrectly, and the preference objective updates only the corresponding part of the action, leaving the student's own reasoning untouched rather than imitating the teacher's full output. The pipeline needs only a few seed queries and no human annotation; since the teacher is used only through its actions, it can be a black-box API model, and training runs offline without online reinforcement learning.

On eight OSWorld domains (Xie et al., 2024), LEARNWEAK improves EvoCUA-8B and OpenCUA-7B by 11.6 and 11.1 points on average, bringing the 8B student close to its 32B teacher (62.2% vs. 63.8%), and under the same data budget outperforms the strongest existing data-construction method by 5.6 points and failure-driven EvoCUA training by 9.2 points. Our contributions are threefold. First, we show that specialization should be conditioned on the particular student, not just the domain: data built from the student's correctable failures outperforms generic data even when the latter is enlarged, whereas data built from another student's failures can even underperform the unadapted model. Second, we propose LEARNWEAK, which contrasts the student with a black-box teacher at the task level to learn from correctable failures and at the state level to construct error-aware preference supervision. Third, controlled analyses show that both components contribute: selecting correctable failures outperforms training on all teacher-successful trajectories, and error-aware supervision outperforms SFT, full-response DPO, and fixed-span variants.

2 PRELIMINARIES

A computer-use agent (CUA) is a policy that operates within an interactive software environment by perceiving the screen and issuing actions to complete a given task. At each step t, the agent receives the current screen as a partial observation of the environment state together with the interaction history $h_t = (o_1, a_1, o_2, a_2, \ldots, o_{t-1}, a_{t-1})$, which records all previously observed screens $o_{<t}$ and executed actions $a_{<t}$. Conditioned on the current context $c_t = (q, o_t, h_t)$ where q is the task instruction, the agent policy π produces a structured output,

$$a_t = \pi(c_t) = (r_t,\ s_t,\ e_t). \tag{1}$$

It consists of three components: (i) *internal reasoning* r_t, which reflects the agent's analysis of the current state; (ii) *action description* s_t, a natural language description of the intended action; and (iii) *tool execution* $e_t = (f_t, p_t)$, the executable action that directly manipulates the envi-

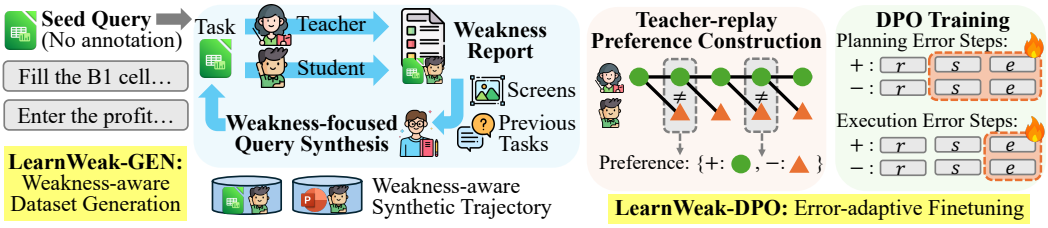


Figure 2: Overview of the LEARNWEAK framework.

ronment, consisting of a function type f_t and its parameters p_t such as `left_click(x,y)` or `type(text)`. The agent repeats this process until the task is complete, producing the full trajectory $\tau = (o_1, a_1, \ldots, o_H, a_H)$.

Problem Formulation. We address *domain specialization*, namely domain-specific finetuning of a broadly capable student policy to a target domain. Let $\mathcal{E}$ be a set of target domains, where each domain $d \in \mathcal{E}$ corresponds to a distinct software application or operating environment. We are given a student policy π^S pretrained on a broad collection of GUI tasks, a stronger teacher policy π^T, a small set of K seed queries $\mathcal{Q}_0^d = \{q_1, \ldots, q_K\}$, and an executable environment equipped with an automatic verifier $V(q, \tau) \in \{0, 1\}$. No human annotation is assumed.

Our problem consists of two coupled stages. In the first stage, we autonomously generate a domain-specific training dataset $\mathcal{D}^d = \texttt{DataGen}(\pi^S, \pi^T, \mathcal{Q}_0^d, V)$ by expanding the seed queries and collecting trajectories from the teacher policy. In the second stage, we train a domain-specialized student on the generated dataset, minimizing a training loss as a surrogate for expected task success on the target domain:

$$\hat{\pi}^{S,d} = \arg\min_{\pi_\theta} \mathcal{L}\big(\pi_\theta;\, \mathcal{D}^d\big). \tag{2}$$

3 METHOD

We first construct a dataset of correctable failures through iterative teacher-student comparison, verification, and weakness-aware query synthesis (Section 3.1), and then convert these failures into step-level preferences for selective DPO training (Section 3.2).

3.1 WEAKNESS-AWARE DATA GENERATION (LEARNWEAK-GEN)

The annotation-free pipeline begins with seed queries, iterates weakness discovery and query synthesis, and aggregates the resulting failures; Appendix A formalizes it.

Seed query setup. For each target domain d, we initialize a small set of executable environment configurations and seed tasks $\mathcal{Q}_0^d$. These initial states are constructed separately from the evaluation benchmark so that data generation does not rely on benchmark-specific assets or leaked task states.

Weakness discovery. Weakness discovery is driven by paired teacher-student execution. For each task $q \in \mathcal{Q}_i^d$ at iteration i, beginning from the seed queries $\mathcal{Q}_0^d$ at $i = 0$, we run a teacher trajectory τ_q^T and a student trajectory τ_q^S in the same environment, where τ_q^S is produced by the fixed pre-adaptation snapshot of student π^S. A verifier V is then applied to both trajectories, yielding binary success outcomes $v_q^T, v_q^S \in \{0, 1\}$ and structured rationales r_q^T, r_q^S. We then collect the *correctable failures* $\mathcal{F}_i^d$, namely the tasks where the teacher is verified to succeed while the student fails:

$$\mathcal{F}_i^d = \{q \in \mathcal{Q}_i^d \mid v_q^T = 1,\ v_q^S = 0\}. \tag{3}$$

Each correctable failure comes with a verified teacher trajectory, and requiring teacher success also excludes generated tasks that are infeasible or ambiguous. Finally, we summarize the verifier diagnostics from these failures into a weakness report $R_i^d = \texttt{Summarize}(\{r_q^S \mid q \in \mathcal{F}_i^d\})$, which captures recurring failure modes in domain d, such as incorrect operation selection, inaccurate element localization, or invalid action arguments.

Screenshot-guided query generation. To generate new queries, we first construct a representative screenshot set S_i^d from both teacher and student trajectories of the current iteration via representation-level clustering and VLM-based reranking, selecting screenshots that are both diverse and semantically informative. We then employ a task-query generator G to synthesize queries for the next iteration, conditioned on previously generated tasks $\mathcal{Q}_i^d$, the current weakness report R_i^d, the selected screenshots S_i^d, and domain-level environment metadata M^d such as available assets. Query synthesis proceeds via two complementary strategies: weakness-focused synthesis, which generates tasks conditioned on the weakness report to target identified deficiencies, and exploration-focused synthesis, which omits the report and instead relies on screenshots to generate tasks covering unexplored functionalities or UI elements. Using both strategies together maintains a balance between student-aware targeting and open-ended domain exploration:

$$\begin{gathered}\mathcal{Q}_{i+1}^{\text{weak}} = G(\mathcal{Q}_i^d, R_i^d, S_i^d, M^d), \qquad \mathcal{Q}_{i+1}^{\text{explore}} = G(\mathcal{Q}_i^d, \varnothing, S_i^d, M^d), \\ \mathcal{Q}_{i+1}^d = \mathcal{Q}_{i+1}^{\text{weak}} \cup \mathcal{Q}_{i+1}^{\text{explore}}.\end{gathered} \tag{4}$$

Iterative generation. Let N denote the total number of generation iterations, including the seed iteration. We perform weakness discovery at each iteration and generate new queries after the first $N-1$ iterations. After all iterations are complete, we aggregate the correctable failures into $\mathcal{F}^d = \bigcup_{i=0}^{N-1} \mathcal{F}_i^d$ and construct the corresponding teacher-student trajectory collection:

$$\mathcal{D}^d(\pi^S) = \{(q, \tau_q^T, \tau_q^S) \mid q \in \mathcal{F}^d\}. \tag{5}$$

3.2 Agent Training for Domain Specialization (LearnWeak-DPO)

We train the student with DPO (Rafailov et al., 2023) on the failure-focused dataset $\mathcal{D}_{\text{pref}}^d$ by adaptively selecting the supervised response span for different failure types. Since trajectory-wise training of CUAs is resource-intensive due to multiple screenshots and long-context reasoning traces, we apply step-level supervision focusing only on steps that require correction, by filtering out those where the teacher and student produce the same tool execution.

Teacher-replay preference construction. For each task $q \in \mathcal{F}^d$, we replay the teacher trajectory step by step. At each step t, we query the student policy π^S using a teacher context $c_t^T = (q, o_t^T, h_t^T)$ and obtain a replayed student response $\hat{a}_t^S \sim \pi^S(\cdot \mid c_t^T)$. If the action executions differ, we build a preference tuple as $a_t^+ = a_t^T, a_t^- = \hat{a}_t^S$ and aggregate them into domain-specific preference dataset:

$$\mathcal{D}_{\text{pref}}^d = \{(c_t^T, a_t^T, \hat{a}_t^S) \mid q \in \mathcal{F}^d,\ t \in \mathcal{T}_q^d\}, \tag{6}$$

where $\mathcal{T}_q^d = \{t \mid e_t^T \neq \hat{e}_t^S\}$ denotes the set of steps at which the teacher and replayed student produce differing tool executions. Since pairs are drawn from teacher-success and student-failure tasks, rejected actions come from tasks the student fails, while chosen actions come from verified successful trajectories.

Error-aware preference optimization. Recall that tool execution e_t is decomposed into a function type f_t and parameters p_t, we define a failure type ϵ_t in two categories: *planning-level error* (ϵ_{PLAN}) when $f_t^T \neq \hat{f}_t^S$ and *execution-level error* (ϵ_{EXEC}) when $f_t^T = \hat{f}_t^S$ but $p_t^T \neq \hat{p}_t^S$. Both types are read directly from the tool calls without additional annotation: the student either chose a different operation or instantiated the intended one incorrectly. Let π_θ denote the trainable student policy and π_{ref} the frozen reference policy initialized from the base student. Each preference example is associated with a binary mask m over the token position j of $a_t = (r_t, s_t, e_t)$, denoted as:

$$m^{(j)} = \begin{cases} 0 & \text{if } a_t^{(j)} \in r_t, \\ g(t) & \text{if } a_t^{(j)} \in s_t, \\ 1 & \text{if } a_t^{(j)} \in e_t, \end{cases} \qquad g(t) = \begin{cases} 1 & \text{if } \epsilon_t = \epsilon_{\text{PLAN}}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Since the chosen and rejected responses may have different token lengths, m denotes the response-wise mask applied to each score term under the same rule. We define the masked action score as

$$s_\theta(c, a_t; m) = \sum_{j=1}^{|a_t|} m^{(j)} \log \pi_\theta(a_t^{(j)} \mid c, a_t^{(<j)}), \tag{8}$$

Table 1: Domain specialization results on OSWorld (mean success rate, %). Yellow marks the teacher; blue marks students with separately trained adapters for each target domain.

	GIMP	Calc	Impress	Writer	OS	Thunderbird	VLC	VSCode	Avg.
Generalized Models									
Kimi K2.6 (Team et al., 2026)	73.08	80.85	82.19	73.91	79.17	80.00	75.71	91.30	79.53
Claude Sonnet 4.6 (Anthropic, 2026)	69.23	74.47	70.21	86.83	91.67	66.67	81.41	72.73	76.65
Qwen3.5-27B (Team, 2026)	39.74	22.70	43.97	52.17	41.67	66.67	44.12	47.83	44.86
Domain Specialized CUA Models									
SEAgent (Sun et al., 2025c)	42.30	-	22.70	31.80	-	-	35.30	40.50	-
OSExpert (Liu et al., 2026)	30.80	44.70	42.60	34.70	-	-	-	-	-
CUA Models									
EvoCUA-32B (Xue et al., 2026a)	76.29	51.06	52.98	65.22	75.00	60.00	64.65	65.22	63.80
OpenCUA-32B (Wang et al., 2025)	74.36	35.46	48.21	56.52	61.11	57.78	37.25	72.73	55.43
EvoCUA-8B	66.15	28.07	37.66	50.43	60.83	65.33	45.71	51.30	50.69
EvoCUA-8B **+ Ours**	82.05	41.13	50.35	55.07	66.67	73.33	56.86	72.46	62.24
Δ	**+15.9**	**+13.1**	**+12.7**	**+4.6**	**+5.8**	**+8.0**	**+11.2**	**+21.2**	**+11.6**
OpenCUA-7B	48.46	11.91	31.49	30.43	40.00	54.67	32.94	51.30	37.65
OpenCUA-7B **+ Ours**	57.69	19.15	36.88	40.58	59.42	66.67	47.06	62.32	48.72
Δ	**+9.2**	**+7.2**	**+5.4**	**+10.2**	**+19.4**	**+12.0**	**+14.1**	**+11.0**	**+11.1**

and define $s_{\text{ref}}(c, a; m)$ analogously using π_{ref}. We then optimize

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}_{(c_t, a_t^+, a_t^-)\sim\mathcal{D}_{\text{pref}}^d}\Big[\log\sigma\Big(\beta\big(s_\theta(c_t, a_t^+; m) - s_\theta(c_t, a_t^-; m) - s_{\text{ref}}(c_t, a_t^+; m) + s_{\text{ref}}(c_t, a_t^-; m)\big)\Big)\Big], \tag{9}$$

where $\sigma(\cdot)$ denotes the logistic sigmoid function and β is a temperature hyperparameter controlling the strength of the preference signal. This objective increases the relative likelihood of teacher actions over replayed student actions while restricting updates to the behaviorally relevant span.

Domain specialization. We freeze the student and only update LoRA (Hu et al., 2022) adapters $\{\Delta^d\}_{d\in\mathcal{E}}$. At deployment time, the base policy π^S is shared across domains, while the adapter corresponding to the current domain is activated to obtain the specialist.

4 Experiments

4.1 Experimental Setup

Evaluation protocol. We evaluate eight OSWorld domains (Xie et al., 2024): `GIMP`, `Libreoffice Calc/Impress/Writer`, `OS`, `Thunderbird`, `VLC`, and `VSCode`. For our evaluation runs, we allow up to 50 interaction steps and report success rates averaged over three evaluations of the same checkpoint using the official task evaluators.

Models and comparisons. EvoCUA-8B and OpenCUA-7B are the main students, with EvoCUA-32B as the teacher. Table 1 places their specialization results alongside three groups: general-purpose models (Kimi K2.6, Claude Sonnet 4.6, and Qwen3.5-27B; official benchmark results), larger CUAs (EvoCUA-32B and OpenCUA-32B; our evaluations), and domain-specialized CUAs (SEAgent and OSExpert; published results). The corresponding base students provide the direct comparison for specialization gains. Unless otherwise specified, data-construction comparisons and component analyses use EvoCUA-8B on `Calc`, `Impress`, `VLC`, and `VSCode`.

Domain specialization setup. For each domain, we start from 10 seed queries across seven training configurations with files and assets separate from the benchmark evaluation setups. GPT-5-mini supports query generation, weakness summarization, screenshot ranking, and trajectory verification. Data generation and LoRA training are performed per domain; at evaluation time, the shared frozen backbone uses the adapter trained for the known target domain. All domains and students share the same LoRA configuration and training hyperparameters. The resulting training sets contain only 22–52 and 27–68 trajectories per domain for EvoCUA-8B and OpenCUA-7B, respectively (Figure A1). Implementation details and prompts are provided in Appendices B and F.

Table 2: Comparison of data construction and training approaches. Retained-trajectory budgets are matched except for full AgentNet.

	Calc	Impress	VLC	VSCode	Avg.
Zero-shot	28.07	37.66	45.71	51.30	40.69
Human-labeled Data					
AgentNet (Wang et al., 2025)	34.04	39.01	49.01	69.57	47.91
AgentNet (Same budget)	32.62	40.43	49.02	63.77	46.46
Seed-Trajectory Expansion					
Trajectory Boosting (He et al., 2025a)	30.50	19.88	45.10	49.28	36.19
Automated Construction					
AgentSynth (Xie et al., 2025)	31.21	39.01	39.22	71.01	45.11
OS-Genesis (Sun et al., 2025a)	31.91	37.59	45.10	68.12	45.68
ZeroGUI (Yang et al., 2025a)	36.17	40.43	48.86	62.30	46.94
EvoCUA (Xue et al., 2026a)	34.02	41.98	45.07	63.05	46.03
WebSTAR (He et al., 2025b)	31.21	40.43	52.94	**73.91**	49.62
LEARNWEAK	**41.13**	**50.35**	**56.86**	72.46	**55.20**

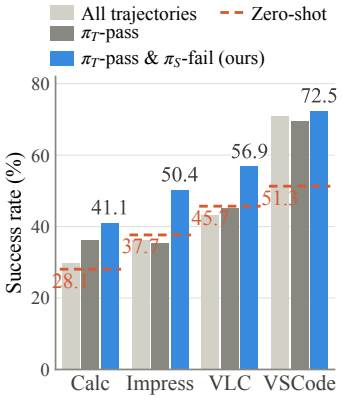


Figure 3: Comparison on different trajectory-selection rules.

4.2 DOMAIN SPECIALIZATION RESULTS

Table 1 shows that LEARNWEAK improves both students in every domain, raising the average success rate of EvoCUA-8B by 11.6 points (50.69 to 62.24) and that of OpenCUA-7B by 11.1 points (37.65 to 48.72). The gains are similar in size despite very different starting points, indicating that the procedure is not tied to a particular backbone. After specialization, EvoCUA-8B closes most of its 13.1-point gap to the 32B teacher (62.24 vs. 63.80), outperforms the larger OpenCUA-32B, and exceeds the teacher on `GIMP` and `VSCode`, where the base student started well below it.

The domains that benefit most differ between the two students: EvoCUA-8B gains most on `VSCode` (+21.2) and `GIMP` (+15.9), whereas OpenCUA-7B gains most on `OS` (+19.4) and `VLC` (+14.1). Because both students are specialized with the same teacher, seed queries, and procedure, this divergence is consistent with LEARNWEAK adapting to each student's own weaknesses rather than to the domain alone, which we test directly in Section 5.1. Although each adapter also transfers to other domains, every tested domain obtains its best result from its own adapter, indicating that the adapter captures domain- and student-specific weaknesses beyond general improvements (§D.1).

4.3 COMPARISON WITH DATA CONSTRUCTION BASELINES

Table 2 compares human-validated data, seed-trajectory expansion, and automated data construction methods for specializing EvoCUA-8B. We follow each method's proposed construction and training procedure and match the final number of training trajectories, except for full AgentNet. For WebSTAR, we apply its filter to teacher trajectories generated by LEARNWEAK-GEN in place of our selection rule, without a student-failure condition, and train with LEARNWEAK-DPO. These comparisons evaluate each construction-and-training procedure. Additional details are in Appendix B.3.

LEARNWEAK achieves the highest average success rate, outperforming the strongest alternative by 5.58 points. Reusing all successful AgentNet trajectories yields only a modest gain over its budget-matched subset, while other automated construction methods also underperform LEARNWEAK. Trajectory Boosting, which expands only the ten seed trajectories, even falls below zero-shot, as its supervision remains concentrated on the seed workflows. Notably, WebSTAR shares our generation and training pipeline and differs only in trajectory selection. Compared with methods that are independent of our pipeline, LEARNWEAK leads by at least 7.29 points.

LEARNWEAK also outperforms failure-based training. EvoCUA constructs critical-deviation preferences from failed student rollouts. With rollout counts, optimizer updates, and processed examples additionally matched, LEARNWEAK improves the average by 9.17 points and outperforms EvoCUA in every domain. Since EvoCUA draws its reference from the student's own successful rollouts, this comparison directly contrasts self-referenced failure correction with learning from correctable failures. These results support combining weakness-aware data construction with teacher-grounded correction.

Table 3: Cross-student comparison of data generated using different source students.

Data source	Target: OpenCUA-7B				Target: EvoCUA-8B			
	Calc	Impress	VLC	VSCode	Calc	Impress	VLC	VSCode
Zero-shot	11.91	31.49	32.94	51.30	28.07	37.66	45.71	51.30
UI-TARS-1.5-7B	7.80	33.35	–	49.28	22.70	31.21	–	71.01
OpenCUA-7B	**19.15**	**36.88**	**47.06**	**62.32**	39.01	43.26	47.06	**73.91**
EvoCUA-8B	9.93	27.54	45.10	50.72	**41.13**	**50.35**	**56.86**	72.46

Table 4: Ablation of iterative generation and weakness reports.

Domain Special.	Iterative Generation	Weakness Report	Calc	Impress	VLC	VSCode	Avg.
✗	✗	✗	28.07	37.66	45.71	51.30	40.69
✓	✗	✗	34.57	39.72	47.06	**73.91**	48.82
✓	✓	✗	24.82	42.55	43.14	72.46	45.74
✓	✓	✓	**41.13**	**50.35**	**56.86**	72.46	**55.20**

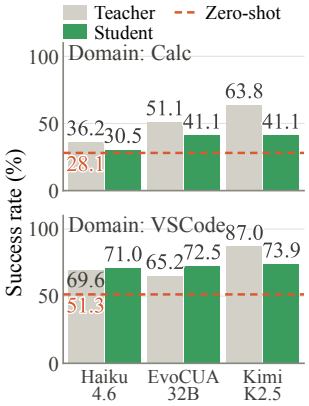


Figure 4: Teacher and specialized-student success rates under different teacher choices (π^T).

5 Analysis

We now analyze the two levels of LearnWeak: (i) whether learning from correctable failures yields better data at the task level, (ii) whether state-level, error-aware correction is more effective. Then, we examine (iii) whether diagnosed weaknesses lead to relevant practice and corrected behavior. Data quality and generalization are analyzed in Appendices C and D.

5.1 What Data Should the Student Learn From?

Task-selection criteria. Figure 3 distinguishes successful teacher demonstrations from demonstrations that address an observed student failure. The all-trajectory condition includes teacher executions regardless of success, whereas the two filtered conditions require teacher success. Retaining teacher-pass trajectories alone provides only a modest average improvement over retaining all trajectories, with mixed effects across domains. Restricting the data to correctable failures yields the strongest result in every domain. Successful teacher execution establishes a reference, while student failure identifies where it is useful. All conditions use the same generated pool with matched optimizer updates; retained-set sizes differ by construction, and the trend is unchanged under equal epoch counts. Such failures are also scarce in generic data: among tasks generated by exploration alone, only about one in six is a correctable failure (§C.1).

Source and target students. Table 3 tests whether data generated for one student is equally useful to another in the same domain. Generation and task selection use the source student, while rejected responses and masks are reconstructed for the target student; optimizer updates are matched. Both targets perform best on average with data generated from their own failures. This pattern holds in every tested domain for OpenCUA-7B, while EvoCUA-8B benefits more from OpenCUA-sourced data on `VSCode`. UI-TARS-sourced data also underperforms each target's own data on the available domains. For OpenCUA-7B, data sourced from EvoCUA-8B even falls below zero-shot in three of the four domains, consistent with supervision built on another student's failures penalizing behavior the target already performs correctly. These results suggest that useful specialization data depends on the target student's weaknesses as well as the software domain.

Iterative generation and weakness reports. Table 4 examines whether repeated generation benefits from feedback about the student's weaknesses. Without weakness reports, iterative generation underperforms one-shot generation on average; with reports, it achieves the strongest average performance. Thus, repetition alone does not explain the benefit. We also vary the number of generation rounds in Figure 5. Because weaknesses are diagnosed against the fixed pre-adaptation student, performance saturates within a few rounds as later rounds revisit already-covered weaknesses; re-diagnosing the adapted student is a natural extension.

Table 5: Comparison of training objectives and response masking.

	Calc	Impress	VLC	VSCode	Avg.
Zero-shot	28.07	37.66	45.71	51.30	40.69
SFT					
Standard (no masking)	29.08	39.72	45.10	68.12	45.51
LEARNWEAK-SFT	34.04	46.81	45.10	69.57	48.88
DPO					
Standard (no masking)	27.66	40.43	49.02	65.22	45.58
$m_j = 1$ if $a_t^{(j)} \in \{s_t, e_t\}$	18.44	17.02	41.18	63.77	35.10
$m_j = 1$ if $a_t^{(j)} \in \{e_t\}$	24.82	39.72	45.10	71.01	45.16
LEARNWEAK-DPO	**41.13**	**50.35**	**56.86**	**72.46**	**55.20**

Table 6: Comparison of preference construction types.

	Calc	VLC
Zero-shot	28.07	45.71
Student-context (c_t^S)	38.13	45.19
Teacher-context (c_t^T)	**41.13**	**56.86**

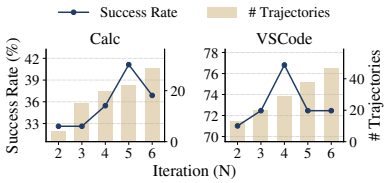


Figure 5: Ablation on the number of generation rounds (N).

Teacher choice. Figure 4 varies only the teacher, matching optimizer updates; Kimi K2.5 and Haiku 4.6 are accessed only through API responses. Although Kimi K2.5 has higher standalone success rates than EvoCUA-32B, both yield similar specialized students, with identical performance on `Calc` and only a small difference on `VSCode`. Haiku 4.6, which is weaker than EvoCUA-32B on `Calc`, yields a weaker specialized student there, while the differences are smaller on `VSCode`. These results suggest that a teacher needs to be sufficient on the states where the student fails rather than strongest overall, and that beyond that point the bottleneck shifts to what the student can absorb.

5.2 How Should the Student Be Supervised?

Training objectives and response masking. Table 5 varies only the training objective and response masks, holding step examples and optimizer updates fixed. Relative to standard DPO (45.58), error-aware masking raises the average to 55.20, while neither fixed span recovers this gain: supervising only the tool execution reaches 45.16, and supervising the action description on every pair drops to 35.10, consistent with penalizing correct plans on execution-level discrepancies. The same masking, applied to the chosen teacher responses, also improves SFT (45.51 to 48.88), but benefits DPO most. The gain therefore comes from matching the supervised span to the failure type, which the state-level comparison with the teacher identifies.

Replay context. Table 6 tests whether corrections could instead be collected at states the student visits, as in DAgger-style labeling (Ross et al., 2011; Lauffer et al., 2025). It compares this student-context construction (c_t^S), where the teacher labels the student's current screenshot given its interaction history, with our teacher-context replay (c_t^T), using the same trajectories, error-aware masks, and optimizer updates. Teacher-context replay performs better on both `Calc` and `VLC` (41.13 vs. 38.13 and 56.86 vs. 45.19). At teacher states, each chosen action lies on a verified successful trajectory, whereas corrections at student-visited states are unverified, suggesting that a verified reference matters more than broader state coverage in this setting.

5.3 Does Weakness Diagnosis Lead to Relevant Practice and Better Behavior?

Report–query alignment. Beyond downstream performance, we verify that generated tasks exercise the diagnosed weaknesses. A domain-stratified audit of the default EvoCUA-8B student and EvoCUA-32B teacher setting checks whether completing each query requires the capability identified in its report. Averaged over two human judges, 92 of 100 pairs are classified as faithful and eight as partially faithful, and two auxiliary-model judges agree closely (§C.2). This supports the intended connection between diagnosis and generated practice.

Case studies. We further compare benchmark trajectories before and after specialization and examine whether the corrected behaviors correspond to diagnosed weaknesses in Appendix G. In the inspected cases, specialization repairs the behaviors identified in the weakness reports.

6 Related Work

Computer-Use Agents and Domain Specialization. Proprietary models such as Claude Sonnet (Anthropic, 2026) and Kimi (Team et al., 2026) lead computer use, while open CUAs such as UI-TARS (Qin et al., 2025), OpenCUA (Wang et al., 2025), and EvoCUA (Xue et al., 2026a) advance end-to-end vision-language-action modeling. Execution-based benchmarks (Xie et al., 2024; Bonatti et al., 2024; Yang et al., 2025b) reveal domain-dependent gaps, especially in productivity software that requires application-specific knowledge (Zhao et al., 2026; Ofengenden et al., 2025). SEAgent (Sun et al., 2025c) and OSExpert (Liu et al., 2026) acquire software-specific skills through self-evolving exploration, and Fara-7B (Awadallah et al., 2025) shows that compact models can be competitive. We instead specialize small CUAs with a stronger, possibly black-box, teacher as the reference, training per-domain adapters without human annotation.

Automated Data Construction for CUAs. AgentNet provides large-scale human-validated trajectories (Wang et al., 2025), and PC-Agent-E expands a small human-trajectory set with stronger models (He et al., 2025a). Annotation-free pipelines compose subtasks (Xie et al., 2025), synthesize instructions from exploration (Sun et al., 2025a), pair generated tasks with automatic rewards (Yang et al., 2025a), convert tutorials and videos into trajectories (Xu et al., 2025; Song et al., 2025), or filter steps (He et al., 2025b), mainly improving data volume, diversity, or quality independently of the learner. Adaptive generation draws on learner experience through capability-aware curricula (Xue et al., 2026b; Sun et al., 2025c), performance-based rollout allocation (Xue et al., 2026a), or gap-driven synthesis (He et al., 2026); beyond CUAs, LLM2LLM (Lee et al., 2024), DataEnvGym (Khan et al., 2025), and Montessori-Instruct (Li et al., 2025b) tailor synthetic data to a student's errors, feedback, or data influence. These methods rely on the agent's own feedback or operate on static tasks with given labels, whereas in interactive software a generated task carries no reference, and a student failure alone cannot reveal whether the task is feasible or what the correct behavior is. LearnWeak conditions synthesis on *correctable failures*, tasks that a stronger teacher solves but the student fails, so every targeted weakness has a verified reference, and grounds generation in target-software screenshots.

Learning from Failures and Teacher Corrections. Imitation of GUI actions suffers from covariate shift, which DAgger (Ross et al., 2011) and on-policy expert corrections (Lauffer et al., 2025) address by labeling learner-induced states; reinforced distillation corrects students with stronger teachers (Lyu et al., 2025), and on-policy distillation trains on the student's own samples under the teacher's token-level distribution, requiring white-box teacher access (Agarwal et al., 2024). Failures also supply preferences through successful versus failed trajectories (Song et al., 2024; Rafailov et al., 2023), the first erroneous reasoning step (Lai et al., 2024), critical deviations between rollouts (Xue et al., 2026a), or fork points of failed experience (Lin et al., 2026). We instead query the student at the teacher's states, comparing actions without successful student rollouts or access to the teacher's internals, so that black-box API models can serve as teachers; this trades learner-state coverage (Ross et al., 2011; Lauffer et al., 2025; Fan et al., 2026) for a verified reference at each step. Unlike learned sparse token masks (Christopoulou et al., 2025), our rule selects the span by discrepancy type, excluding the student's reasoning and supervising the action description only when the student chose a different operation.

7 Conclusion

We study fully automated domain specialization for small computer-use agents, premised on targeting the student's own weaknesses rather than broad synthetic coverage. LearnWeak learns from correctable failures, tasks that a stronger teacher solves but the student fails, and corrects them with selective step-level preference supervision, consistently improving two small CUAs across eight OSWorld domains and outperforming alternative data-construction and training strategies. Our analyses show that domain relevance alone is insufficient, as data built from another student's failures can even fall below the unadapted model, and that a stronger standalone teacher does not necessarily yield a stronger specialist, allowing black-box API teachers. These results frame specialization as targeted diagnosis and repair rather than data scaling, without human annotation. Future work includes automatic adapter routing, multi-application deployment, and combining targeted specialization with online reinforcement learning. Limitations are discussed in Appendix E.

Reproducibility Statement

We describe all components needed to reproduce our results. The data generation procedure is formalized in Appendix A, and Appendix B details the environment configurations, seed queries, models, training hyperparameters, evaluation protocol, and baseline implementations. All prompts are provided in Appendix F, and analyses of the generated data and of generalization are given in Appendices C and D. The code, environment configurations, and seed queries are publicly released.

Ethics Statement

Computer-use agents can manipulate files and application settings, so deployment requires appropriate permissions and safeguards against unintended actions. Our work studies domain adaptation in controlled benchmark environments. Prepared assets and model-generated trajectories may contain biases or errors, and judged success should not be treated as a safety guarantee for deployment.

AI Use Statement

Generative models are used for synthetic task generation, trajectory judging, weakness summarization, and screenshot ranking, as documented in Section 3 and Appendix F.

Beyond the method, AI tools assisted with implementation of standard methods, result visualization, literature discovery, and manuscript editing and grammar check. All AI-assisted outputs were reviewed, revised, and verified by the authors, who made all final decisions and take full responsibility for the claims, numerical results, references, and artifacts in this paper.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations*, 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, 2025. URL https://arxiv.org/abs/2504.00906.

Anthropic. Claude sonnet 4.6 system card, February 2026. URL https://anthropic.com/claude-sonnet-4-6-system-card.

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. Fara-7b: An efficient agentic model for computer use, 2025. URL https://arxiv.org/abs/2511.19663.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows agent arena: Evaluating multi-modal os agents at scale. *arXiv preprint arXiv:2409.08264*, 2024.

Renqi Chen, Zeyin Tao, Jianming Guo, Jingzhe Zhu, Yiheng Peng, Qingqing Sun, Tianyi Zhang, and Shuai Chen. Risk: A framework for gui agents in e-commerce risk management. *arXiv preprint arXiv:2509.21982*, 2025.

Fenia Christopoulou, Ronald Cardenas, Gerasimos Lampouras, Haitham Bou Ammar, and Jun Wang. SparsePO: Controlling preference alignment of LLMs via sparse token masks. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 25477–25503, 2025. URL https://aclanthology.org/2025.findings-emnlp.1389/.

Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Richard Charles Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. TinyAgent: Function calling at the edge. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language*

Processing: System Demonstrations, pp. 80–88, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-demo.9. URL https://aclanthology.org/2024.emnlp-demo.9/.

Zhimin Fan, Hongwei Yu, Yeqing Shen, Haolong Yan, Guozhen Peng, Tianhao Peng, Yudong Zhang, Xiaowen Zhang, Kaijun Tan, Zheng Ge, Xiangyu Zhang, and Daxin Jiang. Skill-guided continuation distillation for GUI agents. *arXiv preprint arXiv:2606.18890*, 2026. URL https://arxiv.org/abs/2606.18890.

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. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

Tiantian He, Yihang Chen, Keyue Jiang, Ka Yiu Lee, Kaiwen Zhou, Kun Shao, and Shuai Wang. EE-MCP: Self-evolving MCP-GUI agents via automated environment generation and experience learning. *arXiv preprint arXiv:2604.09815*, 2026. URL https://arxiv.org/abs/2604.09815.

Yanheng He, Jiahe Jin, and Pengfei Liu. Efficient agent training for computer use, 2025a. URL https://arxiv.org/abs/2505.13909.

Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. Scalable data synthesis for computer use agents with step-level filtering. *arXiv preprint arXiv:2512.10962*, 2025b.

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

Zaid Khan, Elias Stengel-Eskin, Jaemin Cho, and Mohit Bansal. DataEnvGym: Data generation agents in teacher environments with student feedback. In *The Thirteenth International Conference on Learning Representations*, 2025.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-DPO: Step-wise preference optimization for long-chain reasoning of LLMs. *arXiv preprint arXiv:2406.18629*, 2024.

Niklas Lauffer, Xiang Deng, Srivatsa Kundurthy, Brad Kenstler, and Jeff Da. Imitation learning for multi-turn lm agents via on-policy expert corrections. *arXiv preprint arXiv:2512.14895*, 2025.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipalli, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. LLM2LLM: Boosting LLMs with novel iterative data enhancement. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: GUI grounding for professional high-resolution computer use. In Cathal Gurrin, Klaus Schoeffmann, Min Zhang, Luca Rossetto, Stevan Rudinac, Duc-Tien Dang-Nguyen, Wen-Huang Cheng, Phoebe Chen, and Jenny Benois-Pineau (eds.), *Proceedings of the 33rd ACM International Conference on Multimedia, MM 2025, Dublin, Ireland, October 27-31, 2025*, pp. 8778–8786. ACM, 2025a. doi: 10.1145/3746027.3755688. URL https://doi.org/10.1145/3746027.3755688.

Xiaochuan Li, Zichun Yu, and Chenyan Xiong. Montessori-Instruct: Generate influential training data tailored for student learning. In *The Thirteenth International Conference on Learning Representations*, 2025b.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for GUI visual agent. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, pp. 19498–19508. Computer Vision Foundation / IEEE, 2025. doi: 10.1109/CVPR52734.2025.01816. URL https://openaccess.thecvf.com/content/CVPR2025/html/Lin_ShowUI_One_Vision-Language-Action_Model_for_GUI_Visual_Agent_CVPR_2025_paper.html.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, Deheng Ye, and Jie Jiang. UI-Voyager: A self-evolving GUI agent learning via failed experience. *arXiv preprint arXiv:2603.24533*, 2026. URL https://arxiv.org/abs/2603.24533.

Jiateng Liu, Zhenhailong Wang, Rushi Wang, Bingxuan Li, Jeonghwan Kim, Aditi Tiwari, Pengfei Yu, Denghui Zhang, and Heng Ji. Osexpert: Computer-use agents learning professional skills via exploration. *arXiv preprint arXiv:2603.07978*, 2026.

Yuanjie Lyu, Chengyu Wang, Jun Huang, and Tong Xu. From correction to mastery: Reinforced distillation of large language model agents. *arXiv preprint arXiv:2509.14257*, 2025.

Michael Ofengenden, Yunze Man, Ziqi Pang, and Yu-Xiong Wang. Pptarena: A benchmark for agentic powerpoint editing, 2025. URL https://arxiv.org/abs/2512.03042.

OpenAI. Introducing gpt-5.4, March 2026. URL https://openai.com/index/introducing-gpt-5-4/.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

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.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011. URL https://arxiv.org/abs/1011.0686.

Chan Hee Song, Yiwen Song, Palash Goyal, Yu Su, Oriana Riva, Hamid Palangi, and Tomas Pfister. Watch and learn: Learning to use computers from online videos. *arXiv preprint arXiv:2510.04673*, 2025.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents, 2024. URL https://arxiv.org/abs/2403.02502.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. Os-genesis: Automating GUI agent trajectory construction via reverse task synthesis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 5555–5579. Association for Computational Linguistics, 2025a. URL https://aclanthology.org/2025.acl-long.277/.

Zeyi Sun, Yuhang Cao, Jianze Liang, Qiushi Sun, Ziyu Liu, Zhixiong Zhang, Yuhang Zang, Xiaoyi Dong, Kai Chen, Dahua Lin, et al. Coda: Coordinating the cerebrum and cerebellum for a dual-brain computer use agent with decoupled reinforcement learning. *arXiv preprint arXiv:2508.20096*, 2025b.

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*, 2025c.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, SH Cai, Yuan Cao, Y Charles, HS Che, Cheng Chen, Guanduo Chen, et al. Kimi k2. 5: Visual agentic intelligence. *arXiv preprint arXiv:2602.02276*, 2026.

Qwen Team. Qwen3. 5-omni technical report. *arXiv preprint arXiv:2604.15804*, 2026.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, et al. Opencua: Open foundations for computer-use agents. *arXiv preprint arXiv:2508.09123*, 2025.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. *arXiv preprint arXiv:2506.14205*, 2025.

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.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials, 2025. URL https://arxiv.org/abs/2412.09605.

Taofeng Xue, Chong Peng, Mianqiu Huang, Linsen Guo, Tiancheng Han, Haozhe Wang, Jianing Wang, Xiaocheng Zhang, Xin Yang, Dengchang Zhao, et al. Evocua: Evolving computer use agents via learning from scalable synthetic experience. *arXiv preprint arXiv:2601.15876*, 2026a.

Tianci Xue, Zeyi Liao, Tianneng Shi, Zilu Wang, Kai Zhang, Dawn Song, Yu Su, and Huan Sun. Autonomous continual learning for environment adaptation of computer-use agents. *arXiv preprint arXiv:2602.10356*, 2026b. URL https://arxiv.org/abs/2602.10356.

Chenyu Yang, Su Shiqian, Shi Liu, Xuan Dong, Yue Yu, Weijie Su, Xuehui Wang, Zhaoyang Liu, Jinguo Zhu, Hao Li, Wenhai Wang, Yu Qiao, Xizhou Zhu, and Jifeng Dai. Zerogui: Automating online gui learning at zero human cost. *arXiv preprint arXiv:2505.23762*, 2025a.

Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for gui agents. *arXiv preprint arXiv:2506.04135*, 2025b.

Henry Hengyuan Zhao, Kaiming Yang, Wendi Yu, Difei Gao, and Mike Zheng Shou. Worldgui: An interactive benchmark for desktop gui automation from any starting point, 2026. URL https://arxiv.org/abs/2502.08047.

Arman Zharmagambetov, Chuan Guo, Ivan Evtimov, Maya Pavlova, Ruslan Salakhutdinov, and Kamalika Chaudhuri. Agentdam: Privacy leakage evaluation for autonomous web agents. *arXiv preprint arXiv:2503.09780*, 2025.

APPENDIX OVERVIEW

This appendix first specifies the method and experimental setup, then examines the generated data, the generality of specialization, and its limitations, and finally provides prompts and qualitative examples.

- Appendix A: The LEARNWEAK-GEN algorithm.
- Appendix B: Implementation details of LEARNWEAK, the evaluation protocol, and the data-construction baselines.
- Appendix C: Composition and quality of generated data: supervision composition, report–query alignment, and trajectory-judge reliability.
- Appendix D: Generality of specialization: cross-domain transfer, a smaller student, and a different response format.
- Appendix E: Limitations.
- Appendix F: Prompt templates used by LEARNWEAK-GEN.
- Appendix G: Qualitative examples: weakness reports and generated queries, an end-to-end trace, and benchmark behavior before and after specialization.

A DATA GENERATION ALGORITHM

Algorithm 1 formalizes the per-iteration operation of LEARNWEAK-GEN described in Section 3.1.

Algorithm 1 Data Generation Pipeline

Require: Target domain d, teacher policy π^T, fixed student policy π^S, verifier V, task generator G, VLM-based screenshot selector SELECT, domain-level metadata M^d, number of iterations N
Ensure: Aggregated failure task set $\mathcal{F}^d(\pi^S)$ and teacher-student trajectory collection $\mathcal{D}^d(\pi^S)$

Initialize seed task set $\mathcal{Q}_0^d$ and environment configurations for domain d
$\mathcal{D}^d_{\text{raw}} \leftarrow \emptyset$
for $i = 0, 1, \ldots, N-1$ **do** ▷ *Weakness Discovery*
  for each $q \in \mathcal{Q}_i^d$ **do**
    $\tau_q^T \leftarrow \text{RUN}(\pi^T, q)$ ▷ Teacher trajectory
    Student trajectory $\tau_q^S \leftarrow \text{RUN}(\pi^S, q)$ ▷ Student trajectory
    $(v_q^T, r_q^T) \leftarrow V(q, \tau_q^T)$ ▷ Evaluate teacher trajectory
    $(v_q^S, r_q^S) \leftarrow V(q, \tau_q^S)$ ▷ Evaluate student trajectory
  end for
  $\mathcal{F}_i^d = \{q \in \mathcal{Q}_i^d \mid v_q^T = 1,\ v_q^S = 0\}$ ▷ Identify failure set
  $R_i^d \leftarrow \text{SUMMARIZE}(\{r_q^S \mid q \in \mathcal{F}_i^d\})$ ▷ Summarize weakness report
  $\mathcal{D}^d_{\text{raw}} \leftarrow \mathcal{D}^d_{\text{raw}} \cup \{(q, \tau_q^T, \tau_q^S) \mid q \in \mathcal{F}_i^d\}$
  if $i < N-1$ **then**
    $S_i^d \leftarrow \text{SELECT}(\{\tau_q^T, \tau_q^S \mid q \in \mathcal{Q}_i^d\})$ ▷ Collect representative screenshot set
    $\mathcal{Q}_{i+1}^{\text{weak}} \leftarrow G(\mathcal{Q}_i^d,\ R_i^d,\ S_i^d,\ M^d)$ ▷ Weakness-conditioned
    $\mathcal{Q}_{i+1}^{\text{explore}} \leftarrow G(\mathcal{Q}_i^d,\ \varnothing,\ S_i^d,\ M^d)$ ▷ Unconstrained
    $\mathcal{Q}_{i+1}^d \leftarrow \mathcal{Q}_{i+1}^{\text{weak}} \cup \mathcal{Q}_{i+1}^{\text{explore}}$
  end if
end for
$\mathcal{F}^d(\pi^S) \leftarrow \bigcup_{i=0}^{N-1} \mathcal{F}_i^d$
$\mathcal{D}^d(\pi^S) \leftarrow \{(q, \tau_q^T, \tau_q^S) \mid q \in \mathcal{F}^d(\pi^S)\}$
return $\mathcal{F}^d(\pi^S), \mathcal{D}^d(\pi^S)$

B Implementation Details

B.1 Shared Setup

Benchmark-disjoint Configurations. Before generating domain-specific data, we construct training configurations with files and assets separate from those used in benchmark evaluation. Many exploration-based generators, including ZeroGUI (Yang et al., 2025a) and OS-Genesis (Sun et al., 2025a), operate directly on OSWorld configurations that contain benchmark files such as presentation decks and application-specific documents. In contrast, we construct separate training configurations that are disjoint from the original OSWorld evaluation setups, separating generated trajectories from the evaluation assets used in our experiments.

For the main LearnWeak runs, we construct 7 environment configurations and 10 seed queries per target domain. Comparison-specific configuration counts are reported in Appendix B.3. Each configuration covers launching the target software and, when applicable, downloading or loading the necessary files. These configurations are designed to be structurally similar to the original OSWorld setups while containing different files and assets as follows:

- `GIMP`: image files.
- `Libreoffice Calc`: spreadsheet files along with per-sheet data.
- `Libreoffice Impress`: presentation files along with per-slide text content.
- `Libreoffice Writer`: document files along with their textual content.
- `OS`: linux commands (e.g., `mkdir -p /home/user/Project/Project1`).
- `Thunderbird`: downloading and setting up an email profile.
- `VLC`: video files.
- `VS Code`: cloning source code from GitHub.

Based on these configurations, we write 10 simple seed queries per domain. We release the code, all configurations, and seed queries to support reproducibility.

Evaluation. Evaluations that we conduct use the local Docker provider in OSWorld, with a maximum of 50 interaction steps per task. For each evaluated model–domain pair, we evaluate the same checkpoint three times and report the mean success rate using the official OSWorld task evaluators. In Table 1, general-purpose model scores are taken from official benchmark reports, and SEAgent and OSExpert scores from their respective reports; these are not rerun under this protocol. We directly evaluate EvoCUA-32B, OpenCUA-32B, and both base and specialized students. We exclude the `Chrome` domain from our evaluation suite because it exhibited weaker reproducibility and less stable evaluation behavior.

LoRA Fine-tuning. We freeze the vision tower and train LoRA adapters with rank 32 and $\alpha = 64$. We use a visual budget of up to 10^6 image pixels, an effective batch size of 64, a learning rate of 1×10^{-6}, cosine scheduling with 10% warmup, and train the main specialists for 20 epochs. Ablation-specific matching is described below. Each domain has a separately trained adapter on the shared frozen backbone, and evaluation uses the adapter for the known target domain. For the main 7–8B students, LearnWeak LoRA fine-tuning is conducted on a single H200 GPU and takes under 5 hours per domain, depending on data size.

B.2 LearnWeak

Data Generation. Unless otherwise stated, our main data-generation runs use EvoCUA-8B as the student, EvoCUA-32B as the teacher, and GPT-5-mini for verification, weakness summarization, screenshot ranking, and query generation. Each domain uses 10 seed queries defined over 7 benchmark-disjoint configurations, and we run a total of $N = 5$ iterations per domain. For screenshot selection, we reduce the raw screenshot pool with CLIP-based diversity filtering and then keep the top 10 screenshots after GPT-5-mini ranking. For query generation, we issue 2 calls per configuration, with and without the weakness report, and request 3 instructions per call. This yields up to 42 candidate queries per round and up to 168 generated candidate queries per domain across the 4 iterative rounds after seeding. The generation prompts additionally constrain instructions to be short, executable, and compliant with the workspace and path constraints of the current configuration.

Preference Construction. To build the step-wise preference dataset, we parse teacher and replayed student outputs into structured tool calls and compare them after removing `wait` actions. Exact matches are discarded, as are steps that differ only by `wait`. Usable mismatches are split into two cases: parameter differences, mapped to execution-level errors, and action-type or tool-count differences, mapped to planning-level errors. For coordinate-based actions, teacher and student selections are treated as equivalent when they fall within a 20-pixel tolerance. We then train on this dataset with the DPO loss using $\beta = 0.1$.

Component-Analysis Controls. For the objective and masking comparison, all conditions share the same sample extraction and filtering, step examples, epoch count, optimizer updates, and number of processed examples. SFT uses the chosen teacher responses from those examples. LearnWeak-SFT supervises action descriptions and tool executions for planning discrepancies, and tool executions alone for execution discrepancies, as in LearnWeak-DPO. For trajectory-selection ablations, we apply each rule to the same generated pool and match optimizer updates across conditions; retained-data size differs by construction of each rule. The all-trajectory condition may include failed teacher trajectories as supervision. For cross-student comparisons, query generation and task filtering use the source student, while replayed rejected responses and masks are constructed using the target student. Optimizer updates are matched across sources. Teacher-choice comparisons change the teacher while keeping the rest of the procedure fixed; training-data budgets vary, but optimizer updates are matched. All teachers, including Haiku 4.6 and Kimi K2.5 accessed only through API responses, produce responses in the same (r_t, s_t, e_t) format, and reasoning is excluded from the loss.

B.3 Data-Construction Baselines

We implement each baseline following its proposed data-construction procedure and, where specified, its associated training recipe. The common specialization target is EvoCUA-8B; generated-task methods use benchmark-disjoint training configurations and GPT-5-mini for auxiliary generation or verification where required. Method-specific choices are detailed below. All comparisons use retained-trajectory budgets of 22, 52, 32, and 38 for `Calc`, `Impress`, `VLC`, and `VSCode`, respectively, except for the full-data AgentNet condition. These counts refer to trajectories used in final training, whose source and selection criteria depend on the method. EvoCUA additionally matches rollout counts, optimizer updates, and processed training examples. All comparisons use the same LearnWeak checkpoints.

Domain Selection. Under the available compute and time budget, we prioritize the breadth of data-construction baselines over evaluating each on all eight domains. Calc, Impress, and VLC are the three lowest-scoring domains for the base EvoCUA-8B student. We include VSCode instead of the next-lowest domain, Writer, to avoid having three of the four domains drawn from LibreOffice.

AgentNet (Wang et al., 2025). We specialize the student using existing human-validated AgentNet trajectories. The full-data condition uses all successful trajectories in each target domain, while the budget-matched condition randomly samples from that successful pool to match the per-domain trajectory counts above. The full successful pools contain 4,171, 5,013, 145, and 3,049 step-level training examples for Calc, Impress, VLC, and VSCode, respectively; these are step counts, not trajectory counts. Both conditions reuse existing data rather than synthesizing new tasks.

Trajectory Boosting. We adopt the trajectory boosting mechanism introduced in PC-Agent-E (He et al., 2025a). The original procedure constructs training data from a small set of human-annotated trajectories by generating candidate actions for each state. In our implementation, we replace the human-annotated trajectories with 10 teacher trajectories from EvoCUA-32B collected on the seed queries and boost them with eight candidate actions at each state.

AgentSynth (Xie et al., 2025). We follow the official multi-subtask pipeline while replacing the original executor with EvoCUA-32B. Each chain contains 6 subtasks, and each subtask execution is limited to 10 steps. We run the pipeline on 7 configurations per domain, using 2 chains per configuration for `Calc` and 3 chains per configuration for `Impress`, `VLC`, and `VS Code`. This

Table A1: Components of the EvoCUA-style DPO comparison with ours.

Component	EvoCUA-style DPO	LEARNWEAK
Generation	Capability taxonomy, roles, resources	Previous tasks, failure reports, screenshots, resources
Selection	Successful and failed student rollouts	Teacher-pass/student-fail tasks
Preferences	Critical deviations in student trajectories and successful references	Student re-queried at teacher contexts
Loss	Full-response DPO	Selective response-span DPO
Budget controls	Rollout counts, optimizer updates, processed training examples	

yields 84 raw examples for `Calc` and 126 for the other three domains before budget matching. We then retain examples according to the same per-domain budgets used for LEARNWEAK.

OS-Genesis (Sun et al., 2025a). We follow the original four-stage pipeline: environment exploration, reverse task synthesis, clean trajectory recollection, and TRM-based filtering. Exploration and recollection are both executed by EvoCUA-32B. For each domain, we first generate an exploration buffer twice the size of the target budget, synthesize instructions from the resulting exploration trajectories, and then re-execute these instructions to collect clean trajectories. We keep only trajectories with a TRM-style evaluation score of at least 3 and retain the top-scoring examples within the matched per-domain budget. As in the official implementation, we generate both planning and action data for each retained example and train on them jointly in a single supervised stage.

ZeroGUI (Yang et al., 2025a). ZeroGUI consists of two training stages: generated-task training and test-time training. We follow only the generated-task stage and its reward-based training recipe, excluding test-time training from the comparison. Following the official implementation, we generate 10 instructions per round for 20 rounds per domain, yielding 200 candidate tasks per domain. We then randomly sample trajectories to match the same per-domain budget used by LEARNWEAK.

EvoCUA (Xue et al., 2026a). We follow EvoCUA's task-synthesis and critical-deviation preference-training procedure, using GPT-5-mini for task generation and verification. We instantiate the baseline on the same benchmark-disjoint configurations used by LEARNWEAK in each domain. The baseline generates verifiable task specifications from a capability taxonomy, user roles, and configuration resources. Student rollouts are allocated dynamically, and tasks containing both successful and failed executions are used to construct critical-deviation preferences, including action-correction and reflection/recovery examples. Validation, alignment checks, and deduplication are applied before training. For the matched comparison, we control the rollout budget, optimizer updates, and number of processed training examples. Each method uses 336 generation rollouts per domain, before final training-data selection: the EvoCUA-style baseline uses 112 tasks with an average of three student rollouts per task, whereas LEARNWEAK uses 168 tasks with one teacher and one student rollout per task. Optimizer updates and processed examples are matched between the two methods. Table A1 summarizes the matched and method-specific components.

WebSTAR (He et al., 2025b). Since WebSTAR focuses on trajectory filtering given pre-collected trajectories, we apply it to LEARNWEAK-GEN. WebSTAR fully replaces our filtering stage; we do not additionally require student failure. We follow the official implementation for the filtering procedure: each step is augmented with a generated thought, graded on a 0–10 scale, and retained only if the score exceeds 5. After filtering, we use the same teacher-replay preference construction and LEARNWEAK-DPO objective as our method.

C COMPOSITION AND QUALITY OF GENERATED DATA

LEARNWEAK trains on automatically generated tasks whose trajectories are selected by an automatic judge. This section examines that data: how much supervision is retained and which discrepancy types it contains (Appendix C.1), whether generated queries target the reported weaknesses (Appendix C.2), and how reliably the judge identifies successful trajectories (Appendix C.3).

Table A2: Teacher and student outcome counts for 1,344 generated tasks.

Teacher outcome	Student outcome	Tasks	Fraction (%)
Pass	Fail	264	19.6
Fail	Fail	474	35.3
Pass	Pass	454	33.8
Fail	Pass	152	11.3

C.1 Composition of Generated Supervision

Figure A1 summarizes the retained teacher-pass/student-fail trajectories and their planning/execution discrepancy breakdown for the two main student backbones. The amount and composition of supervision vary across domains and students, with some domains dominated by planning discrepancies and others exhibiting a more balanced mixture. These statistics characterize the supervision used to train each domain-specific adapter. Discrepancy types are assigned using the tool-comparison rules described in Appendix B.2.

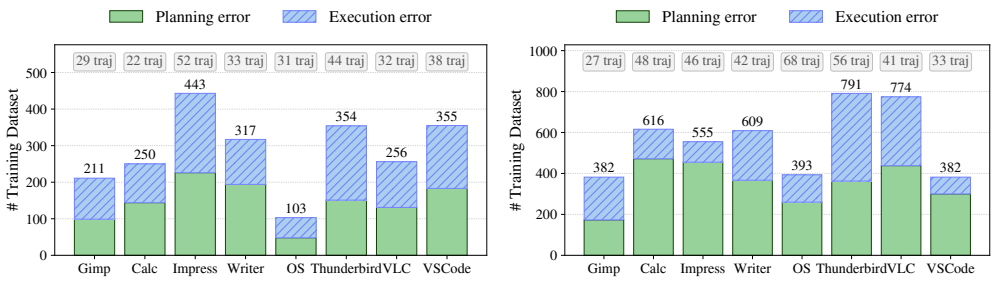
Generated-task outcomes and retention. To quantify how much generated data supplies teacher-supported student failures, we summarize the joint outcomes of teacher and student execution in Table A2. The collection uses EvoCUA-8B as the student and EvoCUA-32B as the teacher and contains 1,344 generated-task evaluations: 42 queries per round over four post-seed generation rounds across eight domains. Each task contributes one joint outcome, based on the automatic verifier, rather than two separately counted rollouts.

The teacher-pass/student-fail rule retains 19.6% of this generated-task collection and excludes 80.4%. The excluded tasks either lack a teacher-successful reference or are already solved by the student.

The two generation branches also differ in the fraction of queries satisfying the retention rule. With equal numbers of queries generated by each branch, the recorded mean teacher-pass/student-fail yield is 22.0% for weakness-conditioned synthesis and 17.3% for exploration-only synthesis.

C.2 Report–Query Alignment and Task Quality

We evaluate whether generated queries faithfully target the weaknesses identified in their corresponding reports. We sample 100 report–query pairs across domains from the default EvoCUA-8B student and EvoCUA-32B teacher setting. Two human judges classify the pairs as faithful, partially faithful, or not faithful; the Human row reports their average category counts. Human and auxiliary-model judgments consistently place most pairs in the faithful category in Table A3. We also assess feasibility and evaluability: human annotators judge 90.5% of queries to be feasible and 94.0% to have an unambiguous evaluation criterion, indicating that the generated queries are generally executable and verifiable.



(a) EvoCUA-8B specialization.

(b) OpenCUA-7B specialization.

Figure A1: Domain-wise statistics of the generated specialization data for each model.

Table A3: Fidelity judgments for report–query alignments (total 100 pairs).

Judge	Faithful (#)	Partial (#)	Not faithful (#)
Human	92	8	0
Claude Opus 5	95	5	0
Gemini 3.5 Flash	96	4	0

Table A4: Training-judge (V) validation for reliability of LEARNWEAK-GEN.

Reference	n (#)	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Human	100	81.0	88.0	77.2	82.2
OSWorld evaluator	636	78.3	84.9	77.8	81.2

For example, an `Impress` report about failing to complete image insertion yields a query that requires inserting and verifying an image, while a `Thunderbird` preference-editing report yields a query that requires adding and verifying a string-valued preference. These examples illustrate how reported weaknesses are converted into concrete practice tasks.

C.3 TRAJECTORY-JUDGE VALIDATION

We validate the trajectory-level judge used to select teacher-pass/student-fail examples against both human annotations and official OSWorld evaluators. Given the task instruction and the agent's response, executed action, and resulting screenshot at each step (prompt in Figure A2), the judge evaluates each trajectory with up to three calls and accepts it only if all returned judgments are pass. For the human-reference audit, we sample 100 pipeline trajectories across eight domains, with 25 from each teacher/student $\times$ predicted-pass/fail group. Two raters independently assess task completion without seeing the verifier judgments; the human-reference metrics in Table A4 are averaged over the two raters. For the automatic-reference audit, we apply the same judging procedure to 636 trajectories from the trained student's benchmark evaluations, covering 215 OSWorld tasks with up to three rollouts per task, and compare against the official task evaluators.

The judge shows conservative agreement with both references, with higher precision than recall, which limits false-pass selections entering teacher supervision. This audit targets trajectory-level selection, which determines the tasks from which step-level preferences are constructed.

D GENERALITY OF SPECIALIZATION

The main experiments specialize two 7–8B students with domain-matched adapters. This section examines how far the approach extends beyond that setting: applying an adapter outside its training domain (Appendix D.1), specializing a smaller 4B student (Appendix D.2), and adapting the objective to a student with a different response format (Appendix D.3).

D.1 CROSS-DOMAIN TRANSFER

Domain-specific training may change behavior outside the target application. To examine this effect, we keep each of four single-domain EvoCUA-8B adapters active while evaluating all eight domains. This differs from the main evaluation, which selects the adapter trained for the known target domain.

Table A5 shows improvement in 25 of the 28 non-target comparisons, while every adapter has a positive average non-target gain. The exceptions occur for the `Impress` adapter on `VLC` and the `VLC`/`VSCode` adapters on `Writer`. Each target domain still obtains its best score from its own adapter among those tested. Thus, the adapters provide benefits beyond their training application, while domain-matched selection yields the best performance.

D.2 SMALLER STUDENT BACKBONE (4B)

To assess whether the approach remains useful below the main students' parameter scale, we specialize Holo3.1-4B on `Calc`, `Impress`, `VLC`, and `VSCode`. The evaluation covers 134 distinct tasks (47, 47, 17, and 23, respectively), with success rates averaged over three runs. Specialization improves this student in all four domains (Table A6), extending the evidence to a smaller backbone.

Table A5: Success rate (%) with each single-domain adapter evaluated across domains. Bold entries identify its training domain.

Model	GIMP	Calc	Impress	Writer	OS	Thunderbird	VLC	VSCode
Base	66.15	28.07	37.66	50.43	60.83	65.33	45.71	51.30
Adapter *Calc*	73.08	**41.13**	42.11	52.16	62.50	66.67	46.15	66.67
Adapter *Impress*	76.92	36.86	**50.35**	50.71	61.10	71.11	44.46	63.77
Adapter *VLC*	75.64	34.04	39.93	47.82	66.67	71.11	**56.86**	71.01
Adapter *VSCode*	78.20	34.04	45.57	49.26	62.50	73.33	46.21	**72.46**

Table A6: Domain specialization of Holo3.1-4B and UI-TARs1.5-7B.

Model	Calc	Impress	VLC	VSCode
Holo3.1-4B	29.79	42.35	40.63	65.22
Holo3.1-4B **+ Ours**	31.91	47.89	47.06	69.57
Δ	**+2.12**	**+5.54**	**+6.43**	**+4.35**
UI-TARS-1.5-7B	7.09	21.98	-	30.43
UI-TARS-1.5-7B **+ Ours**	8.51	22.70	-	40.57
Δ	**+1.42**	**+0.72**	-	**+10.14**

D.3 Different Student Output Format

The selective objective assumes a response with reasoning, action description, and tool execution. UI-TARS-1.5-7B (Qin et al., 2025) provides a compatibility test because it exposes reasoning and tool execution without a separate action-description component. We therefore modify the supervision mask: planning-level errors receive loss on reasoning and execution tokens, while execution-level errors receive loss only on execution tokens. This specifies how the procedure is applied when the main response decomposition is unavailable. The modified procedure improves success in all three evaluated domains (`Calc`, `Impress`, and `VSCode`), with the largest gain on `VSCode` (Table A6). It shows that the framework can be used with a different response format after changing the mask.

E Limitations

Our study has several limitations. First, we assume the availability of a teacher model that provides reasonably reliable guidance within the target domain. If the teacher is highly unstable or systematically biased for a given domain, the resulting supervision may inherit such errors. However, this limitation is not specific to our specialization framework; it reflects a general dependency of teacher-guided offline learning methods on the quality of the supervision source. Second, our specialization framework focuses on domain knowledge and therefore assumes a student base model that already possesses general computer-use skills at least, including visual grounding, action generation, and error recovery. Therefore, it does not guarantee effective improvement for general-purpose models that have not been trained for computer-use tasks, or for students whose failures primarily arise from missing foundational GUI capabilities rather than domain-specific weaknesses.

F PROMPT TEMPLATES

This section shows the main prompt templates used in LEARNWEAK-GEN. We use GPT-5-mini with following prompts during trajectory verification, weakness summarization, screenshot ranking, and query generation.

```
You are a strict verifier for one GUI task.
You will receive the task instruction and an agent step sequence.
For each step, data is ordered as: Thinking/Response -> Executed
    Action -> Screenshot.

## Task instruction
<instruction>

## Your job
1) Analyze the task instruction and set the criteria for task
    completion.
   All tasks in the instruction should be completed to get the pass.
2) Decide if the agent correctly completed the task objective (pass/
    fail).
3) If fail, provide a SHORT reason (3-4 sentences), concrete and
    behavior-focused.
   This should be detailed enough to help the agent improve without
       seeing the trajectory.
   Include which sub-task it failed, which component it did not ground
        correctly,
   or why the progress got stuck.

Return STRICT JSON only, with this exact schema:
{
  "task_completion_criteria": "list of task completion criteria",
  "verification_process": "check the task completion criteria one by
      one based on the trajectory and screenshots.",
  "agent_pass": true or false,
  "agent_failure_analysis": "detailed reason in 3-4 sentences; use
      empty string if agent_pass is true"
}

Rules:
- Pass only if the agent completed all required tasks in the
    instruction correctly.
- Be careful that the model may think the right plan in thought but
    the wrong action in execution.
- Do not trust self-reports like "done", "completed", or DONE action
    by themselves.
- Judge by actual trajectory behavior and screenshot evidence.
- Do not rely on literal PASS/FAIL labels in terminate messages.
- Be concise. English only in JSON values.
```

Figure A2: Trajectory verification prompt.

```
You are analyzing failure patterns of a student UI agent.
Input cases are tasks where TEACHER passed but STUDENT failed.

For each case, you receive:
- instruction
- student_agent_failure_analysis (judge explanation)
```

```
Please produce a concise, high-level report in JSON with this schema:
{
  "overall_summary": "string",
  "failure_categories": [
    {
      "category": "string",
      "what_student_cannot_do": "string",
      "likely_failed_features_or_operations": ["string"]


    }
  ]
}

Requirements:
1) Focus on sub-tasks the agent cannot do reliably.
2) Identify concrete operations the agent misuses or fails to execute.
3) Categories should be notably different from each other.
```

```
4) Group repeated failures into reusable categories.
5) Do not include markdown; return JSON only.
```

Figure A3: Teacher–student weakness summarization prompt.

```
You are evaluating screenshots from a single software domain.
Goal: select the screenshots that maximize understanding of the domain
    's
features and UI components.

You will receive candidate screenshots in this pattern:
Image 0: <image>
```

```
Image 1: <image>
...
Use the number in each "Image N" label as the index.

Select exactly K screenshots.
Prioritize:
1) Coverage of distinct major features/workflows.
2) Diversity of visible UI components/layout states.


3) Informational richness (settings/panels/dialogs/menus/output views)
    .
Avoid near-duplicates and low-information transitional frames.

Return ONLY valid JSON with this schema:
{
  "selected_indices": [int, ...],
  "reasons": [
```

```
    {
      "index": int,
      "reason": "short reason focused on coverage value"
    }
  ]
}
```

Figure A4: Screenshot ranking prompt.

```
Goal:
- Propose new task instructions that specifically improve abilities
    where the student still fails.
- Keep tasks realistic for the given config environment.
- Obey the Workspace / path contract; do not name repos, folders, or
    files
  that this docker setup does not open or download.

- While instructions are grounded in the config and failure analysis,
  they should be diverse and distinct from the prior instructions.
- Do not duplicate or lightly paraphrase the prior instructions.
- Tasks should not be tutorials or step-by-step guides.
- Tasks should be easy, concise, and possible to finish in a few steps
    .

Input context:


1) Prior instructions already used (avoid overlap/paraphrase)
2) Student weakness analysis (teacher pass, student fail)
3) Workspace / path contract
4) Current docker config array to target
5) Extra file/folder/code context from this config (provide_info)

Requirements:
- Generate exactly Y instructions.


- Each instruction must be concise end-user style English.
- Do not include more than two simple and easy sub-tasks.
- Every instruction must satisfy the workspace/path contract.
- Must target one or more weak abilities from the analysis.
- Must be feasible with this config and attached context.
- Instructions should be less than 15 words long.
- Generated tasks should be significantly different from each other.



Return STRICT JSON only:
{
  "queries": [
    {
      "reference_config_id": "string",
      "instruction": "string",
      "targets_student_gaps": ["string"],
      "rationale": "one short sentence"

    }
  ]
}
```

Figure A5: Query-generation prompt `with weakness report`.

```
Goal:
- Propose new task instructions by exploring what appears in
    screenshots.
- Prioritize new features/workflows/subtasks not present in prior
    instructions.
- Keep tasks easy, short, and realistic for the given config
    environment.


- Section 3 (Workspace / path contract) overrides screenshots.
- While instructions are grounded in the config and the screenshots,
  they should be diverse and distinct from the prior instructions.
- Do not duplicate or lightly paraphrase the prior instructions.
- Tasks should not be tutorials or step-by-step guides.
- Tasks should be easy, concise, and possible to finish in a few steps
    .



Input context:
1) Prior instructions already used (avoid overlap/paraphrase)
2) Student weakness analysis: (Not used in this run.)
3) Workspace / path contract
4) Current docker config array to target
5) Extra file/folder/code context from this config (provide_info)

Requirements:


- Generate exactly Y instructions.
- Each instruction must be concise end-user style English.
- Do not include more than two simple and easy sub-tasks.
- Every instruction must satisfy the workspace/path contract.
- Must maximize diversity and novelty versus prior instructions.
- Each query object must include only reference_config_id, instruction
    , and rationale.
- Instructions should be less than 15 words long.


- Generated tasks should be significantly different from each other.

Return STRICT JSON only:
{
  "queries": [
    {
      "reference_config_id": "string",
      "instruction": "a short string",
      "rationale": "one short sentence"
    }
  ]
}
```

Figure A6: Query-generation prompt `without weakness report`.

G QUALITATIVE EXAMPLES

These examples make different stages of the procedure inspectable. The report–query examples show how a diagnosed weakness is converted into practice; the `Writer` trace follows diagnosis through task synthesis and preference construction; the benchmark traces show local decisions before and after specialization.

G.1 WEAKNESS REPORT AND SYNTHETIC QUERY RESULTS

The `Calc` examples allow the reader to inspect which operation a report identifies and how the subsequent queries exercise it. Each example pairs an excerpt from the report with generated instructions and a short explanation of their connection. These concrete instances complement the alignment audit by exposing the content behind its categories.

W1 Weakness Report

Teacher-pass / student-fail evidence

Task 41. Create a department summary listing each department and its average salary.
Student failure analysis. The agent never produced the required summary table or any aggregation formulas. The actions are limited to repeatedly selecting cells and columns without extracting unique departments or entering `AVERAGE` formulas or creating a pivot table. In other words, the sub-tasks of computing averages and writing the summary were not attempted or completed.

Model-level analysis

Failure category
- **Unable to compute aggregations or produce summary tables.** The student cannot aggregate data to produce a department summary, such as generating a unique department list, computing average salary values, or creating a PivotTable to present the result.

W1 Synthetic Query

Target gap: Unable to compute aggregations or produce summary tables
- In `ACCOUNT_INFO.xlsx`, place the record count in F1 and the average AMOUNT in F2.
- Create a PivotTable showing Payment types as rows and Average of Total on a new sheet.
- Insert a PivotTable that sums Population by State on a new sheet.

Analysis: This weakness is not about a single missed click. The report reveals that the student lacks a reusable pattern for constructing summary tables: it neither extracts grouping keys nor applies aggregation operators. The generated queries therefore cover both lightweight formula-based aggregation and PivotTable construction, creating multiple training opportunities for the same abstract weakness.

Figure A7: Weakness Report and Synthetic Queries: Example #1

W2 Weakness Report

Teacher-pass / student-fail evidence

Task 12. Freeze the header row and set the used range as the print area.
Student failure analysis. The agent did not complete the print-area subtask: after selecting the used range there is no evidence it set the range as the print area. The agent also applied a freeze, but the freeze line is below row 2 rather than immediately below the header row, so the header-only freeze was not precisely applied.

Task 16. Freeze the header row and first column in `Order_Details.xlsx`.
Student failure analysis. The agent failed to select the required freeze origin (`B2`) before applying the Freeze command. The Name box in the screenshots shows `A1` throughout, so the wrong cell remained active and the required row-and-column freeze could not be applied reliably.

Model-level analysis

Overall summary. Across these cases, the student UI agent reliably navigates and clicks cells but fails at higher-level spreadsheet operations that require setting the correct active selection and invoking specific commands or dialogs.
Failure categories
- **Failure to apply print-area settings after range selection.** The student can select a used range but fails to execute the Print Area command to define that range as the sheet's print area.
- **Incorrect freeze/pane origin selection.** The student cannot freeze exactly the intended rows and columns because it does not activate the correct origin cell before applying Freeze operations.

W2 Synthetic Query

Target gap: Failure to apply print-area settings after range selection
- Set print area to A1:D12 and confirm via Print Preview.
- Set columns A:D as the print area and open Print Preview.

Target gap: Incorrect freeze/pane origin selection
- Freeze panes so rows 1–2 and columns A–B remain visible when scrolling.
- Select cell C2 then freeze panes so top rows and left columns remain visible.

Target gap: Unable to compute aggregations or produce summary tables
- In `ACCOUNT_INFO.xlsx`, place the record count in F1 and the average AMOUNT in F2.
- Create a PivotTable showing Payment types as rows and Average of Total on a new sheet.

Analysis: The weakness report identifies two operational failures: the student does not commit the selected range as a print area, and it applies freeze panes from the wrong active cell. The generated synthetic queries then isolate those exact operations into short practice tasks, repeatedly requiring explicit print-area definition or a specific freeze origin such as `B2` or `C2`.

Figure A8: Weakness Report and Synthetic Queries: Example #2

W3 Weakness Report

Teacher-pass / student-fail evidence

Task 15. Sort the `Population_of_India` sheet by Population in descending order.
Student failure analysis. The agent selected only columns B and C (Population and Area) before sorting, omitting the State column, so the sort operated on a partial range rather than the full table. This risks misaligning states from their population and area values because whole rows were not moved together.

Task 37. Add column `Quantity_x2` after Quantity and fill it with `=Quantity*2` for all rows.
Student failure analysis. The header was added correctly, but the agent overwrote the formula cells by pasting the header text into the data rows.

Model-level analysis

Overall summary. Two distinct, repeatable failure modes emerged: the agent does not reliably select the correct cell ranges before table-level operations, and it does not consistently propagate formulas without corrupting neighboring cells.
Failure categories
- **Incorrect range selection before table operations (sorting/filtering).** The student fails to select the full table, including all relevant columns and headers, before applying sort or similar operations.
- **Incorrect copy/paste or fill propagation of formulas.** The student struggles to propagate formulas correctly without overwriting or corrupting surrounding cells.

W3 Synthetic Query

Target gap: Incorrect range selection before table operations
- Sort entire table by country A-¿Z then AMOUNT largest first.
- Select entire Customers table and sort rows by PostalCode ascending, keeping all columns aligned.
- Select entire OrderDetails table (A1:D301) then sort by OrderID then ProductID.

Target gap: Incorrect copy/paste or fill propagation of formulas
- Add AMOUNT_USD in E2 =D2*0.013 and fill down to row12.
- In Customers.xlsx add “Tax” column; enter =E2*0.1 in row2 then fill down with fill-handle.
- Copy E2 then use Paste Special -¿ Formulas into range E3:E301.

Analysis: This is one of the clearest weakness-to-query links in our pipeline. The weakness report pinpoints a structural error: sorting was attempted after selecting only a subset of columns. The synthetic queries then repeatedly require whole-table selection before sorting, which later reappears in Case Study C1 as a concrete before–after behavior change.

Figure A9: Weakness Report and Synthetic Queries: Example #3

G.2 An End-to-End Writer Example

The Writer trace illustrates the two uses of teacher–student comparison in one construction example (Table A7). A failure to restrict page orientation to the requested scope informs a subsequent scope-control task, while replay on the retained task supplies action-level preference pairs. Reading the stages together distinguishes generating a new practice task from constructing supervision on an existing one.

Table A7: Recorded Writer example linking diagnosis and supervision.

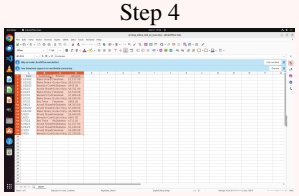
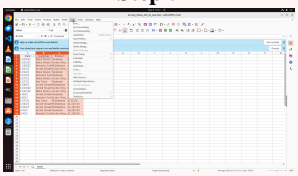
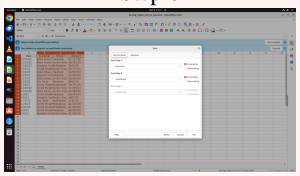
Stage	Evidence
Task	Set the first page of Analysis_101_DTM.docx to landscape.
Executions	The teacher changes orientation and saves, receiving a pass. The student changes Default Page Style but does not establish that the change is limited to the first page.
Diagnosis	Incorrect scope/style application: a global style change does not establish the requested page-level scope.
Next query	Insert a section break before page two and set only that section to landscape.
Replay pairs	The task produces six pairs. One contrasts seeking a manual-break operation with returning to Format; another contrasts selecting Landscape in the open dialog with clicking the Format-menu location.
Interpretation	The report conditions a new scope-control task, while replay provides candidate action preferences on the retained task.

G.3 Benchmark Behavior before and after Specialization

Task success alone does not show which local decisions changed after training. The `Calc` and `Impress` cases therefore compare benchmark trajectories before and after specialization, pairing the visible screenshot with the model response and an action summary at each displayed step. The reasoning text is omitted for brevity. The sequences let the reader inspect the operation attempted and its observed consequence; the range-selection case supplies the detailed trace for the main paper's qualitative example.

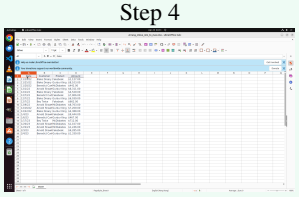
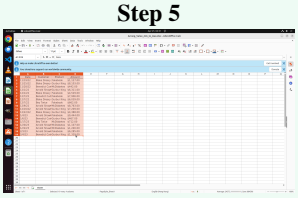
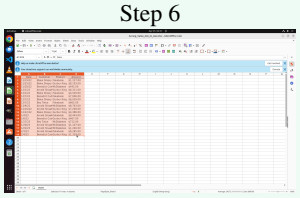
Instruction: Could you help me to sort the records accoring to the amounts ascendingly?

Before Specialization

Step 4	**Step 5**	Step 6
		
Clicked on cell B1 (Customer column) and selected the range from the current cell to the last populated cell in column D. `HotKey(ctrl, shift, end)`	The data range B1:D19 is now selected, with "Selected: 19 rows, 3 columns" shown in the status bar. `Click(455,75)`	Opened the Data menu and proceeded to the Sort dialog. `Click(480,99)`

The baseline model selects only columns B–D, so the sort is applied to a partial table rather than the full sheet range.

After Specialization

Step 4	**Step 5**	Step 6
		
Clicked on cell A1 and selected the entire data range, including all columns (Date, Customer, Product, Amount) and all rows with data. `KeyDown(Shift), Click(474,694), KeyUp(Shift)`	The data range A1:D19 is now selected, with "Selected: 19 rows, 4 columns" shown in the status bar. `Click(455,75)`	Opened the Data menu and proceeded to the Sort dialog. `Click(480,99)`

The specialized model explicitly expands the selection to A1:D19 before proceeding to sorting.

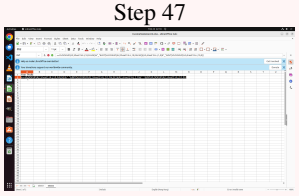
Analysis: The decisive difference is the selected range. The failing baseline trajectory sorts only `B1:D19`, excluding the leftmost column and therefore violating the full-table sorting requirement. The specialized model instead selects `A1:D19`, preserving row integrity during sorting. This behavioral change matches the step-4 weakness category shown above.

Figure A10: Case Study #1 (Domain: `Libreoffice Calc`)

Instruction: Help me fill in the Gross profit column by subtracting all the available expenses including discounts, allowances, material and labor charges, and overhead from the actual sale, i.e., the sales after deducting the returns. Then under column A named "Year_Profit" in a new sheet "Sheet2", display the Year Column in Sheet 1 as text appended by a "_" with the corresponding integer digits of Gross Profit value.

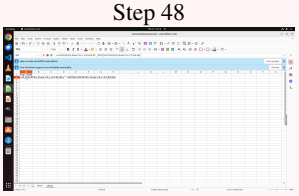
Before Specialization

Step 47



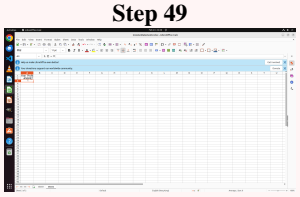
Selected the formula bar contents and prepared to replace them with a lookup-style Year_Profit formula.
`HotKey(ctrl,a)`

Step 48



Typed a long `VLOOKUP`-based concatenation formula into cell A2.
`Type(=VLOOKUP(...)&"_"&)`
`INT(VLOOKUP(...)))`

Step 49

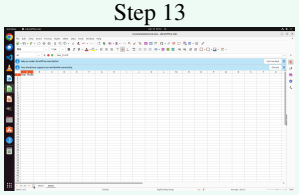


Executed the formula, which resulted in a `#NAME?` error in `Sheet2!A2`.
`Key(enter)`

The baseline commits to an unnecessarily complex lookup-based construction and ends with a `#NAME?` error instead of a usable row-wise formula.

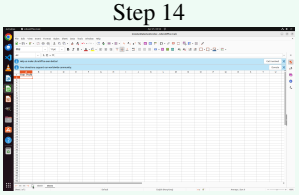
After Specialization

Step 13



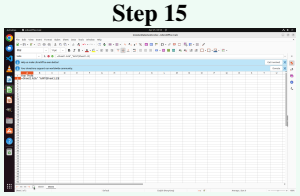
Entered the header `Year_Profit` in `Sheet2!A1`.
`Type(Year_Profit)`

Step 14



Confirmed the header and moved to the next row for formula entry.
`Key(enter)`

Step 15



Entered a direct row-wise formula, =Sheet1.A2&"_"&INT(Sheet1.J2), and produced a concrete value in `A2`.
`Type(=Sheet1.A2&"_"&`
`INT(Sheet1.J2))`

The specialized model uses the simplest formula consistent with the task: a direct row-wise reference that immediately yields a valid `Year_Profit` value.

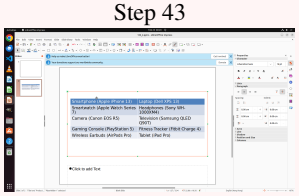
Analysis: The baseline model chooses an unnecessarily complex `VLOOKUP`-based formula, repeatedly edits it, and eventually produces a `#NAME?` error. The specialized model instead uses a direct row-wise reference, `=Sheet1.A2&"_"&INT(Sheet1.J2)`, and then propagates it down the column. The improvement is therefore not just a successful outcome but a better solution strategy: it replaces brittle lookup logic with the simplest reference pattern compatible with the task.

Figure A11: Case Study #2 (Domain: `Libreoffice Calc`)

Instruction: Move the title of page 2 to the bottom of the slide.

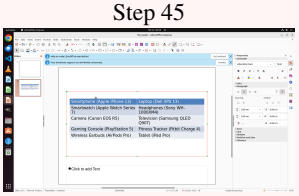
Before Specialization

Step 43



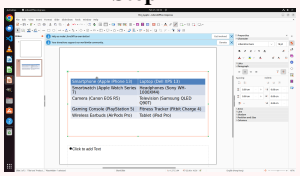
Repeatedly clicked on empty slide space in an attempt to clear the current selection before moving the title.
`Click(599,449)`

Step 45



Stayed stuck in the same deselection loop while the title remained near the top of the slide.
`Click(599,449)`

Step 47

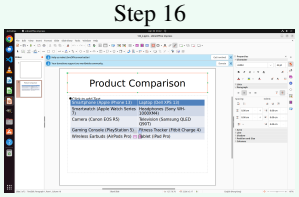


Still attempted the same deselection action instead of re-entering a valid object-move workflow.
`Click(599,449)`

The baseline falls into a long recovery loop, repeatedly clicking empty slide space without returning to a successful object-move interaction.

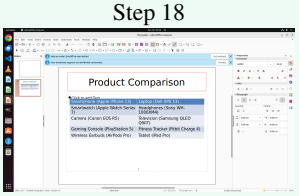
After Specialization

Step 16



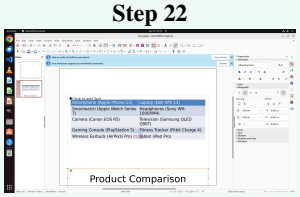
Explicitly exited text-edit mode after recognizing that direct dragging was blocked by the current interaction state.
`Key(esc)`

Step 18



Dragged the title box downward once object handles were visible again.
`DragTo(882,920)`

Step 22



Performed the final downward drag that places the title below the table at the bottom of the slide.
`DragTo(428,920)`

The specialized model recovers by leaving text-edit mode and then resumes the correct object-drag sequence until the title reaches the bottom region.

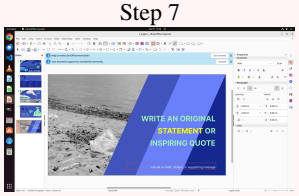
Analysis: The core difference is recovery from an incorrect interaction mode. The baseline remains trapped in a repeated deselection loop and never returns to a successful move operation. The specialized model instead recognizes that it is still in text-edit mode, exits that state with `Esc`, and then completes the object-level drag needed to reposition the title.

Figure A12: Case Study #3 (Domain: `Libreoffice Impress`)

Instruction: Navigate to slide 5 and set the font color of all textboxes to yellow. Use exactly 'yellow'—no variations such as light yellow, dark yellow, or any other color.

Before Specialization

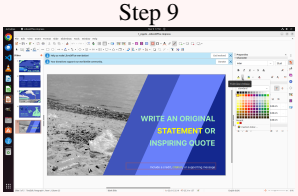
Step 7



Switched from the title to the subtitle after recoloring only part of the slide text.

`Click(1143,916)`

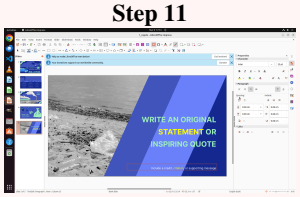
Step 9



Opened the yellow color picker for the subtitle text.

`Click(1570,330)`

Step 11



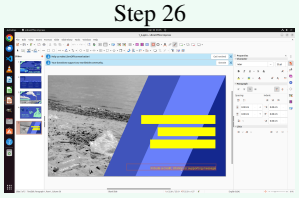
Terminated once the title and subtitle were yellow, without covering the remaining text elements on the slide.

`DONE`

The baseline stops after recoloring two obvious textboxes, leaving the broader "all textboxes" requirement only partially satisfied.

After Specialization

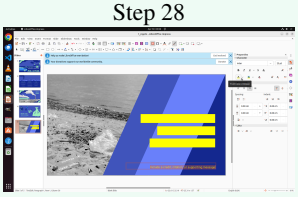
Step 26



Re-selected the subtitle textbox and explicitly selected all text again to ensure the intended edit target was active.

`HotKey(ctrl,a)`

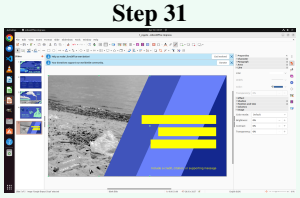
Step 28



Applied yellow through the font-color control after returning to the correct text-formatting context.

`Click(1564,330)`

Step 31



Continued until the full slide satisfied the instruction, including the remaining yellow bar text elements.

`DONE`

The specialized model continues beyond the first successful edits and only finishes once all required text elements on slide 5 are yellow.

Analysis: This example highlights improved task-completion coverage rather than a single local operation. The baseline performs the obvious recoloring steps and then terminates early. The specialized model instead keeps checking whether the entire slide matches the instruction, re-enters the text-formatting context when needed, and finishes only after all required textboxes are yellow.

Figure A13: Case Study #4 (Domain: `Libreoffice Impress`)