Skill-Guided Continuation Distillation for GUI Agents

Zhimin Fan[1,2,*] **Hongwei Yu**[1,3,*] **Yeqing Shen**[1,†] **Haolong Yan**[1]
Guozhen Peng[1] **Tianhao Peng**[4] **Yudong Zhang**[5] **Xiaowen Zhang**[3,‡]
Kaijun Tan[1] **Zheng Ge**[1] **Xiangyu Zhang**[1] **Daxin Jiang**[1]
[1]StepFun [2]Beihang University [3]University of Science and Technology Beijing
[4]Nanyang Technological University [5]Tsinghua University
[*]Equal contribution. [†]Project lead. [‡]Corresponding author.

Abstract

Improving GUI agents typically relies on behavior cloning on expert trajectories. However, as the current policy deviates from the expert policy, it inevitably encounters policy-induced off-trajectory states during closed-loop execution, i.e., states that fall outside the expert trajectories. Since expert trajectories provide no demonstrations for these unseen states, such states receive no effective supervision, leaving the policy unable to select the correct action. To close this supervision gap, we propose Skill-Guided Continuation Distillation (SGCD), an iterative self-improvement framework. SGCD first runs the plain policy without skill guidance for a few steps to reach realistic off-trajectory states. From these states, a skill-guided policy then completes the task and produces successful continuations, which are mixed with expert trajectories to supply supervision over policy-induced off-trajectory states. The skills are extracted from both successful and failed rollouts, consisting of Continuation Plans, Critical Targets, Failure Traps, and Success Criteria. On OSWorld-Verified, SGCD improves the success rate of three base models from the low-30% range to over 50%, demonstrating its effectiveness and generality.

1 Introduction

Built on recent vision-language foundation models (Google DeepMind, 2026; Bai et al., 2025; Yan et al., 2025; OpenAI, 2026; Anthropic, 2026b; Hurst et al., 2024), GUI agents perceive screen observations and predict actions to operate desktop, web, and mobile interfaces in a closed loop, supporting open-ended computer tasks such as document editing, software operation, and web navigation. Such agents are typically trained by supervised fine-tuning (SFT) over trajectory data, which adapts the underlying vision-language models to GUI-specific observations, action spaces, and interaction protocols.

Existing end-to-end GUI agents are trained on human or synthetic expert trajectories (Qin et al., 2025; Wang et al., 2026b; Yan et al., 2025; Xu et al., 2026; Xue et al., 2026), teaching task-specific behaviors, action formats, and procedural knowledge. Self-improvement methods further expand the training pool by converting model-generated rollouts into supervision through filtered self-training, sandbox-based reinforcement learning, or experience-driven knowledge refinement (Yan et al., 2025; Wu et al., 2026; Lai et al., 2025; Zhang et al., 2025; Lin et al., 2026; Wang et al., 2026c). Despite differences in data sources, these approaches share a common supervision paradigm: behavior cloning on successful expert trajectories, where the policy is trained to imitate the action taken at each visited expert state. While such supervision is strong on the expert state distribution, the discrepancy between the expert policy and the current policy inevitably drives the current policy into states that fall outside the expert trajectories (Lauffer et al., 2025; Ross et al., 2011). We refer to these states as **policy-induced off-trajectory states**. Expert trajectories provide no effective supervision for these states, leaving the policy unable to predict the correct action. Reinforcement learning is explored as an alternative source of supervision for such states (Lai et al., 2025; Li et al., 2025), but rollouts from the current policy rarely produce correct actions, yielding sparse reward signals and inefficient training (Zeng et al., 2025; Wang et al., 2026a).

Unlike single-step prediction, where each output is independent, end-to-end tasks involve long sequences of actions executed in a closed loop. An early off-trajectory action thus propagates through subsequent interactions and drives the agent into states increasingly far from the expert trajectories (Chen et al., 2025). Importantly, such states are not arbitrary perturbations of expert states but reflect systematic biases of the learned policy, which

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 33355–33373
October 24-29, 2026 ©2026 Association for Computational Linguistics

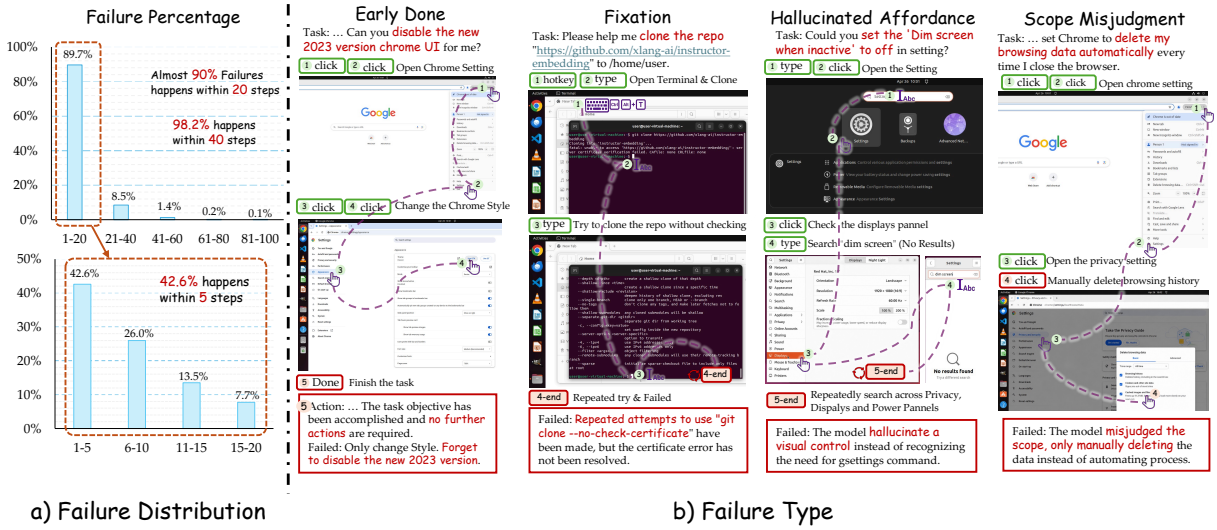

Figure 1: Failure analysis of GUI agents. Left: failures are concentrated in early execution. Right: representative recurring failure patterns, including early done, fixation, hallucinated affordance, and scope misjudgment.

tends to repeat a small set of erroneous behaviors. We refer to this distributional and supervisory mismatch as the off-trajectory supervision deficit.

Closing this supervision deficit is particularly challenging in GUI domains. Reaching realistic off-trajectory states requires actually executing actions in the environment, making such states costly to revisit, reproduce, or reset. Existing methods rely on hand-crafted rules to select important off-trajectory states (Lin et al., 2026), but such heuristics introduce selection bias and yield only sparse coverage of the states the current policy traverses. Moreover, obtaining successful continuations is difficult, as the current policy seldom completes the task from such states. An effective method for supplying such supervision should therefore ① **expose the policy to realistic off-trajectory states** and ② **obtain successful continuations** from such states.

To meet both objectives, we propose Skill-Guided Continuation Distillation (SGCD), an iterative self-improvement framework. SGCD rolls out the plain policy (i.e., the current policy without skill guidance) to reach realistic off-trajectory states, then invokes a skill-guided policy to complete the task and produce successful continuations from those states. Mixing verified continuations with expert trajectories for training supplies additional supervision over policy-induced off-trajectory states. Concretely, each objective is realized as follows.

For objective ①, we first analyze where GUI failures occur during trajectories. As shown in Fig. 1(a), failures are strongly concentrated in early execution, suggesting that off-trajectory deviations are often induced early and can lead to erroneous subsequent actions and eventual task failure. Accordingly, we induce off-trajectory states from the early execution of the plain policy, where such deviations naturally arise. For each task, the plain policy interacts with the GUI for k steps to instantiate realistic off-trajectory states. By sweeping k over a range of values, SGCD avoids hand-crafted state-selection heuristics and densely covers the off-trajectory states the current policy actually traverses. This aligns the supervision distribution with the states the policy encounters at deployment, directly mitigating the distributional shift inherent in behavior cloning.

For objective ②, we analyze the structured failure patterns induced by the learned policy. As illustrated in Fig. 1(b), policy failures exhibit recurring error tendencies rather than isolated accidental mistakes (Wanyan et al., 2026; Lù et al., 2025). From successful and failed rollouts, we extract *off-trajectory continuation skills* (Continuation Plans, Critical Targets, Failure Traps, and Success Criteria), which guide a skill-guided policy to roll out from each off-trajectory state and produce verified successful continuations for training. By fulfilling these two objectives, SGCD synthesizes effective supervision over policy-induced off-trajectory states, closing the supervision deficit inherent in behavior cloning on expert trajectories.

We evaluate our method on OSWorld-Verified across three vision-language models: Qwen3-VL-

8B, Qwen3-VL-30B-A3B, and STEP3-VL-10B. Across all three models, SGCD consistently improves performance from the low-30% range to over 50%. The main contributions are as follows:

- We identify the **off-trajectory supervision deficit**, where agents trained on successful demonstrations lack supervision for policy-induced off-trajectory states, and show that failures are concentrated in early stage.
- We propose **Skill-Guided Continuation Distillation**, which uses off-trajectory continuation skills to collect successful continuations from off-trajectory states and mitigate the expert-state bias.
- We validate SGCD on OSWorld-Verified across three base models, with success rates improving from the low-30% range to over 50%, demonstrating its generality.

2 Related Work

GUI Agents. Recent GUI agents advance on web, mobile, and desktop benchmarks (Deng et al., 2023; Zhou et al., 2024; Rawles et al., 2025; Xie et al., 2024) along several directions. UI-TARS (Qin et al., 2025) unifies perception, reasoning, and action generation through large-scale GUI-specific pretraining. OpenCUA (Wang et al., 2026b) scales human-annotated desktop trajectories with reflective state-action conversion to support open-ended tasks. SeeClick (Cheng et al., 2024) and UGround (Gou et al., 2025) target element grounding via screen-localized pretraining for accurate UI localization. EvoCUA (Xue et al., 2026) and LiteGUI (Wu et al., 2026) automatically synthesize task and trajectory data to continuously update the policy. These works establish strong foundational capabilities for GUI agents and demonstrate effective data synthesis pipelines. Building on this foundation, SGCD proposes an iterative self-improvement approach that further addresses the lack of expert supervision for policy-induced off-trajectory states.

Self-Improvement. Self-improvement methods seek to enhance agent performance by leveraging the agent's own interaction experience as a training signal. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use inference-time verbal feedback to revise outputs. Recent GUI pipelines convert model-generated rollouts into supervision through filtered self-training (Yan et al., 2025; Wu et al., 2026), sandbox-based reinforcement learning (Lai et al., 2025), experience-driven knowledge refinement (Zhang et al., 2025; Lin et al., 2026), or policy-aligned experience assimilation (Wang et al., 2026c). SGCD follows this paradigm and specifically targets policy-induced off-trajectory states, synthesizing skill-guided continuation supervision from the states the current policy actually traverses. Detailed related work is shown in Appendix C.

3 Preliminaries

We consider a distribution of executable GUI training tasks $\mathcal{X}$ constructed to be compatible with the OSWorld-Verified interaction protocol. Each task $x \in \mathcal{X}$ contains a natural-language instruction, an initial environment state, and an automatic verifier. The environment supports state reset, execution of mouse and keyboard actions in real desktop applications, and rule-based final-state evaluation.

At step t, the agent observes a multimodal observation o_t and predicts an action from the interaction history $h_t = (o_1, a_1, \ldots, o_t)$. A rollout defines as

$$\tau = (x, o_1, a_1, \ldots, o_T, a_T), \tag{1}$$

and the verifier $V_x(\tau) \in \{0, 1\}$ determines whether the final environment state satisfies the task goal. The plain policy model is

$$\pi_{\text{policy}}(a_t \mid h_t, x) \triangleq \pi_\theta(a_t \mid h_t, x), \tag{2}$$

which is not conditioned on any skill. Given a task-specific skill s_x, the skill-guided policy model can be expressed as

$$\pi_{\text{skill}}(a_t \mid h_t, x, s_x) \triangleq \pi_\theta(a_t \mid h_t, x, s_x). \tag{3}$$

Both policies are parameterized by the shared parameters θ, operate over the same action space, and differ only in the conditioning context. The skill acts as a training-time privileged recovery context that steers the current model toward a more informed recovery mode without changing its underlying interface.

Given a trajectory dataset $\mathcal{D}$, standard supervised fine-tuning optimizes the plain policy with the loss

$$\mathcal{L}_{\text{SFT}}(\mathcal{D}; \theta) = -\mathbb{E}_{\tau \sim \mathcal{D}} \sum_{t=1}^{|\tau|} \log \pi_\theta(y_t \mid h_t, x). \tag{4}$$

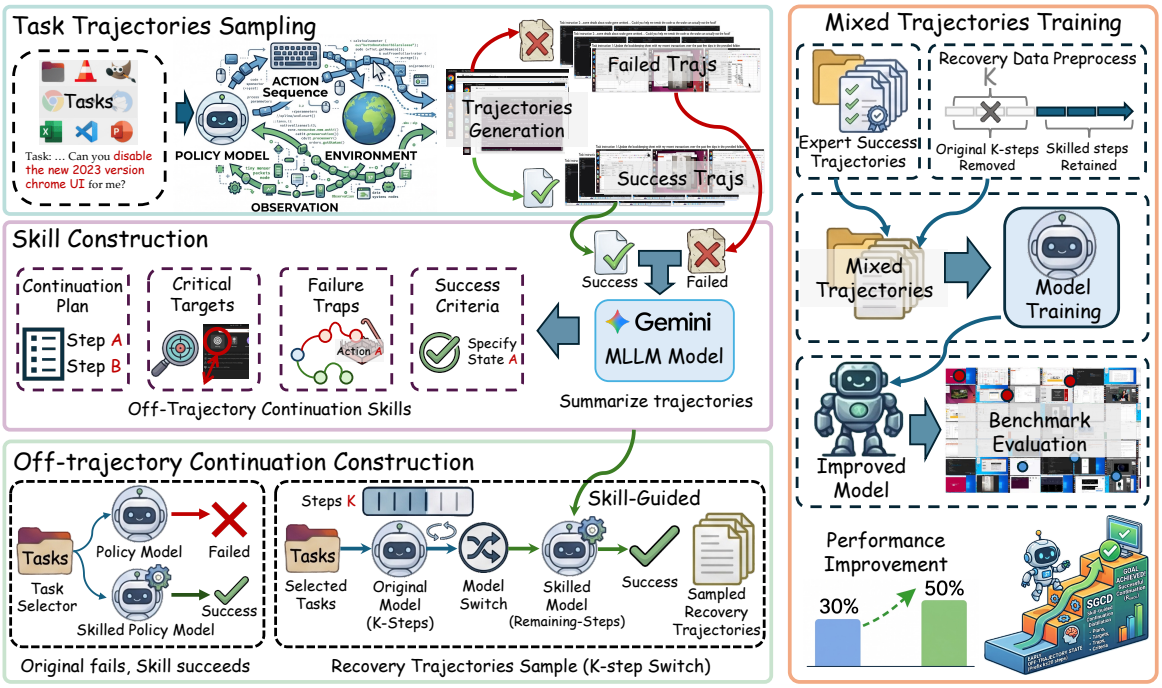

Figure 2: Overview of SGCD. (1) **Task Trajectories Sampling**: collect successful and failed plain-policy trajectories. (2) **Skill Construction**: extract off-trajectory continuation skills from trajectory evidence. (3) **Off-trajectory Continuation Construction**: use k-step handoff to collect skill-guided successful continuations. (4) **Mixed Trajectories Training**: train the plain policy with expert and verified continuation trajectories.

Here, y_t denotes the ground-truth action of the trajectory τ. This objective trains the policy to imitate the action labels in $\mathcal{D}$ and serves as the base training objective used throughout SGCD.

4 Method

4.1 Motivation

Expert trajectories provide no effective supervision for policy-induced off-trajectory states (Lauffer et al., 2025; Ross et al., 2011). Such states are not arbitrary perturbations: induced through the agent's closed-loop execution process, these states reflect the systematic biases of the learned policy. To inform the design of continuation supervision, we analyze failed rollouts from the plain policy (Fig. 1) and identify two empirical properties: failures are strongly concentrated in early execution, and failed trajectories exhibit recurring error tendencies rather than isolated accidental mistakes. These observations motivate **Skill-Guided Continuation Distillation** (SGCD), which first runs the plain policy to reach realistic off-trajectory states and then invokes the skill-guided policy to obtain successful continuations from these states. As illustrated in Fig. 2, SGCD proceeds in four stages. In Stage I (Sec. 4.2), we roll out the plain policy on different training tasks to collect successful and failed trajectories. In Stage II (Sec. 4.3), we summarize these trajectories and construct task-specific off-trajectory continuation skills. In Stage III (Sec. 4.4), the plain policy executes the first k GUI actions on recoverable failure tasks to reach realistic off-trajectory states, after which the current GUI state is passed to the skill-conditioned policy to obtain verified successful continuations. In Stage IV (Sec. 4.5), we process the resulting continuations and incorporate these continuations with expert trajectories to optimize the deployment policy without skill prompts.

4.2 Stage I: Task Trajectories Sampling

The first stage samples the unassisted behavior of the current plain policy. For each training task $x \in \mathcal{X}$, we reset the environment and run the plain policy M times, where M denotes the number of rollouts sampled per task. The resulting set is

$$\mathcal{T}_{x,\text{policy}} = \{\tau^m_{x,\text{policy}} \sim \pi_{\text{policy}}(\cdot \mid x)\}_{m=1}^{M}, \quad (5)$$

where $\tau^m_{x,\text{policy}}$ denotes the m-th complete trajectory collected by executing π_{policy} on task x. Using the task verifier V_x, we partition all sampled rollouts of task x into successful and failed trajectories:

$$\mathcal{D}^+_x = \{\tau \in \mathcal{T}_{x,\text{policy}} : V_x(\tau) = 1\}, \quad (6)$$

$$\mathcal{D}_x^- = \{\tau \in \mathcal{T}_{x,\text{policy}} : V_x(\tau) = 0\}, \quad (7)$$

The first stage deliberately collects rollouts from the plain policy rather than an external expert, because the resulting rollouts directly expose the off-trajectory states and recurring mistakes characteristic of the deployed agent's closed-loop behavior. Successful rollouts provide feasible workflows and terminal evidence, while failed rollouts expose model-specific traps, target confusions, and premature termination patterns that are later summarized into off-trajectory continuation skills.

4.3 Stage II: Skill Construction

We use Gemini-3-Pro (Google DeepMind, 2026) to transform task-level trajectory evidence into compact natural-language skills. The skill format is motivated by the path-plural nature of GUI control. Unlike math and code tasks, GUI tasks rarely admit a unique ground-truth solution trace. The same instruction for GUI agents may be completed through different menus, shortcuts, dialog states, file views, or interaction orders. Directly conditioning on a successful path is therefore a brittle form of supervision. It over-binds the current decision to a particular historical state sequence and preserves incidental low-level details such as menu order or scroll position. As a result, the model may learn to imitate a route rather than understand how the task can be recovered from an uncertain intermediate state.

SGCD represents a skill as a *trajectory abstraction* rather than a trajectory replay. It summarizes trajectory-level evidence into off-trajectory continuation skills while filtering out incidental path details. We construct this abstraction from our failure analysis. As illustrated in Fig. 1(b), failed GUI rollouts can be organized into four recurring failure families: *Early Done*, where the model terminates before the task is actually complete; *Fixation*, where it repeats ineffective actions without switching strategy; *Hallucinated Affordance*, where it searches for unavailable menus, settings, or commands; and *Scope Misjudgment*, where it selects an incorrect UI function or dialog despite understanding the broad instruction.

The skill schema is designed to guide successful continuations from off-trajectory states. It summarizes what should be achieved, which UI targets matter, what failure traps should be avoided, and how completion should be verified, while leaving the skill-conditioned policy free to choose a feasible continuation from the current GUI state. This abstraction is especially suitable for continuation construction, since the handoff state is produced online by the policy model and may not match any historical successful trace. Accordingly, each skill s_x is indexed by task id and organized according to the schema in Tab. 1.

Skill Field	Skill Role
Continuation Plan	Addresses *Fixation* with high-level maneuvers and alternatives for switching away from stalled strategies.
Critical Targets	Addresses *Scope Misjudgment* by identifying task-relevant UI functions, files, widgets, cells, menus, or dialogs.
Failure Traps	Addresses *Hallucinated Affordance* by listing nonexistent affordances, misleading operations, and common dead ends.
Success Criteria	Addresses *Early Done* with observable completion checks specifying when termination is justified.

Table 1: The structured schema of a task-specific off-trajectory continuation skill.

4.4 Stage III: Off-Trajectory Continuation Construction

The third stage constructs successful continuations from recoverable failed tasks. A recoverable task is selected only if the plain policy fails while the skill-guided policy succeeds:

$$\mathcal{X}_{\text{rec}} = \{x \in \mathcal{X}_{\text{fail}} : V_x(\tau_{x,\text{skill}}) = 1\}, \quad (8)$$

where $\mathcal{X}_{\text{fail}}$ represents failed tasks of policy model π_{policy}. This filtering avoids spending sampling budget on tasks for which the skill-guided policy π_{skill} cannot yet find a successful solution, since such tasks are unlikely to yield verified successful continuations. Because SGCD is iterative, this recoverable task set is not fixed. As the plain policy improves in later rounds, additional tasks may become recoverable and enter subsequent off-trajectory continuation data construction.

As shown in Fig. 1(a), failures are strongly concentrated in early execution. We therefore enumerate switch steps within this early window, i.e., $k \in \{1, \ldots, 20\}$. By sweeping over multiple handoff depths, SGCD reduces hand-crafted bias in selecting off-trajectory states while preserving enough remaining horizon for the skill-conditioned policy to complete the task.

For each selected task $x \in \mathcal{X}_{\text{rec}}$ and switch step k, we reset the environment and let the plain policy

execute the first k GUI actions. This produces a live policy-induced state with history

$$h^p_{k+1} = (o_1, a_1, \ldots, o_k, a_k, o_{k+1}) \sim \pi_{\text{policy}}(\cdot \mid x). \tag{9}$$

This action segment is not replayed from a stored trajectory. It is produced in the current environment, so the handoff state reflects a state that the plain policy actually reaches. Starting from this state, the skill-guided policy model continues execution with the task-specific skill s_x:

$$\hat{\tau}_{>k} = (\hat{a}_{k+1}, \ldots, \hat{o}_T, \hat{a}_T) \sim \pi_{\text{skill}}(\cdot \mid h^p_{k+1}, x, s_x). \tag{10}$$

The resulting successful continuation trajectory is

$$\hat{\tau} = (x, o_1, a_1, \ldots, o_{k+1}, \hat{a}_{k+1}, \ldots, \hat{o}_T, \hat{a}_T). \tag{11}$$

Each spliced rollout is filtered by two complementary signals. First, the executable verifier must accept the final state, $V_x(\hat{\tau}) = 1$. Second, an LLM judge (Zheng et al., 2023; Lù et al., 2025) inspects the task, observations, actions, and final outcome to remove accidental success, redundant loops, inconsistent reasoning, unsafe operations, and behavior unrelated to the instruction. Only post-handoff continuations that pass both filters are added to the continuation dataset $\mathcal{D}_{\text{cont}}$.

4.5 Stage IV: Mixed Trajectories Training

The final stage trains the deployment policy on a mixture of original expert trajectories $\mathcal{D}_{\text{exp}}$, verified successful policy trajectories $\mathcal{D}^+$, and verified successful continuations $\mathcal{D}_{\text{cont}}$. For each retained continuation trajectory $\hat{\tau}$, we discard the pre-handoff policy segment $(o_1, a^p_1, \ldots, o_k, a^p_k)$ from action supervision. This segment is used only as historical context and does not provide supervised action targets, since it may contain the policy behaviors that induced the off-trajectory state.

The continuation loss is

$$\mathcal{L}_{\text{cont}}(\theta) = -\mathbb{E}_{\hat{\tau} \sim \mathcal{D}_{\text{cont}}} \sum_{t=k+1}^{|\hat{\tau}|} \log \pi_\theta(\hat{a}_t \mid \hat{h}_t, x), \tag{12}$$

where each trajectory $\hat{\tau}$ comes with its source task x and handoff step k. Here, $\hat{h}_t$ denotes the history used to predict the continuation action $\hat{a}_t$.

We combine standard supervised training on original expert trajectories and verified successful policy trajectories with continuation supervision:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{SFT}}(\mathcal{D}_{\text{exp}} \cup \mathcal{D}^+; \theta) + \mathcal{L}_{\text{cont}}(\mathcal{D}_{\text{cont}}; \theta). \tag{13}$$

The objective remains a standard action-generation objective, but the mixed training data shifts supervision toward successful continuations from policy-induced off-trajectory states. After training, the deployment policy receives no skill. The skill context is used only as a data-synthesis scaffold.

SGCD achieves continual self-improvement through iterative application. After each round, the improved policy induces a new failure distribution, and previously unrecoverable tasks may become recoverable in subsequent rounds. Each iteration reruns trajectory sampling, updates skills, and regenerates verified successful continuations for training, progressively expanding continuation supervision as the policy's capability grows.

5 Experiments

5.1 Experimental Setup

Backbones. We apply SGCD to three vision-language GUI policy models with different scales and model families: Qwen3-VL-8B, Qwen3-VL-30B-A3B (Bai et al., 2025), and STEP3-VL-10B (Huang et al., 2026). This setting evaluates the effectiveness and generality of SGCD.

Training settings. All experiments are trained on 64 H100 GPUs. Following (Sun et al., 2025; Wu et al., 2025), we synthesize tasks over multiple real-world OS applications and construct the initial expert dataset by sampling trajectories with several advanced models, followed by filtering to retain high-quality successful executions. Detailed training settings are provided in the appendix.

Baselines and metrics. We compare against commercial and GUI-specialized models, with the full list provided in Tab. 2. Following (Li et al., 2025; Fu et al., 2025; Agashe et al., 2025; Gonzalez-Pumariega et al., 2025), we evaluate on the OSWorld-Verified (Xie et al., 2024), which reflects the real user experience. We use end-to-end task success rate on OSWorld-Verified. To measure off-trajectory continuation ability, we define *Continuation Success Rate*. We select tasks that the plain policy model fails. For each task, we first let the plain policy execute k steps to induce an off-trajectory state, and then hand control to the evaluated model to complete the task from that state. Importantly, the evaluated model is not given any skill or guidance, so the metric measures whether the model has ability itself. Continuation Success Rate is the success rate of these tasks.

Method	#Params	OSWorld-Verified					
		Full↑	OS	Office	Daily	Prof.	Workflow
General-Purpose Models							
Seed1.8(Seed, 2026)	–	61.9	16	81	29	58	39
Kimi K2.5(Team et al., 2026)	1T	63.3	17	81	29	59	43
Claude Sonnet 4.6 (Anthropic, 2026a)	–	72.1	22	88	33	58	56
GPT-5.5 (OpenAI, 2026)	–	78.7	–	–	–	–	–
GUI-Specialized Models							
OpenCUA-7B (Wang et al., 2026b)	7B	28.7	11	28	18	34	13
UI-TARS-1.5-7B (Qin et al., 2025)	7B	29.6	8	35	19	37	8
TianXi-Action-7B (Tang et al., 2025)	7B	30.4	8	34	19	43	5
GUI-Owl-7B (Ye et al., 2025)	7B	32.1	12	29	19	47	9
OpenCUA-32B (Wang et al., 2026b)	32B	35.6	15	36	20	45	12
DeepMiner-Mano-7B (Fu et al., 2025)	7B	40.1	12	46	18	53	16
DART-GUI-7B(Li et al., 2025)	7B	40.5	13	39	26	52	17
OpenCUA-72B-preview (Wang et al., 2026b)	72B	45.9	14	55	26	52	19
UI-TARS-2 (Wang et al., 2025)	–	53.1	10	72	29	49	32
DeepMiner-Mano-72B (Fu et al., 2025)	72B	53.9	16	74	25	57	23
Backbone-Matched Models							
Qwen3-VL-30B-A3B backbone							
Lite-GUI-30B-A3B (Wu et al., 2026)	30B-A3B	22.7	–	–	–	–	–
Qwen3-VL-30B-A3B-Instruct† (Bai et al., 2025)	30B-A3B	31.9	8	43	15	37	11
Qwen3-VL-30B-A3B-Thinking† (Bai et al., 2025)	30B-A3B	31.3	11	40	13	41	8
Holo2-30B-A3B (Company, 2025)	30B-A3B	37.4	–	–	–	–	–
SGCD-30B-A3B	**30B-A3B**	**58.4**	**18**	**78**	**17**	**55**	**43**
Qwen3-VL-8B backbone							
Qwen3-VL-8B-Instruct† (Bai et al., 2025)	8B	32.7	10	39	15	40	14
Qwen3-VL-8B-Thinking† (Bai et al., 2025)	8B	36.0	11	43	21	42	13
Step-GUI-8B(Yan et al., 2025)	8B	40.2	9	45	18	55	18
EvoCUA-8B(Xue et al., 2026)	8B	46.1	18	44	25	56	24
GUI-Owl-1.5-8B-Instruct† (Xu et al., 2026)	8B	52.4	18	59	19	65	28
SGCD-8B	**8B**	**55.1**	**16**	**75**	**18**	**58**	**32**
STEP3-VL-10B backbone							
STEP3-VL-10B† (Huang et al., 2026)	10B	24.1	12	17	13	39	6
SGCD-10B	**10B**	**53.2**	**13**	**69**	**19**	**59**	**32**

Table 2: Main results on OSWorld-Verified. We report overall OSWorld-Verified performance and category-level scores on different tasks. SGCD models are highlighted in **blue** and base models are in yellow. † denotes results reproduced under our evaluation protocol. Officially reported full scores are 52.3 for GUI-Owl-1.5-8B-Instruct, 30.3 for Qwen3-VL-30B-A3B-Instruct, 30.6 for Qwen3-VL-30B-A3B-Thinking, and 33.9 for Qwen3-VL-8B variants.

5.2 Main Results on OSWorld-Verified

Tab. 2 reports the main comparison on OSWorld-Verified. We include overall performance and category-level scores on OS, Office, Daily, Professional, and Workflow tasks. The *GUI-Specialized Models* block contains models specifically designed or trained for GUI tasks, while the *Backbone-Matched Models* block compares methods that share the same base backbones as our models. SGCD achieves state-of-the-art results among models with comparable parameter scales. In the backbone-matched comparison, SGCD substantially improves the original base models, with gains of over 20% on the Qwen3-VL-8B backbone and over 25% on the Qwen3-VL-30B-A3B backbone. These consistent improvements across different model families and scales demonstrate the effectiveness and generality of SGCD. Overall, the results show that SGCD provides a strong and complementary training signal for GUI agents, beyond simply scaling the base model or relying on successful trajectory imitation.

5.3 Off-Trajectory Continuation Ability

We further evaluate whether SGCD trains the model to complete tasks from policy-induced off-trajectory states. For each task that the original policy fails, we first run the original policy for k steps to induce an intermediate GUI state, and then let the evaluated model continue from that state without any skill prompt. This setting tests whether successful continuation ability is distilled into the

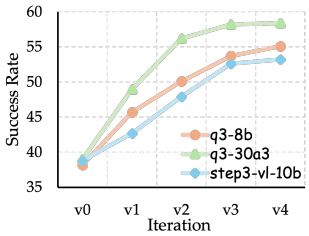

Figure 3: Iterative SGCD across training rounds.

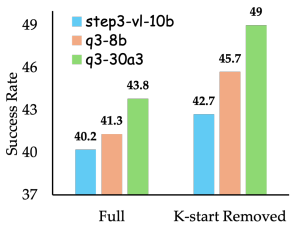

Figure 4: Full trajectory vs. K-start supervision.

Skill Block	Full	OS	Office	Daily	Prof.	Work.
Without Skill	39.9	10	51	19	45	19
CP	41.3	12	48	15	59	15
CP + CT	42.7	12	50	16	52	24
CP + CT + FT	43.2	12	46	16	57	25
CP + CT + FT + SC	45.7	12	52	16	61	24

Table 3: Ablation on skill components. CP, CT, FT, and SC denote Continuation Plans, Critical Targets, Failure Traps, and Success Criteria.

Method	#Params	Cont. SR↑
General-Purpose Models		
Kimi K2.5	1T	56.3
Claude Sonnet 4.6	–	63.3
Backbone-Matched Models		
Qwen3-VL-8B backbone		
Qwen3-VL-8B-Instruct	8B	18.9
Qwen3-VL-8B-Thinking	8B	27.2
GUI-Owl-1.5-8B-Instruct	8B	28.2
EvoCUA-8B	8B	32.7
SGCD-8B	**8B**	**39.2**
Qwen3-VL-30B-A3B backbone		
Qwen3-VL-30B-A3B-Instruct	30B-A3B	18.8
Qwen3-VL-30B-A3B-Thinking	30B-A3B	23.3
SGCD-30B-A3B	**30B-A3B**	**50.3**

Table 4: Continuation Success Rate (Cont. SR) comparison on the 142 OSWorld-Verified tasks failed by the original plain policy.

plain policy itself.

As shown in Tab. 4, SGCD attains the highest continuation success rate among open-weight models at both backbone scales, reaching 39.2% on the Qwen3-VL-8B backbone and 50.3% on the Qwen3-VL-30B-A3B backbone, which approaches Kimi K2.5. This indicates that the skill-guided continuations are not merely useful during data synthesis, but are effectively distilled into the plain policy. The resulting model already learns how to complete tasks from off-trajectory states.

5.4 Iterative Distillation

SGCD is iterated because each trained policy induces a new distribution of off-trajectory states. After one round, some previous failures become solvable, and new skill-solvable states emerge near the expanded capability boundary. We therefore repeat the pipeline of SGCD.

Fig. 3 reports performance across SGCD training rounds. Performance improves consistently from v0 to later iterations, showing that repeated SGCD can further enhance the plain policy.

5.5 Ablation Studies

We conduct all ablations on the v0-to-v1 stage of SGCD, where the largest improvement is observed.

Skill components. Tab. 3 studies the contribution of the off-trajectory continuation skill. Starting from no skill, we progressively add Continuation Plans, Critical Targets, Failure Traps, and Success Criteria. These components correspond to the dominant failure modes identified in our trajectory analysis: fixation, scope misjudgment, hallucinated affordance, and early done. The ablation tests whether the full skill works as a trajectory abstraction rather than a fixed path replay.

K-start supervision. SGCD uses only the post-handoff continuation for training. The pre-handoff segment is kept as context but excluded from action supervision. This design follows our motivation that off-trajectory states are more likely to produce erroneous actions. Therefore, the early plain-policy actions that lead to these states may contain noisy decisions rather than useful supervision. Fig. 4 compares this design with training on the full trajectories. The results confirm that excluding the pre-handoff actions leads to better performance.

6 Conclusion

We identify the off-trajectory supervision deficit as a fundamental limitation of behavior cloning on expert trajectories, where policy-induced states encountered during closed-loop execution receive no effective supervision signal. To address this, we propose **Skill-Guided Continuation Distillation (SGCD)**, which leverages skill-guided rollouts to synthesize verified successful continuations from such states, supplying the supervision that expert trajectories cannot provide. On OSWorld-Verified, SGCD improves the success rate of three base models from the low-30% range to over 50%, demonstrating its effectiveness and generality.

Limitations

This study has several limitations. First, the current framework has limited effectiveness in acquiring successful continuations for difficult tasks. Improving the coverage and efficiency of continuation acquisition on such tasks remains an open direction for future work. Second, SGCD currently instantiates off-trajectory states by re-executing the plain policy from scratch in the live environment for each task, which incurs substantial interaction overhead. Developing state-caching infrastructure to store and reuse intermediate GUI states directly could substantially reduce the number of environment steps required per iteration.

Acknowledgments

This work was supported by the Fundamental Research Funds for the Central Universities (Grant number: FRF-TP-26-114) to Xiaowen Zhang.

References

Saaket Agashe, Jiuzhou Han, Shuyu Gan, Jiachen Yang, Ang Li, and Xin Wang. 2025. Agent s: An open agentic framework that uses computers like a human. In *International Conference on Learning Representations*, volume 2025, pages 22924–22946.

Anthropic. 2026a. Claude sonnet 4.6 system card. https://www-cdn.anthropic.com/78073f739564e986ff3e28522761a7a0b4484f84.pdf. Anthropic Technical Report.

Anthropic. 2026b. Model system cards. https://www.anthropic.com/system-cards.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Yihan Chen, Benfeng Xu, Xiaorui Wang, Yongdong Zhang, and Zhendong Mao. 2025. Training llm-based agents with synthetic self-reflected trajectories and partial masking. *arXiv preprint arXiv:2505.20023*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332.

H Company. 2025. Holo2 - open foundation models for navigation and computer use agents.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Tianyu Fu, Anyang Su, Chenxu Zhao, Hanning Wang, Minghui Wu, Zhe Yu, Fei Hu, Mingjia Shi, Wei Dong, Jiayao Wang, et al. 2025. Mano technical report. *arXiv preprint arXiv:2509.17336*.

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

Google DeepMind. 2026. Gemini 3 pro model card. Technical report, Google DeepMind.

Boyu Gou, Demi Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations*, volume 2025, pages 30851–30883.

Ruiling Guo, Xinwei Yang, Chen Huang, Tong Zhang, and Yong Hu. 2025. Candy: Benchmarking llms' limitations and assistive potential in chinese misinformation fact-checking. In *EMNLP (Findings)*, pages 5724–5758.

H Company. 2025. Holo2: Cost-efficient models for cross-platform computer-use. https://hcompany.ai/holo2. Accessed: 2026-05-20.

Ailin Huang, Chengyuan Yao, Chunrui Han, Fanqi Wan, Hangyu Guo, Haoran Lv, Hongyu Zhou, Jia Wang, Jian Zhou, Jianjian Sun, et al. 2026. Step3-vl-10b technical report. *arXiv preprint arXiv:2601.09668*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Hanyu Lai, Xiao Liu, Yanxiao Zhao, Han Xu, Hanchen Zhang, Bohao Jing, Yanyu Ren, Shuntian Yao, Yuxiao Dong, and Jie Tang. 2025. Computerrl: Scaling end-to-end online reinforcement learning for computer use agents. *arXiv preprint arXiv:2508.14040*.

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

Pengxiang Li, Zechen Hu, Zirui Shang, Jingrong Wu, Yang Liu, Hui Liu, Zhi Gao, Chenrui Shi, Bofei Zhang, Zihao Zhang, Xiaochuan Shi, Zedong Yu, Yuwei Wu, Xinxiao Wu, Yunde Jia, Liuyu Xiang, Zhaofeng He, and Qing Li. 2025. Efficient multi-turn rl for gui agents via decoupled training and adaptive data curation. *arXiv preprint arXiv:2509.23866*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, et al. 2026. Ui-voyager: A self-evolving gui agent learning via failed experience. *arXiv preprint arXiv:2603.24533*.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594.

OpenAI. 2026. Gpt-5.5 system card. https://openai.com/index/gpt-5-5-system-card/. OpenAI Deployment Safety Hub.

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

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. 2025. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pages 406–441.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Hongjin Su, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan Arik. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *International Conference on Learning Representations*, volume 2025, pages 89044–89097.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. 2025. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579.

Liang Tang, Shuxian Li, Yuhao Cheng, Yukang Huo, Zhepeng Wang, Yiqiang Yan, Kaer Huang, Yanzhe Jing, and Tiaonan Duan. 2025. Sea: Self-evolution agent with step-wise reward for computer use. *arXiv e-prints*, pages arXiv–2508.

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

Guoqing Wang, Sunhao Dai, Guangze Ye, Zeyu Gan, Wei Yao, Yong Deng, Xiaofeng Wu, and Zhenzhe Ying. 2026a. Information gain-based policy optimization: A simple and effective approach for multi-turn search agents. In *The Fourteenth International Conference on Learning Representations*.

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

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. 2026b. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806.

Zezhou Wang, Ziyun Zhang, Xiaoyi Zhang, Zhuzhong Qian, and Yan Lu. 2026c. From off-policy to on-policy: Enhancing gui agents via bi-level expert-to-policy assimilation. *arXiv preprint arXiv:2601.05787*.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming Yan, Fei Huang, Xiaoshan Yang, et al. 2026. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in gui automation. *Advances in Neural Information Processing Systems*, 38:3907–3929.

Yubin Wu, Zicheng Cai, Liping Ning, Hua Wang, Zhi Chen, Yaohua Tang, and Hao Chen. 2026. Litegui: Distilling compact gui agents with reinforcement learning. *arXiv preprint arXiv:2605.07505*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen

Ding, Liheng Chen, Paul Pu Liang, et al. 2025. Osatlas: Foundation action model for generalist gui agents. In *International Conference on Learning Representations*, volume 2025, pages 5090–5108.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. 2026. Scaling computer-use grounding via user interface decomposition and synthesis. *Advances in Neural Information Processing Systems*, 38.

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

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

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. 2025. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *International Conference on Learning Representations*, volume 2025, pages 79822–79843.

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

Haolong Yan, Jia Wang, Xin Huang, Yeqing Shen, Ziyang Meng, Zhimin Fan, Kaijun Tan, Jin Gao, Lieyu Shi, Mi Yang, et al. 2025. Step-gui technical report. *arXiv preprint arXiv:2512.15431*.

Xinwei Yang, Junyi Fan, Yuqing Liu, Jiaxuan Wang, Jiashuai Zhang, Hongru Liang, Wenqiang Lei, and Yao Song. 2026. Empathy in diversity: Personalized depression and anxiety therapy via dialogue state tracking and patient-aware planning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9370–9416.

Xinwei Yang, Zhaofeng Liu, Chen Huang, Jiashuai Zhang, Tong Zhang, Yifan Zhang, and Wenqiang Lei. 2025. Elaboration: A comprehensive benchmark on human-llm competitive programming. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 59–104.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, et al. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

Hongwei Yu, Jiansheng Chen, Xinlong Ding, Yudong Zhang, Ting Tang, and Huimin Ma. 2024. Step vulnerability guided mean fluctuation adversarial attack against conditional diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 6791–6799.

Hongwei Yu, Xinlong Ding, Jiawei Li, Jinlong Wang, Yudong Zhang, Rongquan Wang, Huimin Ma, and Jiansheng Chen. 2025. Dadet: Safeguarding image conditional diffusion models against adversarial and backdoor attacks via diffusion anomaly detection. In *ICCV 2025 (Spotlight)*, pages 17411–17421.

Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. 2025. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. *arXiv preprint arXiv:2505.11821*.

Ziyun Zhang, Xinyi Liu, Xiaoyi Zhang, Jun Wang, Gang Chen, and Yan Lu. 2025. Ui-evol: Automatic knowledge evolving for computer use agents. *arXiv preprint arXiv:2505.21964*.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. 2025. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 29733–29735.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Additional Handoff-Depth Analysis

To understand how the choice of handoff depth k shapes the continuation dataset, we ablate the k set used to construct $\mathcal{D}_{\text{cont}}$ for Qwen3-VL-8B. All variants share the same skill, verifier, and training recipe described in Sec. 4.5; only the range of admissible handoff depths differs. Results are reported in Tab. 5.

k Set	Score
All Set (1–20)	45.7
1–5	42.9
5–10	43.2
10–15	41.0
15–20	41.8

Table 5: Ablation on the handoff-depth set used for off-trajectory continuation construction.

Sweeping k dominates any single bucket. Enumerating $k \in \{1, \ldots, 20\}$ reaches 45.7, outperforming every individual sub-range by 2.5–4.7 points. This indicates that no single handoff depth is sufficient: different tasks expose their policy-induced off-trajectory states at different points along the rollout, and a fixed k inevitably misses the off-trajectory state where the actionable mistake occurs. Sweeping the full window converts each failed task into multiple supervised continuations from distinct off-trajectory states, which both increases data volume and reduces the hand-crafted bias of any single handoff choice.

Performance degrades with deeper handoff. Within a fixed-width window, accuracy decreases monotonically as k grows beyond 10 (43.2 $\rightarrow$ 41.0 $\rightarrow$ 41.8). This matches the failure-distribution finding in Fig. 1(a): plain-policy errors are concentrated in early execution, so an off-trajectory state induced after only a few steps lies closest to the failure mode that distillation needs to repair. Deeper handoff also leaves the skill-guided policy with a shorter remaining horizon to complete the task, which lowers the verifier pass rate and yields fewer high-quality continuations per task.

B Per-Handoff-Depth Continuation Success Rate

Tab. 4 in the main paper reports the aggregate Continuation Success Rate. Here we provide the full per-handoff-depth breakdown for ten systems on the same evaluation protocol. The pool consists of 142 OSWorld-Verified tasks on which the plain Qwen3-VL-8B policy fails. For each task and each $k \in \{1, \ldots, 20\}$, we let the plain policy execute the first k steps to induce a policy-induced off-trajectory state and then hand control to the evaluated model, which must complete the task without any skill prompt. A trial is counted as successful when the verifier score is ≥ 0.8. The full 20×10 table is reported in Tab. 6.

SGCD beats backbone-matched baselines at every k. SGCD-30B-A3B averages 50.3% vs. 28.2% (GUI-Owl-1.5-8B-Instruct) and 27.2% (Qwen3-VL-8B-Thinking), and the gap is consistent across the entire sweep, not driven by a few favorable handoff depths. SGCD-8B reaches 39.2% vs. EvoCUA-8B's 32.7% under the same backbone budget, with the gap widening on early-to-mid k where the policy-induced off-trajectory state is most non-trivial.

Strong commercial models degrade fastest with k. Claude Sonnet 4.6 starts the highest at $k = 1$ (80.3%) but drops the most by $k = 20$ (47.2%), a 33-point decline. Kimi K2.5 falls 14 points over the same range. We attribute this to compounding prefix errors: the longer the plain Qwen3-VL-8B policy is allowed to act, the more its prefix accumulates mistakes that even strong general-purpose models cannot fully undo. SGCD-30B-A3B's range is much tighter (42–60%), consistent with its training objective: continuation distillation explicitly supervises behavior from policy-induced off-trajectory states, so robustness across handoff depths is the property it is trained to acquire.

Weak baselines flatten or even improve with k. Models with very low $k = 1$ Continuation Success Rate (Qwen3-VL-8B-Thinking at 20.4%; GUI-Owl-1.5-8B-Instruct at 27.5%) often perform better at moderate-to-deep k than at $k = 1$. This is because such models lack their own coherent open-ended plan; a moderately deep prefix from the plain policy effectively narrows the task and steers them toward the relevant UI surface, partially compensating for their weaker planning ability. SGCD-trained policies do not need this scaffolding – they remain strong at small k, where the off-trajectory state is closest to the actual policy failure mode.

C Extended Related Work

Vision-language GUI agents show strong promise in performing open-ended computer tasks (An-

	General-Purpose		SGCD (ours)		Open-Weight		Qwen3-VL Backbones			
k	Claude S4.6	Kimi K2.5	SGCD-30A3B	SGCD-8B	EvoCUA-8B	GUI-Owl-1.5-8B	8B-Inst	8B-Thk	30A3-Inst	30A3-Thk
1	80.28	61.97	59.15	42.25	28.87	27.46	12.68	20.42	13.38	9.15
2	77.46	60.56	57.75	38.73	26.76	23.94	12.68	19.72	7.04	14.79
3	78.17	66.20	57.04	36.62	32.39	26.76	11.97	25.35	11.97	15.49
4	71.83	59.86	59.15	41.55	33.80	26.76	16.90	23.24	12.68	14.08
5	71.83	58.45	59.86	37.32	30.28	26.06	14.08	23.94	13.38	17.61
6	73.94	56.34	49.30	37.32	27.46	19.72	10.56	22.54	16.90	19.72
7	71.83	56.34	48.59	36.62	37.32	24.65	16.90	26.76	12.68	22.54
8	66.20	54.23	49.30	40.85	30.99	30.28	16.20	29.58	15.49	23.24
9	76.76	59.86	52.82	47.89	33.10	23.24	17.61	25.35	15.49	23.94
10	64.08	59.86	51.41	42.25	28.87	29.58	16.90	28.17	18.31	21.83
11	66.90	61.27	44.37	38.73	26.76	27.46	21.83	32.39	19.01	23.94
12	64.08	57.04	46.48	33.80	33.80	25.35	25.35	29.58	19.01	32.39
13	54.93	57.75	45.07	38.03	35.21	30.99	19.01	28.87	21.83	26.76
14	46.48	51.41	46.48	38.03	34.51	30.28	21.83	28.17	24.65	26.06
15	47.18	51.41	47.18	35.21	41.55	30.99	23.94	30.99	25.35	30.28
16	45.77	48.59	46.48	37.32	33.10	30.28	19.01	26.76	26.06	31.69
17	55.63	51.41	47.18	40.14	34.51	34.51	21.83	33.80	23.24	28.17
18	56.34	52.82	49.30	41.55	35.92	31.69	25.35	29.58	25.35	28.17
19	50.00	52.82	47.18	42.25	33.80	33.80	27.46	28.17	26.06	26.76
20	47.18	47.89	42.25	37.32	35.21	30.28	26.06	30.28	28.87	28.87
Avg.	**63.34**	**56.30**	**50.32**	**39.19**	**32.71**	**28.20**	**18.91**	**27.18**	**18.84**	**23.27**

Table 6: Per-handoff-depth Continuation Success Rate (%) on the 142 OSWorld-Verified tasks failed by the original plain policy. For each k, the plain policy executes k steps before handing off to the evaluated model, which must complete the task without any skill prompt. Average row reports the unweighted mean over $k = 1, \ldots, 20$. 8B-Inst/Thk and 30A3-Inst/Thk denote the Qwen3-VL-8B-Instruct/Thinking and Qwen3-VL-30B-A3B-Instruct/Thinking backbones, respectively.

thropic, 2026b; OpenAI, 2026; Google DeepMind, 2026; Hurst et al., 2024; Yang et al., 2026). Standardized benchmarks across web, mobile, and desktop environments provide rigorous evaluation platforms for this line of research (Deng et al., 2023; Zhou et al., 2024; Zheng et al., 2024; Rawles et al., 2025; Xie et al., 2024; Guo et al., 2025; Yang et al., 2025), and a growing body of GUI-specialized systems advances grounding, planning, action generation, and trajectory-level supervision (Qin et al., 2025; Wang et al., 2026b; Yan et al., 2025; Xu et al., 2026; H Company, 2025; Wu et al., 2026; Xue et al., 2026; Ye et al., 2025; Cheng et al., 2024; Gou et al., 2025; Lin et al., 2025). SGCD is complementary to these advances: rather than proposing a new base model or benchmark, it targets the off-trajectory supervision deficit left by behavior cloning on expert trajectories.

Most GUI agents learn from successful demonstrations, synthetic trajectories, or state-action traces (Deng et al., 2023; Zheng et al., 2024; Qin et al., 2025; Wang et al., 2026b; Yan et al., 2025; Xu et al., 2026; Xue et al., 2026). Such supervision teaches grounding, action formatting, and interface conventions, but primarily covers expert-induced states. In closed-loop environments (Zhou et al., 2024; Rawles et al., 2025; Xie et al., 2024), local mistakes produce policy-induced off-trajectory states that are absent from successful traces. This is a direct manifestation of covariate shift in imitation learning (Ross et al., 2011; Lauffer et al., 2025): the policy accumulates errors in states unseen during training, yet obtaining new supervision requires costly re-interaction with the live environment.

Self-improvement methods seek to address this by leveraging the agent's own experience as a training signal. Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023) use inference-time verbal feedback to revise outputs. Recent GUI pipelines convert model-generated rollouts into supervision through filtered self-training (Yan et al., 2025; Wu et al., 2026), sandbox-based reinforcement learning (Lai et al., 2025), experience-driven knowledge refinement (Zhang et al., 2025; Lin et al., 2026), or policy-aligned experience assimilation (Wang et al., 2026c). SGCD follows this paradigm and specifically targets policy-induced off-trajectory states, synthesizing skill-guided continuation supervision from the states the current policy actually traverses.

Finally, structured guidance, memory, and experience retrieval improve agent planning (Shinn et al., 2023; Madaan et al., 2023; Agashe et al., 2025; Qin et al., 2025; Wang et al., 2026b; Xue et al., 2026; Yu et al., 2024, 2025) Agent S uses experience-augmented hierarchical planning (Agashe et al., 2025), while OpenCUA, UI-TARS, and EvoCUA incorporate reflective

conversion, memory-like capabilities, or self-correction (Wang et al., 2026b; Qin et al., 2025; Xue et al., 2026). SGCD uses skills only as a training-time synthesis scaffold: Gemini-3-Pro abstracts successful and failed trajectories into Continuation Plans, Critical Targets, Failure Traps, and Success Criteria, enabling verified continuations without fixed-path replay or replacing the acting policy.

D Skill Extraction Prompt

We use Gemini-3-Pro (Google DeepMind, 2026) to abstract trajectory evidence into structured off-trajectory continuation skills. For each task, the prompt is fed with the task id, all available successful reference trajectories sampled by the plain policy, and a matched set of failed reference trajectories. The model is instructed to (i) infer the stable good path from successful references, (ii) infer obstacles, traps, and bad paths from failed references, (iii) compress trajectories into reusable procedural knowledge rather than verbatim replays, and (iv) emit a JSON object with exactly four fields aligned with the failure families analyzed in Sec. 4.3: *Continuation Plans*, *Critical Targets*, *Failure Traps*, and *Success Criteria*. To suppress hallucinated guidance, alternative paths are only drawn from successful references, and fields without supporting evidence are left empty rather than filled in. The full prompt template is shown in Fig. 5.

Skill Extraction Prompt (Gemini-3-Pro)

System Prompt

Task id: `{task_id}`.
You are extracting a concise SOP (standard operating procedure) for one computer-use task. You will be given successful references and failed references for the same task.

- Use **successful references** to infer the stable good path.
- Use **failed references** to infer obstacles, traps, and bad paths to avoid.
- Do not copy long trajectory text verbatim. Compress them into short, reusable procedural knowledge.
- Only include information supported by the provided references.

Output Format

Return JSON only with exactly these fields:

```
{
  "Continuation Plans": string[],
  "Critical Targets":   string[],
  "Failure Traps":      string[],
  "Success Criteria":   string[]
}
```

Guidelines

- Keep each bullet short, concrete, and reusable.
- Prefer task-level decisions over click-by-click replay.
- Include wrong formulas / wrong pages / wrong menus in `Failure Traps` when supported by failed references.
- Alternative paths must only come from successful references; do not invent or generalize new alternatives.
- If uncertain, leave a field empty instead of hallucinating.

User Prompt

Successful references: `{successful_trajectories}`
Failed references: `{failed_trajectories}`

Figure 5: Skill-extraction prompt used with Gemini-3-Pro. Task-specific successful and failed reference trajectories are concatenated into the user-prompt slot, and the model is required to emit JSON with the four schema fields.

E Skill Examples

To illustrate the structured output produced by the skill-extraction prompt, we show three representative skills covering different task families: a web browsing task on a Chinese auto-information portal (Fig. 6), an office-document editing task on a word processor (Fig. 7), and a browser-settings task (Fig. 8). Each skill contains four task-specific fields aligned with the failure families analyzed in Sec. 4.3: *Continuation Plans* provide the abstracted good path, *Critical Targets* list the salient UI anchors, *Failure Traps* enumerate concrete mistakes observed in failed references, and *Success Criteria* define the verifier-aligned terminal state.

Skill Example: Browse Wuling MINIEV owner community posts on Dongchedi

Continuation Plans

- Find and click the "Find Car" or "Select Car" entry in the top navigation bar of the Dongchedi homepage (dongchedi.com).
- Type "Wuling MINIEV" in the search box and press Enter to search.
- On the search results page, click the "Wuling MINIEV" model card to enter the model detail page.
- On the model detail page, find and click the "Owner Community" or "Community" tab.
- Wait for the community feed to finish loading and confirm the page URL contains `/community/4499`.

Critical Targets

- Dongchedi homepage search box.
- "Wuling MINIEV" search result card.
- "Owner Community" tab on the model detail page.

- Target URL: `dongchedi.com/community/4499`.

Failure Traps
- Do not browse the homepage recommended content without searching for the specific model.
- Do not click "Hongguang MINIEV" or other similarly named model suggestions; confirm it is the "MINIEV" itself.
- Do not stay on the model specs or pricing page; you must navigate to the "Owner Community" tab.
- Do not click ad pop-ups or promotional banners on the page, as they navigate to irrelevant pages.

Success Criteria
- The active tab URL is `dongchedi.com/community/4499`.
- The page displays the Wuling MINIEV owner community feed with a list of posts.

Figure 6: Skill extracted for a web browsing task on a Chinese auto-information portal. The Failure Traps field captures search-suggestion confusables (e.g., "Hongguang MINIEV" vs. "Wuling MINIEV") that repeatedly mislead the plain policy.

Skill Example: Replace all question marks with exclamation marks

Continuation Plans
- Press Ctrl+H to open the "Find and Replace" dialog.
- In the "Search For" input field, type "?" (question mark).
- Click the "Replace With" input field and type "!" (exclamation mark).
- Make sure the "Regular expressions" checkbox is NOT checked; otherwise "?" will be treated as a regex metacharacter.
- Click the "Replace All" button to perform the batch replacement.
- Confirm the replacement completion prompt (showing the number of replacements made).
- Press Escape to close the Find and Replace dialog.
- Press Ctrl+S to save the document.

Critical Targets
- Ctrl+H to open the Find and Replace dialog.
- "Search For" input field (type "?").
- "Replace With" input field (type "!").
- "Replace All" button.
- Ctrl+S to save.

Failure Traps
- Do not check the "Regular expressions" option, because "?" is a special character (quantifier) in regex and will cause the replacement to fail or match incorrectly.
- Do not check options under "Other options" that could affect matching, such as "Current selection only".
- Do not forget to check whether the "Search For" field has leftover text; clear it first before typing.
- Do not replace one by one (clicking "Replace" instead of "Replace All"); this is extremely inefficient for documents with many occurrences.

Success Criteria
- All original question marks in the document have been replaced with exclamation marks.
- All other content and formatting in the document remain unchanged.
- The document is saved.

Figure 7: Skill extracted for an office-document editing task. The Failure Traps field captures a non-obvious dialog-option pitfall (the "Regular expressions" checkbox interfering with the literal "?" match) that is hard to recover from once committed.

Skill Example: Enable blocking third-party cookies in Chrome

Continuation Plans
- Type `chrome://settings/cookies` in the Chrome address bar and press Enter to navigate directly to the Cookie settings page.
- Alternatively, click the three-dot menu in the top-right corner > Settings > Privacy and Security > Third-party cookies.
- On the Cookie settings page, find the "Block third-party cookies" option.
- Click the radio button or toggle for that option to set it to "Block third-party cookies" mode.
- Confirm the setting has taken effect (the option is selected or the toggle is in the on position).

Critical Targets
- Chrome address bar (type `chrome://settings/cookies`).
- Or three-dot menu > Settings > Privacy and Security.
- "Third-party cookies" or "Cookies and other site data" setting.
- "Block third-party cookies" radio button/toggle.

Failure Traps
- Do not select "Block all cookies"; that will break many websites. Only block third-party cookies.
- Do not look for the Cookie option under "Site settings"; it is under the main "Privacy and Security" category.
- Do not confuse "Clear cookies" with "Block cookies"; they are two different operations.
- Do not close the browser to "apply" the settings; the change takes effect immediately.

Success Criteria
- On Chrome's Cookie settings page, the "Block third-party cookies" option is selected/enabled.
- The setting persists after restarting Chrome.

Figure 8: Skill extracted for a browser-settings task. The Failure Traps field disambiguates closely related but semantically different options ("Block all cookies" vs. "Block third-party cookies", "Clear cookies" vs. "Block cookies") that frequently cause plain-policy off-trajectory drift.

F Training Task Construction

We construct an OSWorld-compatible training pool $\mathcal{X}$ that is disjoint from the OSWorld-Verified (Xie et al., 2024) evaluation suite. Each task $x \in \mathcal{X}$ is a triple of (i) a natural-language instruction, (ii) a scripted initial environment state on the Ubuntu

sandbox, and (iii) an automatic verifier returning $V_x(\tau) \in \{0, 1\}$ from the post-rollout environment state. Whereas prior GUI training-data pipelines focus on trajectory synthesis (Xu et al., 2025; Sun et al., 2025; Su et al., 2025; Xue et al., 2026) or grounding data (Xie et al., 2026), our pipeline jointly produces the instruction, the initial-state assets, and the verifier from a per-application scenario specification, covers all ten desktop applications exercised by OSWorld-Verified, and is sized to support the trajectory-mixing recipe of Sec. 4.5.

Domain taxonomy. The ten target applications are organised into five coarse categories that serve as the unit of scenario design:

- **OS** – `os`: file manager, system settings, terminal, package and process management, and other OS-level operations.
- **Office** – `libreoffice_calc`, `libreoffice_impress`, `libreoffice_writer`: spreadsheet formatting, formulas, charts, and pivot tables; slide layouts, animations, and master styles; document editing, find-and-replace, styles, tables, and page setup.
- **Daily** – `chrome`: web navigation, multi-tab search, form filling, bookmarks and history management, downloads, extensions, and browser-setting tasks.
- **Professional** – `gimp`, `thunderbird`, `vlc`, `vs_code`: image editing (layers, filters, color adjustments, selections, export); email composition, folder organisation, and mail-filter management; media playback control, playlists, subtitles, and conversion; code editing, search-and-replace, refactoring, extensions, debugging, and the integrated terminal.
- **Workflow** – `multi_apps`: cross-application tasks that compose two or more of the above, e.g., extracting data from a spreadsheet, rendering a chart in a presentation, and sending the result through email.

For each category we enumerate the modal usage scenarios and feature surfaces from official documentation and an LLM-driven web research pass over real-world usage patterns. The resulting per-application feature taxonomy is used purely as a sampling scaffold for instruction synthesis and is never fed to the trained policy.

Instruction synthesis and human filtering. Conditioned on the per-application feature taxonomy, an LLM proposes candidate natural-language instructions under three constraints: (i) *realistic* – the task should plausibly arise from real user workflows rather than being a synthetic UI exercise; (ii) *concrete and verifiable* – task completion must be decidable by inspecting the post-rollout environment; (iii) *deterministic verifier-relevant outcome* – successful completion must be decidable from a well-defined set of terminal-state properties, even when multiple interaction paths or irrelevant UI configurations are possible. A subsequent human pass de-duplicates near-identical instructions, removes ambiguous or under-specified phrasings, and rejects tasks whose terminal state cannot be operationalised without subjective judgement.

Initial-state assets and automatic verifiers. For every accepted instruction we manually prepare the assets required to instantiate the task on a fresh VM (e.g., the working spreadsheet, source document, reference image, mailbox snapshot, or browser profile), together with the reference file(s) used by the verifier. The initial state is realised by a per-task setup script that places these assets at the canonical paths expected by the OSWorld-Verified contract. The verifier itself follows one of two patterns according to the task family:

- **State-extraction verifiers.** For tasks whose goal is to navigate the system or an application into a specific configuration (e.g., reach a particular Chrome setting page, select a specific track in VLC, switch a VS Code panel), the verifier programmatically extracts the final application state through the application's introspection API or accessibility tree and asserts equality with the expected state.
- **File-match verifiers.** For tasks whose goal is to produce or modify a file (e.g., editing a Calc workbook, exporting a GIMP image, replacing strings in a Writer document), the verifier loads the post-rollout artifact and compares it against the reference file using property-level assertions (cell value/format, image content/metadata, document text/styles) rather than byte-level equality.

Difficulty binning. We label every task with the number of GUI primitives that an expert demonstrator needs to complete it, and bin tasks into three difficulty levels: *easy* (1–5 steps), *medium* (5–15 steps), and *hard* (15–100 steps). The three bins are sampled to be approximately uniformly distributed within each application, so that the training pool exercises both short skill-grounding tasks and long-

horizon planning.

Manual solvability check. Before a task enters the training pool, a human annotator executes it end-to-end on a fresh sandbox VM and confirms that the prepared initial state, the assets, and the automatic verifier are mutually consistent: the instruction is unambiguously actionable, the verifier returns $V_x(\tau) = 1$ on the human-completed trajectory, and no required asset or UI surface is missing. Tasks that fail this manual pass – because of an under-specified instruction, an unreachable goal state, or a verifier that disagrees with a correct human execution – are sent back for asset, instruction, or verifier revision and re-tested, or dropped if they cannot be repaired. This step closes the gap between syntactic verifier acceptance and end-to-end solvability, and guarantees that every released task in $\mathcal{X}$ has at least one verified human solution.

Final corpus. The resulting training pool contains 1,432 verified tasks spanning the ten domains; per-application counts are reported in Tab. 7. The corpus is intentionally weighted toward the WORKFLOW category, since multi-application tasks expose the long-horizon off-trajectory states that SGCD is designed to repair.

Category	Application	#Tasks
OS	`os`	88
Office	`libreoffice_calc`	189
	`libreoffice_impress`	179
	`libreoffice_writer`	90
Daily	`chrome`	188
Professional	`gimp`	86
	`thunderbird`	53
	`vlc`	58
	`vs_code`	90
Workflow	`multi_apps`	411
Total		**1,432**

Table 7: Per-application task counts in the constructed training pool $\mathcal{X}$. Tasks are organised into five coarse categories (OS, Office, Daily, Professional, Workflow) used as the unit of scenario and feature analysis during instruction synthesis.

G Algorithm Overview

Algorithm 1 summarizes the full Skill-Guided Continuation Distillation pipeline. Each iteration samples plain-policy rollouts, extracts a task-specific skill from successful and failed evidence, instantiates policy-induced off-trajectory states with a k-step plain-policy prefix, hands the state off to the skill-guided policy to obtain a verified successful continuation, and distills the post-handoff portion back into the plain policy without skill prompts. As the policy improves across iterations, more failed tasks become recoverable and enter subsequent continuation construction.

Algorithm 1 Skill-Guided Continuation Distillation

Input: tasks $\mathcal{X}$, expert trajectories $\mathcal{D}_{\text{exp}}$, model π_θ, skill constructor G, number of iterations R, handoff range $K_{\max}$
for iteration $r = 1$ to R **do**
  Initialize $\mathcal{D}^+ \leftarrow \emptyset$, $\mathcal{D}_{\text{cont}} \leftarrow \emptyset$, $\mathcal{X}_{\text{fail}} \leftarrow \emptyset$
  for task $x \in \mathcal{X}$ **do**
    Sample policy trajectories $\mathcal{T}_{x,\text{policy}}$ using π_{policy}
    Add verified successes to $\mathcal{D}^+$ and failed tasks to $\mathcal{X}_{\text{fail}}$
  end for
  for task $x \in \mathcal{X}_{\text{fail}}$ **do**
    Construct task skill $s_x \leftarrow G(\mathcal{T}_{x,\text{policy}})$ from success and failure evidence
    Sample $\tau_{x,\text{skill}} \sim \pi_{\text{skill}}(\cdot \mid x, s_x)$
    if $V_x(\tau_{x,\text{skill}}) = 0$ **then continue** ▷ not recoverable yet
    end if
    for $k = 1$ to $K_{\max}$ **do**
      Run π_{policy} online for k steps to reach h^p_{k+1}
      Hand off to $\pi_{\text{skill}}(\cdot \mid h^p_{k+1}, x, s_x)$ for the remaining horizon
      if spliced rollout $\hat{\tau}$ passes verifier V_x and LLM quality judge **then**
        Add only post-handoff examples $(\hat{h}_t, \hat{a}_t)$ for $t > k$ to $\mathcal{D}_{\text{cont}}$
      end if
    end for
  end for
  Train π_θ on $\mathcal{D}_{\text{exp}} \cup \mathcal{D}^+ \cup \mathcal{D}_{\text{cont}}$ without skill prompts
end for
Output: deployment policy π_θ (no skill at inference)

H Training Details

All three backbones are trained on 64 H100 GPUs (8 nodes $\times$ 8 GPUs) with the same trajectory mixture described in Sec. 4.5: original expert trajectories $\mathcal{D}_{\text{exp}}$, verified successful policy trajectories $\mathcal{D}^+$, and verified post-handoff continuations $\mathcal{D}_{\text{cont}}$. The Vision Transformer (ViT) and the vision–language aligner are frozen throughout, and only the language tower is fine-tuned. Training uses bfloat16 with the AdamW optimizer.

Qwen3-VL-8B and Qwen3-VL-30B-A3B. These two backbones share the same training recipe under the SWIFT (Megatron-SFT) framework (Zhao et al., 2025). We use tensor-parallel size 2 with sequence parallel enabled, micro batch size 2 and global batch size 512, and a maximum

sequence length of 10,240. Activation recomputation is set to full with the uniform method (one layer per recompute block) and ViT gradient checkpointing is enabled. Optimization uses learning rate 1×10^{-5} with a 1×10^{-6} minimum, a warmup fraction of 0.05, fused cross-entropy loss, and last-round loss masking so that supervision is applied only to the policy-predicted tokens of each turn. We train for 2 epochs over the mixed dataset with packing enabled to reduce padding overhead. The pre-handoff plain-policy actions in $\mathcal{D}_{\text{cont}}$ are kept as context but masked out of the action loss (K-start; Sec. 4.5).

STEP3-VL-10B. This backbone is trained with the STEPTRON framework using a long-context configuration tailored to its YARN-extended 128k position range. The model uses tensor-parallel size 8, pipeline-parallel size 1, sequence parallel, full activation recomputation, and freezes the ViT encoder. The training packs sequences to a global sequence length of $128 \times 1024 = 131{,}072$ tokens, with micro batch size 1 and global batch size 32 over 3,873 iterations (approximately 2 epochs of the trajectory mixture). Optimization uses AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$) with gradient clipping at 1.0, a cosine schedule with peak learning rate 1×10^{-4}, minimum learning rate 1×10^{-5}, and a 200-iteration linear warmup. Weight decay is set to 0, and bf16 is used without fp16 loss scaling. Key training hyperparameters for the three backbones are summarized in Tab. 8.

Setting	Qwen3-VL-8B / 30B-A3B	STEP3-VL-10B
Framework	Swift	Steptron
Hardware	64×H100	64×H100
TP / PP	2 / 1	8 / 1
Seq. parallel	✓	✓
Micro / global bsz.	2 / 512	1 / 32
Seq. length	10,240	131,072
Optimizer	AdamW	AdamW
LR (peak/min)	1e-5 / 1e-6	1e-4 / 1e-5
LR schedule	cosine, 5% warmup	cosine, 200 iter warmup
Train length	2 epochs	3,873 iter (≈2 ep.)
Precision	bf16	bf16
Freeze ViT / aligner	✓ / ✓	✓ / ✓

Table 8: Training hyperparameters. Qwen3-VL-8B and Qwen3-VL-30B-A3B share the same Swift recipe; STEP3-VL-10B uses the Steptron long-context configuration with its YARN-extended 128k position range.

I Evaluation Setup

Environment. We follow the OSWorld-Verified (Xie et al., 2024) interaction protocol and use the OSWorld-Verified (Xie et al., 2024) task suite for evaluation. The agent acts in an Ubuntu desktop sandbox with a fixed screen size of 1920×1080, predicting one action per step from raw screenshots. The action space is `pyautogui` for mouse and keyboard primitives, augmented with two control functions: `computer.wait` for synchronous pauses (e.g., during installation or long computation) and `computer.terminate` for declaring task completion or failure. Coordinates are predicted in normalized form (relative to image size) and projected back to absolute pixels before execution.

Inference parameters. For all backbones we use the same sampling configuration: `max_tokens` $= 4096$, `top_p` $= 0.95$, `temperature` $= 1.0$. Each task is run with at most 100 environment steps; the agent is forced to terminate with failure once the budget is exhausted. At the API level, we allow up to 10 outer retries for parse-level errors and up to 20 HTTP-level retries per outer call, with a 1,200-second request timeout.

Message construction. Each step the agent assembles an OpenAI-style chat-completion payload with the following structure:

- A `system` message containing the GUI-agent system prompt (Fig. 9). The prompt fixes the response format (*Thought*, *Action*, *Code*), the allowed action space, and a hard-coded sandbox password placeholder that some tasks require.
- Interleaved `user`/`assistant` turns for previous steps. To control context length, only the most recent $H = 3$ historical screenshots are included as `image_url` messages, each paired with the corresponding `assistant` response containing the past *Thought* and *Action*. Steps older than the image-history window are kept as text-only assistant context to preserve the planning chain without their screenshots.
- A final `user` turn containing the current screenshot (base64-encoded PNG) and the task instruction rendered through the `INSTRUCTION_TEMPLATE` ("Please generate the next move according to the screenshot, task instruction and previous steps").

For thinking-capable backbones, the assistant turns are rendered with explicit `[think] ... [/think]` segments so that the historical reasoning is

reusable; otherwise, the standard `## Thought` / `## Action` markdown layout is used. After each model call, the response is parsed for the `Action` block and the trailing fenced code block (`python` or `code`); a parse failure triggers a retry with reduced temperature.

Evaluation System Prompt

You are an agent that operates a computer GUI. For each step, you are given the task goal, the latest screenshot, and the history of earlier actions. Choose the single best next move that advances the task based on the current screen. Keep your action consistent with what is visible, and use the action history to avoid repeating mistakes. The password of the computer is `{password}`.

For each step, provide your response in this format:

```
## Thought: {thought}
## Action:  {action}
## Code:    {code}
```

Code rules:
- Return exactly one executable code block.
- The code block must contain either:
1) pyautogui code for the next GUI action, or
2) one built-in control call:
- computer.wait(): wait 20 seconds if the page, app, or system needs time
- computer.terminate(status=..., answer=...): end the task when it is finished or cannot be completed
- If you use computer.terminate, status must be "success" or "failure".
- Include answer only when there is a final result to report.

Figure 9: System prompt used at evaluation. Thinking-capable backbones receive a variant in which the *Thought* field is replaced by an explicit reasoning segment wrapped in `[think]`/`[/think]` tags.

J Detailed Analysis of Training Data

J.1 Distribution of Training Data

To provide a clearer overview of our dataset, we present the detailed distribution of the training data across diverse application categories in Table 9. The training set covers 9 specific applications alongside complex multi-application workflows, ensuring high diversity for model training.

Table 9: Distribution of the training data across different application categories.

Category	Applications	#Tasks	Ratio
OS	os	88	6.1%
Office	calc, impress, writer	458	32.0%
Daily	chrome	188	13.1%
Professional	gimp, thunderbird, vlc, vs_code	287	20.0%
Workflow	multi_apps	411	28.7%
Total	9 apps + multi-app	1,432	100.0%

J.2 Ablation Study on Mixed Trajectories Training

To analyze the individual contribution of different data types in *Stage IV: Mixed Trajectories Training*, we categorize the training data into three components based on their respective roles:

1. **Base recipe**: Provides stable GUI action formats and broad procedural knowledge.
2. **Verified successful plain-policy trajectories**: Supplements successful behaviors under the current model's own distribution.
3. **SGCD continuations**: Supplies supervision from policy-induced off-trajectory states.

Table 10 presents the ablation results for Qwen3-VL-8B from version v0 to v1. The overall improvement in Success Rate (SR) demonstrates the benefits of incorporating successful plain-policy trajectories. More importantly, the substantial gain in Continuation Success Rate (Cont. SR) primarily stems from the introduction of post-handoff continuation data. This directly validates our core motivation: SGCD effectively provides crucial supplementary supervision for policy-induced off-trajectory states, rather than merely scaling up successful trajectory data.

Table 10: Ablation study on different training data mixtures for Qwen3-VL-8B from v0 to v1.

Training Mixture	SR (%)	Cont. SR
Base Recipe	38.2	18.9
+ Successful Plain-Policy	41.0	21.1
+ SGCD Continuations	44.6	28.4
+ Both (Full Mixture)	**45.7**	**29.8**