CoAdapt-GUI: Joint Workflow Context and Policy Adaptation for Unseen GUI Applications

Anonymous authors
Paper under double-blind review

Abstract

To address the brittleness of mobile GUI agents on unseen applications, we study test-time adaptation using limited target interaction without target demonstrations. We introduce CoAdapt-GUI, a test-time adaptation (TTA) framework that jointly adapts structured workflow context and policy from the agent's own target-app rollouts and rewards. This context guides target adaptation with transferable procedures, failure modes, and verification rules while excluding app-bound source state. For policy adaptation, task–context-matched group-relative optimization updates a LoRA adapter on a frozen vision–language model. Across two unseen-app evaluations, CoAdapt-GUI reaches 45.0% on AndroidWorld-Generalization, compared with 37.08% for the reproduced Policy-Only TTA baseline, and raises AndroidWorld Plus performance from 37.9% to $\mathbf{52.6}\%$. These results show that transfer-constrained workflow context provides substantial gains and that joint policy adaptation further improves held-out performance.

1 Introduction

Mobile GUI agents execute natural-language instructions by interpreting application interfaces and carrying out multi-step actions. Recent advances in GUI grounding, trajectory synthesis, and reinforcement learning have substantially improved their performance on established benchmarks (Cheng et al., 2024; Hong et al., 2024; Cheng et al., 2026; Xu et al., 2025). However, these evaluations generally assume a predefined collection of applications and workflows (Rawles et al., 2025; Xu et al., 2024; Kong et al., 2026). In deployment, agents may encounter applications whose interface and task procedures were absent from training. Adapting to such applications from limited interaction is therefore essential for effective operation beyond the original training environment.

This challenge is particularly pronounced under cross-application shifts. AndroidWorld-Generalization distinguishes generalization to unseen task instances, templates, and applications. Source-side online RL improves a 7B policy by 26.1 percentage points on unseen instances, but the gain drops to 8.3 points on unseen applications (Gu et al., 2026). Success on new tasks within a familiar interface therefore does not imply effective behavior in an unfamiliar application.

An unseen application can expose both interface-specific and procedural gaps. An agent may understand the goal but fail to ground actions to unfamiliar interface elements, or it may execute individual actions correctly yet lack the workflow and completion conditions needed to finish the task. Updating the policy can change how the agent interprets and acts on the new interface, while adapting an external workflow context can preserve procedures learned from target-side experience. Updating only one state may therefore leave an important source of failure unresolved.

Existing work only partially addresses this target-side adaptation problem. AndroidWorld-Generalization updates the policy from the agent's own target-app rollouts while leaving its workflow context fixed (Gu et al., 2026). UI-Mem jointly learns experience memory and policy during source-side training, then transfers the resulting knowledge to unseen applications (Xiao et al., 2026). This leaves open whether an agent can use interactions collected after encountering a new application to adapt both its workflow context and policy before evaluation on held-out target tasks.

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.

We introduce **CoAdapt-GUI**, a test-time adaptation (TTA) framework that maintains and jointly updates these two states from target-app rollouts and task-level rewards. Its context channel contrasts successful and failed traces to revise transferable procedures, failure patterns, and completion checks while excluding app-bound details. Its policy channel updates a lightweight LoRA adapter through a task–context-conditioned group-relative objective (Shao et al., 2024), while keeping the VLM backbone frozen. Both updates are derived from the same rollout groups within each adaptation round, and the resulting context and policy are frozen before evaluation on held-out target tasks.

We evaluate CoAdapt-GUI under two levels of unseen-app generalization. AndroidWorld-Generalization (Gu et al., 2026) evaluates new instances of task types encountered during target adaptation. In this setting, CoAdapt-GUI reaches 45.0%, exceeding the reproduced Policy-Only TTA baseline of 37.08% by 7.92 percentage points. To test whether adaptation transfers beyond the task types encountered during target interaction, we construct AndroidWorld Plus with disjoint adaptation and evaluation task types within each target app. Here, Context-Only TTA improves the Base Policy from 37.9% to 47.2%, while CoAdapt-GUI further raises success to **52.6%**. These results show that target-grounded workflow adaptation provides substantial gains and that jointly adapting the policy achieves the best overall performance in both settings.

This paper makes three contributions:

- **Joint target-side adaptation from autonomous interaction.** We introduce a test-time adaptation framework that updates separate workflow-context and policy states using only the agent's own target-app rollouts and task rewards, without target demonstrations or access to held-out evaluation signals.
- **Transfer-constrained context–policy adaptation.** We separate transferable workflow knowledge from app-bound source state and coordinate two reward-guided updates from the same target interactions. Validated reward differences revise the workflow context, while policy credit is computed only among rollouts sharing the same task and context condition.
- **Evaluation across instance- and task-type generalization.** We evaluate CoAdapt-GUI on the released AndroidWorld-Generalization unseen-app split and construct AndroidWorld Plus to hold out entire task types during target adaptation. CoAdapt-GUI reaches 45.0% and 52.6% in the two settings, respectively, achieving the best overall result in both settings.

2 Related Work

2.1 Mobile GUI Agents and Generalization

Mobile GUI agents commonly combine a vision–language policy with an interaction loop for action execution and task verification. GUI-specific pretraining and high-resolution perception improve visual grounding and action prediction (Cheng et al., 2024; Hong et al., 2024; Gou et al., 2024), while OpenMobile and MobileRL scale supervised or online policy learning from executable trajectories (Cheng et al., 2026; Xu et al., 2025). UI-Mem also evaluates zero-shot transfer to held-out applications by retrieving hierarchical memory accumulated during source-side policy training (Xiao et al., 2026). These approaches acquire transferable behavior on source applications before evaluation; we instead study how an agent can continue adapting after encountering an unseen application.

AndroidWorld, AndroidLab, and MobileWorld provide programmatically evaluated environments for studying GUI agents across apps and multi-app workflows (Rawles et al., 2025; Xu et al., 2024; Kong et al., 2026). Related studies examine generalization across websites, tasks, applications, and app categories (Deng et al., 2023; Li et al., 2024). Most closely, AndroidWorld-Generalization separates unseen instances, templates, and apps, and shows that policy adaptation on target-app interactions can improve unseen-app performance (Gu et al., 2026). We use its released setting for comparison, but adapt both workflow context and the policy from collected target rollouts.

2.2 Test-Time Context and Policy Adaptation

Test-time adaptation updates a deployed model using data from its target environment (Sun et al., 2020; Wang et al., 2021). Recent methods extend this idea to language-model reasoning and inter-

active agents (Zuo et al., 2025; Zweiger et al., 2025; Chen et al., 2026). Few-shot GUI methods such as LearnAct and AdaptAgent also use target experience, but rely on demonstrations rather than the agent's own reward-bearing interaction (Liu et al., 2025a; Verma et al., 2024). Our setting instead uses executable rewards from autonomously collected trajectories and evaluates adaptation on held-out target tasks.

External memory provides a complementary in-context adaptation channel. AppAgent records explored app functionality, Agent Workflow Memory abstracts reusable routines, and Mobile-Agent-E and MobiMem accumulate experience as evolving notes or prompt memory (Zhang et al., 2025; Wang et al., 2024; 2025; Liu et al., 2025b). These methods demonstrate the value of explicit workflow knowledge while generally keeping the underlying policy fixed. Unlike UI-Mem, which varies memory guidance within each policy-optimization group to internalize guided behavior (Xiao et al., 2026), CoAdapt-GUI treats context and policy as separate target-side states. Reward differences across context variants revise the workflow state, while policy credit is computed only among rollouts sharing the same task and context condition. Agent-SAMA additionally represents app execution as a finite-state machine (FSM) for planning and recovery (Guo et al., 2026). In contrast, CoAdapt-GUI uses FSM-grounded context to constrain which workflow knowledge may transfer across applications, excluding app-bound source state from transfer.

E-SPL jointly evolves system prompts and updates policy weights from shared rollouts (Zhang et al., 2026). CoAdapt-GUI introduces transfer-constrained workflow evolution for unseen-app adaptation. Its context state separates an immutable, eligibility-filtered source-workflow prior from a mutable target-grounded component. App-bound source states are excluded from retrieval, and target experience updates only the target component and the policy. Context revisions are applied through typed operations restricted to designated transferable-workflow fields; app-bound operations and revisions that leave the state unchanged are rejected. These constraints define the admissible context states and revision operators for cross-application adaptation, while retaining the shared-rollout coupling between context evolution and policy learning.

3 CoAdapt-GUI: Test-Time Context–Policy Adaptation

We study test-time adaptation to an unseen GUI application through two complementary states: workflow context M_t guides procedures, failure handling, and completion checks, while policy parameters θ_t govern visual–action behavior. Target rollouts provide executable feedback for updating both. Co-adapting these states introduces two forms of interference. First, source experience may mix portable workflow knowledge with application-specific screens, identifiers, geometry, and navigation paths, causing negative transfer on an unseen interface. Second, changes to the workflow context alter the effective policy input, so reward differences across context conditions need not reflect differences in policy behavior.

We propose CoAdapt-GUI, which combines transfer-constrained workflow adaptation with task–context-matched policy learning. The context channel prevents app-bound source details from crossing application boundaries and stores target-grounded revisions separately. The policy channel computes relative advantages only among rollouts sharing the same task and context condition. The updates are coupled through interaction: the current workflow state shapes the trajectories used for policy learning, while the current policy generates the successes and failures used to revise workflow knowledge. Figure 1 summarizes the resulting adaptation loop.

3.1 Test-Time Adaptation Problem

Let $\mathcal{A}_{\mathrm{src}}$ and $\mathcal{A}_{\mathrm{tgt}}$ denote the source and target application sets. In the strict unseen-app setting, $\mathcal{A}_{\mathrm{src}} \cap \mathcal{A}_{\mathrm{tgt}} = \varnothing$. More generally, target trajectories never enter source initialization, and application-specific interaction knowledge from one application is not transferred to another.

For test-time adaptation, tasks are partitioned into disjoint adaptation and evaluation sets, $\mathcal{Q}^{\mathrm{adapt}}$ and $\mathcal{Q}^{\mathrm{eval}}$. Only tasks in $\mathcal{Q}^{\mathrm{adapt}}$ may update the deployed agent; evaluation tasks and outcomes remain unavailable throughout adaptation.

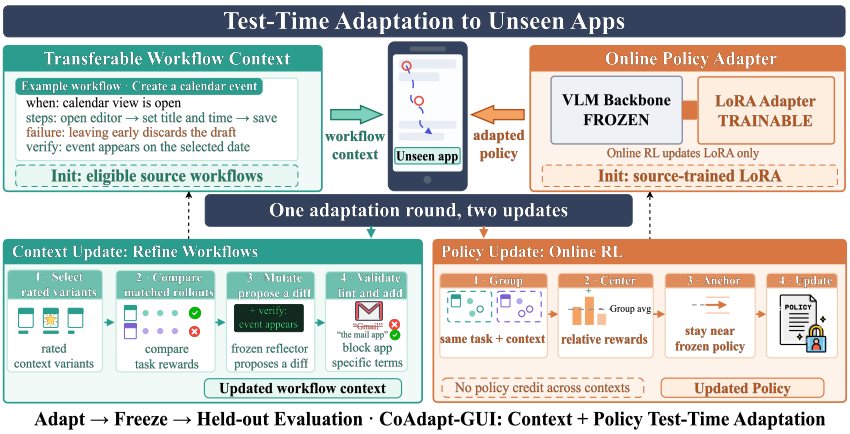

Figure 1: CoAdapt-GUI uses matched target-app rollouts to refine transferable workflow context and update a lightweight policy adapter on a frozen VLM. Context-Only TTA executes only the context branch, whereas CoAdapt-GUI executes both branches using the same rollout batches. The adapted context and policy are frozen before held-out evaluation.

The workflow state for task q combines an immutable source workflow context $M_0(q)$ with a target-grounded state M_t^{tgt}:

$$M_t(q) = \left(M_0(q), M_t^{\mathrm{tgt}}\right), \quad C_t(q) = \mathrm{Render}(M_t(q), q)\,.$$

At interaction step k, the effective policy conditions on the task, current observation o_k, interaction history h_k, and rendered workflow context:

$$u_k \sim \pi_{\theta_t}\left(\cdot \mid q, o_k, h_k, C_t(q)\right).$$

Executing the action sequence produces a trajectory $\tau = \{(o_k, u_k)\}_{k=1}^{K}$ and a task reward $r(\tau)$.

The parametric state θ_t contains the trainable LoRA parameters of an otherwise frozen vision–language policy. Together with the target environment, $M_t(q)$ and θ_t induce the rollout distribution

$$\tau \sim d(\cdot \mid q, M_t(q), \theta_t)\,.$$

Updating either state therefore changes the interaction data available for updating the other.

Given an interaction budget B, adaptation returns final states $(M^\star, \theta^\star)$. Both are frozen before evaluation, and their generalization is estimated only on $\mathcal{Q}^{\mathrm{eval}}$. Adaptation rewards are therefore training signals rather than the final evaluation objective. Source preparation trains a shared LoRA adapter on source-app trajectories with workflow context held fixed. Target adaptation initializes the policy from a source-side checkpoint selected using source metadata; target trajectories and held-out outcomes never influence source construction or checkpoint selection.

3.2 Transfer-Constrained Workflow Context

Source trajectories mix reusable procedural knowledge with application-specific interaction details. Procedures, failure conditions, and completion checks may remain useful across applications with similar functionality, whereas screen layouts, visible strings, resource identifiers, coordinates, and source-specific navigation paths may cause negative transfer.

We represent each transferable workflow entry as $w = \langle c, P, F, V \rangle$, where c specifies when the workflow applies, P describes an abstract procedure, F records failure or recovery conditions, and V specifies observable or executable completion checks. These fields are stored in a typed schema so that applicability, execution guidance, failure handling, and verification can be validated and revised separately.

For each source application a, we construct a finite-state machine (FSM) grounded workflow context with two components: an app-bound state M_{app}^a and a transferable state M_{tr}^a. The app-bound state

is instantiated as a screen-transition FSM recording concrete screens, action-conditioned transitions, visible interface cues, and resource-level information. This representation makes the expected effect of each action explicit, supporting the identification of failed transitions and possible recovery paths (Guo et al., 2026). The transferable state contains workflow entries abstracted from the source trajectories used to construct the FSM, capturing procedures, failure conditions, and completion checks rather than source-specific screens or transitions.

An eligibility predicate, $\mathrm{Eligible}_{\mathrm{tr}}(w) \in \{0, 1\}$, determines which entries may cross application boundaries. Schema validation checks the workflow structure, while a hard linter rejects identifiable source-app strings, package or resource identifiers, and app-specific state identifiers. Constraints on concrete UI labels, coordinates, launcher behavior, and task-instance values are specified in the prompts (Appendix F). Eligibility does not imply that an entry will help the target; it excludes identifiable source-specific information, while target rollout rewards subsequently determine utility.

A frozen synthesizer constructs the app-bound and transferable source states from audited trajectories. Let $\mathcal{W}_{\mathrm{src}}$ denote the set of workflow entries w such that $w \in M^a_{\mathrm{tr}}$ for some source application $a \in \mathcal{A}_{\mathrm{src}}$ and $\mathrm{Eligible}_{\mathrm{tr}}(w) = 1$. These entries are grouped by functional category and consolidated into the source library

$$\mathcal{L}_{\mathrm{src}} = \mathrm{Consolidate}\left(\mathcal{W}_{\mathrm{src}}\right).$$

For target task q, initialization retrieves bounded, eligibility-constrained protocol-permitted source workflow context, $M_0(q) = \mathrm{Retrieve}(\mathcal{L}_{\mathrm{src}}, q)$. Strict unseen-app evaluation excludes all app-bound source states. If the target belongs to a functional category absent from the source pool, retrieval returns an empty prior rather than forcing unrelated guidance. The source library remains fixed during target adaptation. Target interactions instead maintain the separate target-grounded state M^{tgt}_t. Rendering selects task-relevant entries from the source workflow context and target-grounded state while preserving their provenance. This separation prevents target adaptation from overwriting reusable source knowledge while allowing target experience to refine or supplement the guidance used during execution.

3.3 Joint Context and Policy Adaptation

Target interactions can update both the workflow state and the policy parameters. At adaptation round t, the controller selects a task $q_t \in \mathcal{Q}^{\mathrm{adapt}}$ and one or more already materialized context variants from the current population. It then collects matched rollouts under controlled reset conditions:

$$D_t = \{(\tau_j, r_j, \kappa_j, z_j)\}_{j=1}^{N_t}.$$

Here, κ_j identifies the context variant that generated trajectory τ_j, and z_j records its reset identity. Only context variants present before collecting D_t can receive rating updates from their rewards. The trajectories are also added to the policy buffer $\mathcal{B}$ with their task and context identities.

The two channels reuse the same rollout stream but operate at different frequencies. Each matched batch updates the ratings of the evaluated context variants and may produce a validated child for subsequent evaluation. The LoRA parameters are updated only when $\mathcal{B}$ contains valid task–context comparison groups. The updates are therefore interaction-coupled rather than jointly differentiable: the current context shapes the trajectories used for policy learning, while the current policy determines the successful and failed behaviors available for future context revisions.

Reward-guided context update. For the repeated seeded tasks used in our evaluation, the controller maintains a population of TrueSkill-rated context variants (Herbrich et al., 2006). At each round, it samples already materialized variants, evaluates them on matched tasks and resets seeds, and updates their ratings using the resulting task rewards. A frozen reflector contrasts the evaluated variants' success and failure traces and proposes a typed revision to a parent selected by the current round's mean task reward. The acting policy collects the target-app rollouts, while the frozen external LLM proposes workflow revisions from these traces and their executable rewards.

The resulting child enters the population only after typed-diff parsing, transferable-state scope filtering, and a check that the applied revision changes the parent state; the caller records its provenance (Appendix F). Each child inherits its parent's pre-round rating with increased uncertainty and becomes eligible for final selection only after a subsequent rollout-based rating update. At budget

exhaustion, selection considers all evaluated historical variants, prioritizes the posterior mean, and falls back to the root if none has been evaluated. Ratings persist across policy updates without final-policy reranking; Appendix C.3 specifies initialization, eligibility, and tie handling. Appendix D illustrates excluded source-app details and an accepted workflow revision evaluated in subsequent adaptation rollouts.

Task–context-matched policy update. The policy update learns online from trajectories generated by the current policy rather than from a fixed dataset. Because the context condition changes the effective policy input, trajectories are partitioned into groups sharing both task and context:

$$G(q, \kappa) = \{j \mid q_j = q,\ \kappa_j = \kappa\}.$$

For a group G of size n_G, we compute

$$\bar{r}_G = \frac{1}{n_G} \sum_{j \in G} r_j, \qquad A_j = \frac{r_j - \bar{r}_G}{s_G},$$

where s_G is the normalization factor: it is set to one for mean-centered advantages, or to the within-group standard deviation plus a small constant for standardized advantages. Singleton and constant-reward groups provide no active policy gradient.

For trajectory τ_j, we average log-probabilities over its action-generation units:

$$\ell_j(\theta) = \frac{1}{|\tau_j|} \sum_{m \in \tau_j} \log \pi_\theta \left(u_{j,m} \mid x_{j,m}\right),$$

where $x_{j,m}$ contains the task, observation, history, and rendered context available when action unit $u_{j,m}$ is generated. Let $\mathcal{B}_{\text{act}} \subseteq \mathcal{B}$ contain trajectories belonging to nondegenerate groups. The resulting online group-relative objective (Shao et al., 2024) is

$$\mathcal{L}_{\text{policy}} = -\frac{1}{|\mathcal{B}_{\text{act}}|} \sum_{j \in \mathcal{B}_{\text{act}}} A_j \ell_j(\theta) + \beta \mathcal{R}\left(\theta; \pi_{\text{anchor}}\right),$$

where $\mathcal{R}$ denotes the frozen-policy regularizer when anchoring is enabled. For AndroidWorld Plus, we use GRPO-style group-relative advantages with an on-policy policy-gradient objective, without PPO-style policy-ratio clipping; Appendix C.4 specifies the K3 anchor and full update loss. The vision–language backbone remains frozen, and only the LoRA parameters are optimized (Hu et al., 2021). Restricting comparisons to $G(q, \kappa)$ controls variation due to task difficulty and contextual guidance when computing relative advantages. The policy buffer $\mathcal{B}$ stores only trajectories collected since the most recent LoRA update. The policy remains fixed while these trajectories are collected. Once the update threshold is reached, at most one LoRA update is attempted, and the buffer is cleared, regardless of whether it contains an active comparison group. No trajectory is reused after the policy changes. Each trajectory retains the task and rendered-context identity under which it was collected. Algorithm 1 summarizes the complete schedule.

4 Experiments

We evaluate CoAdapt-GUI in two unseen-app settings. **(1) AndroidWorld-Generalization** follows the released setup and tests adaptation to new instances of task templates encountered during adaptation, enabling direct comparison with the reproduced Policy-Only TTA baseline (Gu et al., 2026). **(2) AndroidWorld Plus**, an extension of AndroidWorld (Rawles et al., 2025), separates adaptation and evaluation by task template to test transfer to new task types within unseen apps.

4.1 Experimental Setup

Each experiment setting separates source data, target adaptation tasks, and held-out target evaluation tasks. Source data is used to obtain the initial policy and reusable workflow context. During target adaptation, the agent may update its workflow context, policy, or both, depending on the configuration. All updates stop before evaluation; no held-out task, trajectory, reward, or outcome is used for adaptation or checkpoint selection. A *task template* defines a reusable task type, whereas a *task instance* supplies its concrete arguments. Our primary metric is success rate (SR), defined as the percentage of held-out evaluation episodes successfully completed according to the benchmark's programmatic evaluator.

Algorithm 1 CoAdapt-GUI test-time context–policy adaptation

Require: Source workflow library $\mathcal{L}_{\mathrm{src}}$, policy initialization θ_0, adaptation tasks $\mathcal{Q}^{\mathrm{adapt}}$, frozen reflector, interaction budget B
Ensure: Frozen workflow state $M^\star$ and policy parameters $\theta^\star$
Retrieve the protocol-permitted source workflow context M_0
Initialize $M^{\mathrm{tgt}} \leftarrow \varnothing$, $\theta \leftarrow \theta_0$, and $\mathcal{B} \leftarrow \varnothing$
while interaction budget remains **do**
    Select $q \in \mathcal{Q}^{\mathrm{adapt}}$ and already materialized context variants for q
    Collect matched rollouts D with task, context, and reset identities
    Cache pre-update ratings; update participant ratings and evaluation counts using D
    Propose and validate a child revision from D
    If valid, admit the child with its parent's cached rating, increased uncertainty, and zero evaluations
    Add D to $\mathcal{B}$
    if LoRA adaptation is enabled and $\mathcal{B}$ is ready **then**
        Update LoRA from nondegenerate task–context groups in $\mathcal{B}$, if any
        Clear $\mathcal{B}$
    end if
end while
Select $M^\star$ from all evaluated historical variants by posterior mean (Appendix C.3); use the root if none is evaluated
Set $\theta^\star \leftarrow \theta$; freeze and **return** $(M^\star, \theta^\star)$

Compared methods. We mainly compare five configurations in each setting. **The Base Policy** uses the initial policy without workflow context or a target-side update. **Static Context Transfer** supplies source workflows as frozen context. **Policy-Only TTA** keeps the workflow context fixed and updates only the policy adapter from target adaptation rollouts. **Context-Only TTA** keeps the policy fixed and builds additional workflow context from target adaptation rollouts. **CoAdapt-GUI** also updates a LoRA policy adapter while freezing the VLM backbone. These configurations compare source workflow transfer, target-side context adaptation, policy adaptation, and their joint use, and also form a component ablation of the two adaptation channels. Context-Only TTA and CoAdapt-GUI receive the same adaptation tasks and rollout budget, but are run independently and can therefore collect different policy-dependent trajectories; their difference reflects the complete joint procedure, not a controlled estimate of the LoRA update alone. Experiments were run on NVIDIA H200 GPUs with 141 GB of memory; one 20-round target-app adaptation run required approximately 9–10 GPU-hours on average.

4.2 New Task Instances in Unseen Apps

Setup. We follow the released unseen-app setting of AndroidWorld-Generalization (Gu et al., 2026). Its source split contains 12 apps, 62 task templates, and 905 training instances. All configurations start from the released step-500 UI-TARS-7B checkpoint (Qin et al., 2025), trained on this split. The target split contains five disjoint apps, with eight adaptation instances per app (40 total) and 48 held-out evaluation instances. Adaptation and evaluation cover the same 16 templates but use instances generated with non-overlapping seeds, thereby testing transfer to new instances of task types seen during adaptation.

We use the released manifests throughout. All adaptive methods use the same eight adaptation instances and a nominal 50-step schedule per target app, and all methods are evaluated on the same 48 held-out instances. We reproduce both the Base Policy and Policy-Only TTA baselines from AndroidWorld-Generalization (Gu et al., 2026) in our evaluation pipeline, with five evaluation runs per configuration.

Results. The reproduced Base Policy and Policy-Only TTA achieve $(27.08 \pm 2.55)\%$ and $(37.08 \pm 3.09)\%$, respectively. Our CoAdapt-GUI configuration performs best at $\mathbf{45.00\% \pm 1.86}$, 7.92 percentage points above the reproduced Policy-Only TTA result. Context-Only TTA reaches $35.00\% \pm 1.74$, while Static Context Transfer reaches $28.75\% \pm 2.28$. Under the same adapta-

Table 1: **Setting 1—New Task Instances in Unseen Apps (AndroidWorld-Generalization).** Success rate (SR) is measured on 48 held-out instances of task templates used during target adaptation. All values are the mean ± standard deviation over five evaluation runs in our pipeline, including the reproduced Base Policy and Policy-Only TTA baselines (Gu et al., 2026).

Method	Config.	Adapt.	Steps/app	Eval.	SR (%)	Source
The Base Policy (Gu et al., 2026)	no TTA	0	0	48	27.08 ± 2.55	Reproduced (5 runs)
Static Context Transfer	static context	0	0	48	28.75 ± 2.28	Ours (5 runs)
Policy-Only TTA (Gu et al., 2026)	policy only	40	50/app	48	37.08 ± 3.09	Reproduced (5 runs)
Context-Only TTA	context only	40	50/app	48	35.00 ± 1.74	Ours (5 runs)
CoAdapt-GUI	context + policy	40	50/app	48	$\mathbf{45.00 \pm 1.86}$	Ours (5 runs)

tion instances and rollout budget, CoAdapt-GUI outperforms Context-Only TTA by 10.00 points, indicating that co-adapting workflow context and the policy is more effective than adapting context alone in this setting.

4.3 New Task Templates in Unseen Apps

Setup. To evaluate transfer to task templates not observed during adaptation, we construct AndroidWorld Plus by extending AndroidWorld (Rawles et al., 2025) with three apps from B-MoCA (Lee et al., 2026) and three from AndroidLab (Xu et al., 2024). After executable-task filtering, the benchmark contains 25 apps and 191 task templates. We assign 12 apps with 96 templates to the source set and the remaining 13 apps with 95 templates to the disjoint target set. Appendix A documents filtering criteria, full task manifests, and per-app template and episode counts (Table 4).

All AndroidWorld Plus configurations use Qwen3-VL-8B-Instruct (Bai et al., 2025) with a frozen backbone and start from a LoRA adapter trained only on the 12 source apps. Checkpoint selection occurs before target adaptation without using target rollouts or rewards, and each matched comparison uses the same initialization. Policy-Only TTA updates only the LoRA adapter while keeping its initial source context fixed.

Within each target app, adaptation and evaluation use disjoint task templates. Across the target apps, 60 templates yield a pool of 300 adaptation instances, while 35 held-out templates yield 105 evaluation episodes using non-overlapping seeds. Each adaptive configuration runs for 20 rounds per app. Context-Only TTA, Policy-Only TTA, and CoAdapt-GUI collect four rollouts per selected task–context condition ($\leq$ 160 per app). Context-Only TTA uses its batches only for workflow adaptation, Policy-Only TTA uses them only for LoRA adaptation, and CoAdapt-GUI uses them for both. Supplementary Section C provides the remaining adaptation details.

Source-category coverage. Following Android Control (Li et al., 2024), we use Google Play categories as a coarse, externally defined boundary for source-workflow retrieval. Source workflows are eligible only when their app category matches that of the target app. Under this protocol, six target apps are **Category-Shared** and can retrieve source workflows, whereas seven are **Category-Novel**, for which retrieval returns no source workflow. Static Context Transfer therefore supplies no workflow context for Category-Novel Apps. Context-Only TTA and CoAdapt-GUI instead initialize an empty target-grounded workflow state, construct workflow entries from target adaptation rollouts and rewards, and refine them in subsequent rounds. This breakdown assesses whether target-side adaptation remains effective without category-matched source experience.

Results. Our CoAdapt-GUI performs best, raising overall success from 37.9% to 52.6%, a gain of 14.7 percentage points. The cumulative context path—Static Context Transfer followed by Context-Only TTA—reaches 47.2%, 9.3 points above the Base Policy, whereas Policy-Only TTA with fixed source context reaches 40.4%, 3.2 points below Static Context Transfer. CoAdapt-GUI is a further 5.3 points above Context-Only TTA. This pattern is consistent with evolved workflow guidance producing more informative target experience for policy learning, although the independent-rollout comparison does not isolate this mechanism.

Table 2: **Setting 2—New Task Templates in Unseen Apps (AndroidWorld Plus).** Success rate (SR) is measured over 105 held-out episodes whose task templates are disjoint from target adaptation. All rows report mean ± sample standard deviation over five runs. Category-Shared and Category-Novel group episodes by whether the target app's category is seen in the source pool.

Method	Config.	Adapt.	Steps/app	SR (%)		
				Category-Shared Apps	Category-Novel Apps	Overall
The Base Policy	no TTA	0	0	47.04 ± 2.11	28.24 ± 1.07	37.90 ± 1.24
Static Context Transfer	static context	0	0	57.41 ± 2.27	29.02 ± 1.64	43.62 ± 1.24
Policy-Only TTA	policy only	300	20/app	53.70 ± 3.46	26.27 ± 2.24	40.38 ± 1.97
Context-Only TTA	context only	300	20/app	62.96 ± 2.27	30.59 ± 2.24	47.24 ± 1.09
CoAdapt-GUI	context + policy	300	20/app	$\mathbf{70.37 \pm 2.27}$	$\mathbf{33.73 \pm 1.64}$	$\mathbf{52.57 \pm 1.56}$

On Category-Shared Apps, CoAdapt-GUI reaches 70.4%, compared with 53.7% for Policy-Only TTA and 63.0% for Context-Only TTA. Category-Novel Apps provide a built-in control for source transfer. Because retrieval is empty in this group, Static Context Transfer uses the same fixed-policy, no-workflow-context configuration as the Base Policy. Policy-Only TTA falls to 26.3%, whereas Context-Only TTA and CoAdapt-GUI improve success to 30.6% and 33.7%. In a representative Chrome trace, an evolved workflow achieved a mean task reward of 0.25 over four rollouts in round 12, versus zero for the root context under the matched condition (Appendix D, Table 11).

5 Limitations

Evaluation scope and leakage control. Target-side adaptation can overestimate generalization through leakage or evaluation on overly similar tasks. We mitigate leakage by separating source and target apps, restricting updates and model selection to adaptation tasks, and freezing both states before held-out evaluation. AndroidWorld-Generalization evaluates new instances of adaptation templates, while AndroidWorld Plus holds out entire templates to test transfer beyond the task types encountered during adaptation.

Transfer reliability and adaptation stability. Source workflows may contain interface-specific assumptions or reflection errors that cause negative transfer. Limited target interaction can also yield noisy or uniform rewards, obscuring context utility and policy credit. Before transfer, we exclude app-bound source state and apply eligibility, schema, provenance, and lint checks. We evaluate candidate revisions on matched tasks and reset seeds in subsequent rollouts; rewards update their ratings and guide final selection. Policy advantages are computed within task–context groups, excluding groups without reward variation. Clearing the on-policy buffer after each update attempt prevents reuse of stale trajectories, while LoRA training on a frozen backbone confines parameter updates to the adapter.

Deployment safety and continual use. Our evaluation assumes resettable emulators and programmatic task rewards, which may be unavailable in arbitrary new apps. Privacy risks, irreversible actions, and stability under continual adaptation remain concerns beyond the bounded adaptation protocol studied here.

6 Conclusion

We introduced CoAdapt-GUI, a test-time adaptation framework that updates workflow context and policy on unseen applications using the agent's own target rollouts and task rewards. CoAdapt-GUI reaches 45.0% on AndroidWorld-Generalization, compared with 37.08% for the reproduced Policy-Only TTA baseline, and improves AndroidWorld Plus from 37.9% to 52.6%. Context adaptation improves performance without policy updates in both settings, while CoAdapt-GUI achieves the strongest overall results, supporting context and policy adaptation as distinct but complementary channels for novel-app generalization.

AI USE STATEMENT

Generative AI tools were used to assist with code implementation, suggest and assist with figure revisions, format author-selected references as BibTeX entries, and polish the language and readability of a manuscript draft written by the authors. The authors visually inspected figure revisions, checked reference entries against the selected sources, and reviewed the edited manuscript. The authors take responsibility for the code, figures, references, and all methods, results, claims, and text in the final submission.

ETHICS STATEMENT

Our experiments are conducted in resettable Android emulators using designated benchmark tasks and bounded interaction budgets. Adapted workflow states and policy adapters are frozen before held-out evaluation. Autonomous GUI agents can expose sensitive information, modify application data, or execute unintended actions; interface content may also contain malicious instructions. The experimental setting does not establish safety on personal devices or immunity to prompt injection. Deployment would require user-authorized action boundaries, appropriate protection of observations and traces sent to external models, and confirmation and recovery mechanisms for consequential actions.

REPRODUCIBILITY STATEMENT

Section 3 and Algorithm 1 describe the adaptation procedure, and Section 4 specifies the comparison configurations and evaluation protocol. Appendix A provides the AndroidWorld Plus construction rules, complete task manifests, app-level allocations, and adaptation and evaluation seeds. Appendix B documents the AndroidWorld-Generalization protocol and the provenance of reported results. Appendix C records model initialization and the AndroidWorld Plus adaptation configuration, including LoRA hyperparameters and the policy-update objective. Appendices D and F provide workflow examples, prompt templates, and validation contracts. The source code is provided as a ZIP archive in the supplementary material.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025. URL https://arxiv.org/abs/2511.21631.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Test-time adaptation for LLM agents via environment interaction. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=OH4PE0TDo0.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2401.10935.

Kanzhi Cheng, Zehao Li, Zheng Ma, Nuo Chen, Jialin Cao, Qiushi Sun, Zichen Ding, Fangzhi Xu, Hang Yan, Jiajun Chen, Anh Tuan Luu, Jianbing Zhang, Lewei Lu, and Dahua Lin. Openmobile: Building open mobile agents with task and trajectory synthesis, 2026. URL https://arxiv.org/abs/2604.15093.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.06070.

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

Li Gu, Zihuan Jiang, Zhixiang Chi, Huan Liu, Ziqiang Wang, Yuanhao Yu, Glen Berseth, and Yang Wang. Generalization in online reinforcement learning for mobile agents. *arXiv preprint arXiv:2603.07432*, 2026. URL https://arxiv.org/abs/2603.07432.

Linqiang Guo, Wei Liu, Yi Wen Heng, Tse-Hsun (Peter) Chen, and Yang Wang. Agent-sama: State-aware mobile assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. URL https://arxiv.org/abs/2505.23596.

Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: A bayesian skill rating system. In B. Schölkopf, J. Platt, and T. Hoffman (eds.), *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/f44ee263952e65b3610b8ba51229d1f9-Paper.pdf.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. URL https://arxiv.org/abs/2106.09685.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6142–6167, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.278. URL https://aclanthology.org/2026.acl-long.278/.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. Benchmarking mobile device control agents across diverse configurations. In Sarath Chandar, Razvan Pascanu, Eric Eaton, Bing Liu, Rupam Mahmood, and Amal Rannen-Triki (eds.), *Proceedings of The 4th Conference on Lifelong Learning Agents*, volume 330 of *Proceedings of Machine Learning Research*, pp. 261–306. PMLR, 11–14 Aug 2026. URL https://proceedings.mlr.press/v330/lee26a.html.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. *arXiv preprint arXiv:2406.03679*, 2024. URL https://arxiv.org/abs/2406.03679.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Shuai Ren, Hao Wang, Shibo He, and Wenchao Meng. Learnact: Few-shot mobile GUI agent with a unified demonstration benchmark. *arXiv preprint arXiv:2504.13805*, 2025a. URL https://arxiv.org/abs/2504.13805.

Zibin Liu, Cheng Zhang, Xi Zhao, Yunfei Feng, Bingyu Bai, Dahu Feng, Erhu Feng, Yubin Xia, and Haibo Chen. Beyond training: Enabling self-evolution of agents with MOBIMEM. *arXiv preprint arXiv:2512.15784*, 2025b. URL https://arxiv.org/abs/2512.15784.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin,

Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025. URL https://arxiv.org/abs/2501.12326.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y.K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/1909.13231.

Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. Adaptagent: Adapting multimodal web agents with few-shot learning from human demonstrations. *arXiv preprint arXiv:2411.13451*, 2024. URL https://arxiv.org/abs/2411.13451.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://arxiv.org/abs/2006.10726.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*, 2025. URL https://arxiv.org/abs/2501.11733.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024. URL https://arxiv.org/abs/2409.07429.

Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents, 2026. URL https://arxiv.org/abs/2602.05832.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*, 2024. URL https://arxiv.org/abs/2410.24024.

Yifan Xu, Xiao Liu, Xinghan Liu, Jiaqi Fu, Hanchen Zhang, Bohao Jing, Shudan Zhang, Yuting Wang, Wenyi Zhao, and Yuxiao Dong. Mobilerl: Online agentic reinforcement learning for mobile gui agents, 2025. URL https://arxiv.org/abs/2509.18119.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713941. doi: 10.1145/3706598.3713600. URL https://doi.org/10.1145/3706598.3713600.

Lunjun Zhang, Ryan Chen, and Bradly C. Stadie. Evolutionary system prompt learning for reinforcement learning in llms. *arXiv preprint arXiv:2602.14697*, 2026. URL https://arxiv.org/abs/2602.14697.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, Biqing Qi, Youbang Sun, Zhiyuan Ma, Lifan Yuan, Ning Ding, and Bowen Zhou. TTRL: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025. URL https://arxiv.org/abs/2504.16084.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025. URL https://arxiv.org/abs/2506.10943.

APPENDIX

Overview. This appendix reports the complete data allocation, implementation details, workflow-context examples, additional analyses, prompt templates, and validation contracts for CoAdapt-GUI. Section numbering is continuous within the appendix and cross-references are resolved in the main document.

A ANDROIDWORLD PLUS CONSTRUCTION AND DATA ALLOCATION

A.1 BENCHMARK CONSTRUCTION AND EXECUTABLE FILTERING

AndroidWorld Plus is an experimental extension of AndroidWorld (Rawles et al., 2025). We retain the 19 AndroidWorld apps with app-attributable executable tasks and add six apps with their task suites: Calculator, Snapseed, and Wikipedia from B-MoCA (Lee et al., 2026), and Bluecoins, Maps.me, and Pi Music from AndroidLab (Xu et al., 2024). This produces 25 apps spanning 12 Google Play Store categories. We use the published Play Store category as an external functional taxonomy, following Android Control (Li et al., 2024), rather than defining categories from task outcomes.

The merged metadata contains 194 task rows. Two generic or composite tasks, `OpenApp` and `SaveCopyOfReceipt`, cannot be attributed to one app and are excluded from the app-level split, leaving 192 app-attributable templates. We then validate every template against the executable task registry. `WikipediaDecreaseTextSize50` is defined in the upstream task file but is not registered with the execution harness and therefore cannot be instantiated. We remove this template before freezing the manifests, yielding 191 executable task templates. No filtering decision uses an agent rollout, reward, or evaluation outcome.

A.2 SOURCE AND TARGET APP POOLS

The source and target app pools are disjoint. The source pool contains 12 apps and 96 templates (Table 3). Five task seeds per template, $\{30, 31, 32, 33, 34\}$, provide 480 source episodes for source-side policy training and workflow construction.

Table 3: Complete AndroidWorld Plus source-app allocation. All source templates are excluded from the target pool.

Source app	Category	Templates	Source app	Category	Templates
`markor`	Productivity	14	`tasks_org`	Productivity	6
`joplin`	Productivity	4	`calculator`	Tools	19
`clock`	Tools	3	`files`	Tools	2
`bluecoins`	Finance	15	`pi_music`	Music & Audio	12
`audio_recorder`	Music & Audio	2	`snapseed`	Photography	11
`simple_sms_messenger`	Communication	6	`contacts`	Communication	2
				Total	**96**

The target pool contains the remaining 13 apps and 95 templates. For the category-coverage analysis, a target app's group depends only on whether its category is represented in the source pool. Six target apps satisfy this condition and form the **Category-Shared Apps** group. The other seven belong to categories absent from the source pool and form the **Category-Novel Apps** group. Table 4 gives the complete target allocation.

A.3 TEMPLATE AND SEED SEPARATION

Within each target app, we sort the executable template identifiers and create disjoint adaptation and evaluation sets. For an app with at least three templates, the first $\lceil 0.6n \rceil$ templates are assigned to adaptation and the remainder to evaluation. Two-template apps use a one–one split; a

Table 4: Complete AndroidWorld Plus target-app allocation. The adaptation pool uses five seeds per adaptation template; frozen evaluation uses three disjoint seeds per evaluation template. `simple_draw_pro` has one executable template and is used only for adaptation, so the held-out panel covers 12 of the 13 target apps.

Group	Target app	Category	Templates		Episodes	
			Adapt.	Eval.	Adapt. pool	Eval.
Category-Shared	`simple_calendar_pro`	Productivity	11	6	55	18
Category-Shared	`system_settings`	Tools	9	6	45	18
Category-Shared	`pro_expense`	Finance	6	3	30	9
Category-Shared	`retro_music`	Music & Audio	3	1	15	3
Category-Shared	`camera`	Photography	1	1	5	3
Category-Shared	`chrome`	Communication	2	1	10	3
	Category-Shared subtotal		32	18	160	54
Category-Novel	`maps_me`	Maps & Navigation	9	6	45	18
Category-Novel	`osmand`	Maps & Navigation	2	1	10	3
Category-Novel	`broccoli`	Food & Drink	8	5	40	15
Category-Novel	`opentracks`	Health & Fitness	4	2	20	6
Category-Novel	`wikipedia`	Books & Reference	3	2	15	6
Category-Novel	`vlc`	Video Players & Editors	1	1	5	3
Category-Novel	`simple_draw_pro`	Art & Design	1	0	5	0
	Category-Novel subtotal		28	17	140	51
	Target total		**60**	**35**	**300**	**105**

single-template app contributes only to adaptation. This deterministic rule produces 60 adaptation templates and 35 evaluation templates, with no template overlap in any target app. The fixed lexicographic split was chosen for reproducibility; it is not stratified by task functionality, and robustness to alternative template partitions remains untested.

Each adaptation template is instantiated with seeds $\{30, 31, 32, 33, 34\}$, defining a 300-instance sampling pool. Each evaluation template is instantiated with the disjoint seeds $\{40, 41, 42\}$, producing the frozen 105-episode panel used for all reported AndroidWorld Plus results. The 300 adaptation instances specify the available sampling pool rather than the number of target rollout calls: CoAdapt-GUI samples from this pool under the fixed interaction schedule described in Appendix C.

Base Policy evaluation statistics. Across five evaluation runs, the Base Policy achieves 25, 25, 26, 27, and 24 successes out of 54 Category-Shared episodes, and 15, 14, 15, 14, and 14 out of 51 Category-Novel episodes. The corresponding overall counts are 40, 39, 41, 41, and 38 out of 105. The mean $\pm$ sample standard deviation of the per-run success rates is $(47.04 \pm 2.11)\%$ for Category-Shared, $(28.24 \pm 1.07)\%$ for Category-Novel, and $(37.90 \pm 1.24)\%$ overall. Each overall rate pools the successes across both groups before division by 105; it is not an unweighted average of the two group rates.

Static Context Transfer evaluation statistics. Across five evaluation runs, Static Context Transfer achieves 31, 31, 30, 30, and 33 successes out of 54 Category-Shared episodes, and 15, 16, 15, 14, and 14 out of 51 Category-Novel episodes. The overall counts are 46, 47, 45, 44, and 47 out of 105. The mean $\pm$ sample standard deviation of the per-run rates is $(57.41 \pm 2.27)\%$ for Category-Shared, $(29.02 \pm 1.64)\%$ for Category-Novel, and $(43.62 \pm 1.24)\%$ overall.

Policy-Only TTA evaluation statistics. Across five evaluation runs, Policy-Only TTA achieves 29, 29, 30, 26, and 31 successes out of 54 Category-Shared episodes, and 15, 13, 14, 13, and 12 out of 51 Category-Novel episodes. The overall counts are 44, 42, 44, 39, and 43 out of 105. The mean $\pm$ sample standard deviation of the per-run rates is $(53.70 \pm 3.46)\%$ for Category-Shared, $(26.27 \pm 2.24)\%$ for Category-Novel, and $(40.38 \pm 1.97)\%$ overall.

Context-Only TTA evaluation statistics. Across five evaluation runs, Context-Only TTA achieves 36, 33, 34, 34, and 33 successes out of 54 Category-Shared episodes, and 15, 16, 16, 14, and 17 out of 51 Category-Novel episodes. The overall counts are 51, 49, 50, 48, and 50 out of 105. The mean $\pm$ sample standard deviation of the per-run rates is $(62.96 \pm 2.27)\%$ for Category-Shared, $(30.59 \pm 2.24)\%$ for Category-Novel, and $(47.24 \pm 1.09)\%$ overall.

CoAdapt-GUI evaluation statistics. Across five evaluation runs, CoAdapt-GUI achieves 39, 38, 39, 38, and 36 successes out of 54 Category-Shared episodes, and 17, 16, 18, 18, and 17 out of 51 Category-Novel episodes. The overall counts are 56, 54, 57, 56, and 53 out of 105. The mean $\pm$ sample standard deviation of the per-run rates is $(70.37 \pm 2.27)\%$ for Category-Shared, $(33.73 \pm 1.64)\%$ for Category-Novel, and $(52.57 \pm 1.56)\%$ overall.

A.4 Complete source and target template manifests

Tables 5 and 6 list all 191 executable templates in the frozen AndroidWorld Plus manifest. Source templates are used only for source-side policy training and workflow construction. Target templates are partitioned into adaptation and held-out evaluation sets within each target app. Identifiers match the execution harness exactly.

Table 5: Complete AndroidWorld Plus source-pool template manifest

Source app	Category	Count	Executable source templates
markor	Productivity	14	MarkorAddNoteHeader MarkorChangeNoteContent MarkorCreateFolder MarkorCreateNote MarkorCreateNoteAndSms MarkorCreateNoteFromClipboard MarkorDeleteAllNotes MarkorDeleteNewestNote MarkorDeleteNote MarkorEditNote MarkorMergeNotes MarkorMoveNote MarkorTranscribeReceipt MarkorTranscribeVideo
tasks_org	Productivity	6	TasksCompletedTasksForDate TasksDueNextWeek TasksDueOnDate TasksHighPriorityTasks TasksHighPriorityTasksDueOnDate TasksIncompleteTasksOnDate
joplin	Productivity	4	NotesIsTodo NotesMeetingAttendeeCount NotesRecipeIngredientCount NotesTodoItemCount

Continued on the next page

Complete AndroidWorld Plus source-pool template manifest (continued)

Source app	Category	Count	Executable source templates
calculator	Tools	19	CalculatorConvert45DegreesToRadians CalculatorGeometricMean CalculatorHarmonicMean CalculatorInput1 CalculatorInput10Choose2 CalculatorInput17Times23 CalculatorInput1Plus1 CalculatorInput2Plus24Div3 CalculatorInput3Times5 CalculatorInput5Choose2 CalculatorInputCos180 CalculatorInputCos60 CalculatorInputFactorial6 CalculatorInputLn1234 CalculatorInputPercent50Of28 CalculatorInputSqrt25 CalculatorOpen CalculatorSumFirst5Fibonacci CalculatorSumFirst5Primes
clock	Tools	3	ClockStopWatchPausedVerify ClockStopWatchRunning ClockTimerEntry
files	Tools	2	FilesDeleteFile FilesMoveFile
bluecoins	Finance	15	BluecoinsAddExpense BluecoinsAddExpenseOnDate BluecoinsAddExpenseOnDateWithLabel BluecoinsAddIncomeOnDateWithNote BluecoinsAddIncomeWithLabel BluecoinsEditExpenseAmount BluecoinsEditExpenseDateAmountNote BluecoinsEditIncomeDateAndAmount BluecoinsEditTransactionType BluecoinsEditTransactionTypeAmountNote BluecoinsQueryCategorySpending BluecoinsQuerySpendingCategory BluecoinsQuerySpendingOnDate BluecoinsQueryTotalSpendingOnDate BluecoinsQueryTransactionCount
pi_music	Music & Audio	12	PiMusicCreatePlaylist PiMusicPauseAndSeek PiMusicPlayFromPlaylist PiMusicPlaySongByTitleArtist PiMusicQueryArtistSongCount PiMusicQueryArtistTotalDuration PiMusicQueryLongestSongDuration PiMusicQuerySongAlbum PiMusicQuerySortedSongsByTitle PiMusicQueryTotalSongs PiMusicSortByDurationAscending PiMusicSortByDurationDescending
audio_recorder	Music & Audio	2	AudioRecorderRecordAudio AudioRecorderRecordAudioWithFileName

Continued on the next page

Complete AndroidWorld Plus source-pool template manifest (continued)

Source app	Category	Count	Executable source templates
snapseed	Photography	11	SnapseedTask1 SnapseedTask10 SnapseedTask11 SnapseedTask2 SnapseedTask3 SnapseedTask4 SnapseedTask5 SnapseedTask6 SnapseedTask7 SnapseedTask8 SnapseedTask9
simple_sms_messenger	Communication	6	SimpleSmsReply SimpleSmsReplyMostRecent SimpleSmsResend SimpleSmsSend SimpleSmsSendClipboardContent SimpleSmsSendReceivedAddress
contacts	Communication	2	ContactsAddContact ContactsNewContactDraft

Table 6: Complete AndroidWorld Plus target adaptation/evaluation manifest

Adaptation templates	Held-out evaluation templates
simple_calendar_pro *Category-Shared; Productivity*	
11	**6**
SimpleCalendarAddOneEvent	SimpleCalendarEventsInTimeRange
SimpleCalendarAddOneEventInTwoWeeks	SimpleCalendarEventsOnDate
SimpleCalendarAddOneEventRelativeDay	SimpleCalendarFirstEventAfterStartTime
SimpleCalendarAddOneEventTomorrow	SimpleCalendarLocationOfEvent
SimpleCalendarAddRepeatingEvent	SimpleCalendarNextEvent
SimpleCalendarAnyEventsOnDate	SimpleCalendarNextMeetingWithPerson
SimpleCalendarDeleteEvents	
SimpleCalendarDeleteEventsOnRelativeDay	
SimpleCalendarDeleteOneEvent	
SimpleCalendarEventOnDateAtTime	
SimpleCalendarEventsInNextWeek	
system_settings *Category-Shared; Tools*	
9	**6**
SystemBluetoothTurnOff	SystemWifiTurnOff
SystemBluetoothTurnOffVerify	SystemWifiTurnOffVerify
SystemBluetoothTurnOn	SystemWifiTurnOn
SystemBluetoothTurnOnVerify	SystemWifiTurnOnVerify
SystemBrightnessMax	TurnOffWifiAndTurnOnBluetooth
SystemBrightnessMaxVerify	TurnOnWifiAndOpenApp
SystemBrightnessMin	
SystemBrightnessMinVerify	
SystemCopyToClipboard	
pro_expense *Category-Shared; Finance*	
6	**3**
ExpenseAddMultiple	ExpenseDeleteMultiple
ExpenseAddMultipleFromGallery	ExpenseDeleteMultiple2
ExpenseAddMultipleFromMarkor	ExpenseDeleteSingle
ExpenseAddSingle	
ExpenseDeleteDuplicates	
ExpenseDeleteDuplicates2	
retro_music *Category-Shared; Music & Audio*	
3	**1**
RetroCreatePlaylist	RetroSavePlaylist
RetroPlayingQueue	
RetroPlaylistDuration	

Continued on the next page

Complete AndroidWorld Plus target adaptation/evaluation manifest (continued)

Adaptation templates	Held-out evaluation templates
camera *Category-Shared; Photography*	
1	**1**
CameraTakePhoto	CameraTakeVideo
chrome *Category-Shared; Communication*	
2	**1**
BrowserDraw BrowserMaze	BrowserMultiply
maps_me *Category-Novel; Maps & Navigation*	
9	**6**
MapsMeAddWorkPlace MapsMeCheckDrivingDistanceTime MapsMeCheckNearestHotel MapsMeCheckNearestPlace MapsMeCheckNearestPlaceDriveTime MapsMeCheckNearestPlaceWalkTime MapsMeCheckPublicTransportRoute MapsMeCheckRidingTime MapsMeCheckWalkingDistanceTime	MapsMeCompareRidingVsPublicTransport MapsMeNavigateToBerkeley MapsMeNavigateToLocation MapsMeNavigateToOpenAI MapsMeNavigateToStanford MapsMeNavigateToUniversitySouth
osmand *Category-Novel; Maps & Navigation*	
2	**1**
OsmAndFavorite OsmAndMarker	OsmAndTrack
broccoli *Category-Novel; Food & Drink*	
8	**5**
RecipeAddMultipleRecipes RecipeAddMultipleRecipesFromImage RecipeAddMultipleRecipesFromMarkor RecipeAddMultipleRecipesFromMarkor2 RecipeAddSingleRecipe RecipeDeleteDuplicateRecipes RecipeDeleteDuplicateRecipes2 RecipeDeleteDuplicateRecipes3	RecipeDeleteMultipleRecipes RecipeDeleteMultipleRecipesWithConstraint RecipeDeleteMultipleRecipesWithNoise RecipeDeleteSingleRecipe RecipeDeleteSingleWithRecipeWithNoise
opentracks *Category-Novel; Health & Fitness*	
4	**2**
SportsTrackerActivitiesCountForWeek SportsTrackerActivitiesOnDate SportsTrackerActivityDuration SportsTrackerLongestDistanceActivity	SportsTrackerTotalDistanceForCategoryOverInterval SportsTrackerTotalDurationForCategoryThisWeek
wikipedia *Category-Novel; Books & Reference*	
3	**2**
WikipediaDisablePreviewAndFeed WikipediaGoToSavedTab WikipediaGoToSearchTab	WikipediaIncreaseTextSize180 WikipediaOpen
vlc *Category-Novel; Video Players & Editors*	
1	**1**
VlcCreatePlaylist	VlcCreateTwoPlaylists
simple_draw_pro *Category-Novel; Art & Design*	
1	**0**
SimpleDrawProCreateDrawing	*None*

B AndroidWorld-Generalization Protocol and Run Accounting

B.1 Released source and target splits

Setting 1 follows the released AndroidWorld-Generalization unseen-app split (Gu et al., 2026). Its source split contains 12 apps, 62 task templates, and 905 training instances. Every configuration starts from the released step-500 UI-TARS-7B checkpoint (Qin et al., 2025), which was trained only on this source split. The reusable source workflow library used by CoAdapt-GUI is likewise constructed only from this source split.

The target split contains five apps absent from the source split. It provides eight adaptation instances per app, for 40 in total, and 48 held-out evaluation instances. Adaptation and evaluation cover

the same 16 task templates but instantiate them with non-overlapping seeds. This setting therefore evaluates transfer to new instances of task types encountered during target adaptation, rather than transfer to new task templates.

Table 7: Complete target allocation for AndroidWorld-Generalization. Each template is present in both splits, but the concrete task instances are disjoint.

Target app	**Task templates (released task ID)**	**Adapt. inst.**	**Eval. inst.**
Audio Recorder	`AudioRecorderRecordAudioWithFileName` (28)	8	3
Clock	`ClockTimerEntry` (20)	8	3
OsmAnd	`OsmAndFavorite` (74); `OsmAndMarker` (88)	8	6
Tasks	`TasksDueOnDate` (100); `TasksHighPriorityTasksDueOnDate` (102); `TasksCompletedTasksForDate` (104); `TasksIncompleteTasksOnDate` (105)	8	12
Broccoli	`RecipeDeleteMultipleRecipes` (3); `RecipeDeleteSingleRecipe` (4); `RecipeAddSingleRecipe` (31); `RecipeDeleteSingleWithRecipeWithNoise` (32); `RecipeAddMultipleRecipes` (56); `RecipeDeleteMultipleRecipesWithNoise` (57); `RecipeAddMultipleRecipesFromMarkor2` (82); and `RecipeDeleteMultipleRecipesWithConstraint` (90)	8	24
Total	**16 templates**	**40**	**48**

B.2 Instance seeds and leakage control

The released evaluation manifest instantiates every target template with seeds $\{7, 30, 1234\}$. Adaptation uses different seeds: Audio Recorder and Clock use $\{1, 2, 3, 4, 5, 6, 8, 9\}$; each OsmAnd template uses $\{1, 2, 3, 4\}$; each Tasks template contributes two instances; and each Broccoli template contributes one instance. The released Tasks instances use seeds $\{1, 2\}$, except `TasksCompletedTasksForDate`, which uses $\{2, 3\}$. Seven Broccoli templates use seed 1 and `RecipeDeleteSingleWithRecipeWithNoise` uses seed 2. Consequently, the intersection between adaptation and evaluation $(\texttt{task_id}, \texttt{seed})$ pairs is empty.

All context revisions, policy updates, and checkpoint selection use only the 40-instance adaptation manifest. The workflow context and policy adapter are then frozen before the 48-instance evaluation manifest is opened. No held-out task instance, trajectory, reward, or outcome is used for adaptation or model selection.

B.3 Update schedule and result provenance

The adaptive configurations use the released eight adaptation instances per target app and a nominal 50-update schedule per app. They are evaluated on the same frozen 48-instance manifest. The schedule fixes the number of method-level updates; because context revision and policy optimization use different update operators, it should not be interpreted as an identical gradient or compute budget across methods.

We reproduce the Base Policy and Policy-Only TTA baselines from AndroidWorld-Generalization (Gu et al., 2026) in our evaluation pipeline. All five configurations are evaluated over five complete evaluation runs each, and the main paper reports the mean and standard deviation across runs. The Base Policy achieves 11, 13, 13, 14, and 14 successes out of 48 instances, giving $(27.08 \pm 2.55)\%$. Policy-Only TTA achieves 16, 18, 18, 20, and 17 successes out of 48, giving $(37.08 \pm 3.09)\%$. For both reproduced baselines, the uncertainty is the sample standard deviation across the five runs. All runs use the same frozen 48-instance evaluation manifest; the adaptive configurations also share the frozen adaptation manifest. No incomplete run is included in the reported statistics.

C CoAdapt-GUI Implementation Details

This section specifies the realization of the context and policy branches in Algorithm 1 of the main paper. It describes CoAdapt-GUI itself rather than restating the implementation of every comparison configuration. The notation `layer1` and `layer2` below refers to the runtime JSON schema: `layer1` is the app-bound screen-transition state, and `layer2` is the transferable workflow state described in the paper.

C.1 Policy and source initialization

For AndroidWorld-Generalization, CoAdapt-GUI starts from the released step-500 UI-TARS-7B checkpoint and uses the released 50-step target schedule described in Appendix B. For AndroidWorld Plus, the acting policy is Qwen3-VL-8B-Instruct at revision `0c351dd01ed87e9c1b53cbc748cba10e6187ff3b`. The VLM backbone is loaded in bfloat16 and remains frozen. We initialize it with a rank-16 LoRA adapter trained only on the 12 source apps. All configurations within a matched comparison use the same source-trained initialization, and configurations without policy adaptation keep it frozen. No target rollout, reward, or held-out outcome is used to construct or select this initialization.

Source workflow construction is independent of source-side LoRA training. A frozen synthesizer converts audited source trajectories into one app-bound screen-transition state and one transferable workflow state per source app. Only validated transferable entries are grouped by the externally defined Play Store category and consolidated into the immutable source library. App-bound screen states, transitions, UI strings, and identifiers never enter this library.

Table 8 reports the resulting frozen library. A target app can retrieve only the single library matching its Play Store category; cross-category retrieval is disabled. Thus a Category-Novel App receives no source entry, rather than an entry selected by semantic similarity or target reward.

Table 8: Composition of the frozen transferable source-workflow library used by AndroidWorld Plus. Counts refer to validated structured workflow entries, not source trajectories or task templates.

Source category	Source apps	Workflow entries
Productivity	3	14
Tools	3	15
Finance	1	7
Music & Audio	2	11
Photography	1	6
Communication	2	7
Total	**12**	**60**

C.2 AndroidWorld Plus adaptation configuration

Table 9 records the final configuration used for the AndroidWorld Plus experiments. The four rollouts associated with one task–context condition form the group size $N = 4$; N is not the number of context variants. At most $M = 2$ already materialized variants are selected in a round, so 20 rounds require at most $20 \times 2 \times 4 = 160$ target rollout calls per app. Early rounds may use only the root variant and therefore consume fewer calls.

In AndroidWorld Plus, Qwen3-VL-8B-Instruct collects the target-app rollouts, while frozen Claude Opus 4.7 supplies the source synthesizer and target reflector. Static Context Transfer and Policy-Only TTA also receive synthesizer-built source workflows for Category-Shared Apps, but perform no target-side reflection or workflow revision; neither receives source workflows for Category-Novel Apps. Context-Only TTA and CoAdapt-GUI use the same frozen reflector model, prompt templates, and generation parameters (Appendix F). These comparisons evaluate the full workflow-adaptation procedure, including external-model reasoning.

Table 9: CoAdapt-GUI implementation details for AndroidWorld Plus. The policy buffer is transient and is cleared after every update attempt; its lifetime is specified below.

Component	Parameter	Final value
Policy	Backbone / numerical precision	Qwen3-VL-8B-Instruct / bfloat16
Policy	Maximum image pixels / generated tokens	1,605,632 / 512
Policy	LoRA rank / scale / configured dropout	$16/32/0.0$
Policy	LoRA target modules	`q_proj`, `v_proj`
Policy	Optimizer / learning rate	AdamW / 3×10^{-4}
Policy	Maximum gradient norm	1.0
Policy	Advantage normalization	standardized within task–context group
Policy	Minimum active trajectories for an update	3
Policy	Frozen-policy anchor coefficient / log-ratio clip	$0.05/10.0$
Context	Root TrueSkill mean / standard deviation	$\mu_0 = 25, \sigma_0 = 25/3$
Context	TrueSkill performance / dynamics parameters	$\beta = 25/6, \tau = 1/12$
Context	TrueSkill draw probability	0.10
Context	Sampling window / child uncertainty increment	$15/1.5$
Context	Selection optimism / softmax temperature	$1.0/1.0$
Context	Frozen synthesizer and reflector	Claude Opus 4.7
Context	Reflection / diff output limits	$4{,}096/32{,}000$ tokens
Context	Malformed-output retries	3
Protocol	Adaptation rounds per target app	20
Protocol	Context variants per round	$M \leq 2$
Protocol	Rollouts per task–context condition	$N = 4$
Protocol	Maximum rollout calls per app	160
Protocol	Adaptation / evaluation task seeds	$\{30, 31, 32, 33, 34\}/\{40, 41, 42\}$
Protocol	Hardware / mean adaptation time	NVIDIA H200 141 GB / 9–10 GPU-hours

C.3 Context population and reward-guided revision

Each target app has an independent population of transferable workflow states. For a Category-Shared App, the root contains the retrieved category-matched source workflow context. For a Category-Novel App, source retrieval returns empty and the target-grounded root is initialized empty. In the latter case, the first structurally valid child constructs an initial transferable entry from target rollout evidence; later children refine that entry. The root starts with TrueSkill mean $\mu_0 = 25$, standard deviation $\sigma_0 = 25/3$, and evaluation count $e_0 = 0$.

Sampling and rating updates. At round t, the controller samples up to two variants without replacement from the latest 15-member population window. Sampling weights are

$$p(i) \propto \exp\left(\frac{\mu_i + \lambda\sigma_i}{T}\right), \qquad \lambda = 1, \quad T = 1.$$

Here (μ_i, σ_i) denotes the rating of variant i. Selected variants are evaluated on matched tasks and reset conditions. Their mean executable task rewards determine the ranks for the TrueSkill update, with performance parameter $\beta = 25/6$ and draw probability 0.10 for tied rewards. After this update, each participating variant receives the dynamics increment

$$\sigma_i \leftarrow \sqrt{\sigma_{i,\text{post}}^2 + \tau^2}, \qquad \tau = 1/12,$$

and its evaluation count e_i increases by one, including when its reward is zero. Unselected variants retain their ratings. A frozen reflector contrasts the resulting success and failure traces and proposes a typed revision to the parent with the largest mean reward in the current round. Reward ties are

resolved by larger $\mu - 3\sigma$, shorter rendered context, earlier birth iteration, and then variant identifier. This conservative score is used for parent tie handling, not for final deployment selection.

Child initialization and eligibility. Before updating the selected variants' ratings, the controller caches their pre-round values. A child of parent p is initialized as

$$\mu_{\text{child}} = \mu_{p,\text{pre}}, \qquad \sigma_{\text{child}} = \sigma_{p,\text{pre}} + 1.5, \qquad e_{\text{child}} = 0.$$

Thus the child inherits prior evidence about its parent, but not the rating increment from the rollouts that prompted its revision. Admission requires typed-diff parsing, transferable-state scope filtering, and a nonempty applied state change; the caller records parent, task, iteration, and revision provenance. These are structural checks, not evidence of improved task performance. Only after the child is sampled in a later round, evaluated, and included in a rating update does it become eligible for final selection. Positive reward is not an eligibility requirement.

Final selection. At budget exhaustion, the controller considers all historical variants with $e_i > 0$, including those outside the sampling window. It scans them in creation order, replacing the incumbent whenever a candidate has a strictly larger posterior mean μ. Otherwise, if the means are numerically close under `math.isclose` (relative tolerance 10^{-9}, absolute tolerance 10^{-12}), it prefers the lexicographically smaller pair of rendered-context length in characters and birth iteration. If that pair also ties, the incumbent is retained. The strict greater-mean check precedes the closeness check. If no variant has been evaluated, the root is returned. An unevaluated child, including one created in the final round, is otherwise ineligible; an evaluated zero-reward variant remains eligible. All-zero rewards use the same selection rule and do not trigger a separate root fallback.

Ratings under policy adaptation. Within a round, context evaluation, rating updates, and child admission precede any LoRA update. Ratings are retained across policy updates without resetting, decay, or policy-version separation, and there is no final-policy reevaluation or reranking. The dynamics increment above applies only to participants in a rating update; it is not a correction for policy changes. TrueSkill therefore guides online search using accumulated evidence. Matched rollouts support comparisons within a round, but do not calibrate ratings across changing policies or guarantee the best context for the final policy. Context-Only TTA keeps the policy fixed.

The reflection record contains the task template and instantiated goal, reset seed, executable task reward, and a compact trace of actions, action reasons, post-action summaries, UI evidence, and parse or execution status. The caller retains at most ten UI-element lines per step, clips each line to 160 characters, and clips reasons and summaries to 400 characters. These limits bound the reflector input without replacing executable rewards with model judgments.

C.4 Task–context-matched LoRA update

Every trajectory is stored with its task, rendered-context identity, reset identity, and the exact multimodal inputs and action tokens used during collection. Policy advantages are computed only within groups whose members share the same task and context variant. In the reported AndroidWorld Plus configuration, each such group contains four rollouts and uses standardized advantages:

$$\bar{r}_G = \frac{1}{|G|} \sum_{i \in G} r_i, \qquad A_j = \frac{r_j - \bar{r}_G}{s_G},$$

where s_G is the within-group reward standard deviation plus a small stabilizing constant. Singleton or constant-reward groups have zero advantage. Only trajectories with nonzero advantage enter the active policy batch, and an update is attempted only when at least three active trajectories remain.

For this Qwen implementation, the full update loss, including the frozen-policy anchor, is

$$\mathcal{L}_{\text{policy}} = \frac{1}{J} \sum_{j=1}^{J} \frac{1}{K_j} \sum_{k=1}^{K_j} \left[-A_j \log \pi_\theta(a_{jk} \mid x_{jk}) + \beta \, d_{\text{K3},jk} \right], \qquad \beta = 0.05,$$

where J is the number of active trajectories, K_j counts generated action tokens in trajectory j, and x_{jk} includes the multimodal input and preceding generated tokens. Averaging tokens within each

trajectory prevents long rollouts from dominating solely because they contain more action tokens. The sampled action-token K3 anchor is computed as

$$\begin{aligned}\delta_{jk} &= \log \pi_{\text{ref}}(a_{jk} \mid x_{jk}) - \log \pi_\theta(a_{jk} \mid x_{jk}),\\ \widetilde{\delta}_{jk} &= \text{clip}(\delta_{jk}, -10, 10),\\ d_{\text{K3},jk} &= \exp(\widetilde{\delta}_{jk}) - 1 - \widetilde{\delta}_{jk}.\end{aligned}$$

It is evaluated stably using `expm1` and is a numerically clipped K3-based KL estimate on sampled action tokens, rather than an exact full-vocabulary KL. In both source training and target adaptation, $\beta = 0.05$. The reference policy is the frozen base Qwen policy with LoRA disabled during source training, and the frozen shared source-trained LoRA policy during target adaptation.

The GRPO connection is the standardized group-relative advantage; the implementation directly optimizes $-A_j \log \pi_\theta$ in a single on-policy pass, without a behavior-policy probability ratio or a PPO clipped surrogate. At the collected behavior policy, this policy-gradient term has the same gradient as the corresponding ratio-based surrogate, with advantages held fixed and identical token and trajectory averaging; the scalar objectives differ. The ± 10 log-ratio clamp above stabilizes only the anchor and is separate from both PPO-style policy-ratio clipping and gradient-norm clipping, whose threshold is 1.0.

The policy buffer is a transient on-policy accumulation buffer, not a replay memory. The policy remains fixed while a matched batch is collected. After the batch becomes eligible, at most one LoRA update is attempted and the buffer is cleared whether or not the batch contains an active reward comparison. No trajectory is reused after the policy changes. Adapter dropout is disabled during both collection and gradient replay so that the saved behavior probabilities and the replayed current-policy probabilities refer to the same stochastic policy. Policy-sampling seeds, rather than dropout, provide the within-condition rollout diversity.

C.5 Frozen evaluation

After the interaction budget is exhausted, the selected workflow state and final LoRA adapter are written as immutable artifacts. Evaluation loads only these artifacts and the held-out manifest; the reflector, population controller, and optimizer are disabled. The structural, transferability, and behavioral checks applied before this point are specified together with the operative prompts in Appendix F.

D Workflow-Context Representation

This section gives concrete examples of the structured context described in the main paper. Each transferable workflow entry follows $w = \langle c, P, F, V \rangle$: an applicability condition, an abstract procedure, failure or recovery conditions, and completion checks. Each example below is condensed from a validated workflow artifact by removing repeated safeguards and provenance metadata; no new procedural, failure, or verification rule is introduced. App names are shown in the headings to identify the originating run, but are not part of the transferable text injected into the acting policy.

D.1 App-bound and transferable components

Table 10 illustrates the transfer boundary with a source artifact from `markor`. The app-bound component is a genuine screen-transition FSM: it includes concrete state names, visible strings, resource hints, and action-conditioned transitions. The transferable component retains the task logic while removing those interface bindings. Under unseen-app adaptation, only the right-hand form is eligible to initialize another app.

D.2 Representative highly rated target contexts

We next show three entries from highly rated validated states produced during AndroidWorld Plus adaptation. They cover both initialization regimes: the calendar example refines retrieved source knowledge for a Category-Shared App, whereas the navigation and activity-history examples were

Table 10: Concrete separation between app-bound interaction state and an app-agnostic workflow entry. The left column may support execution within its source app but is excluded from cross-application retrieval.

App-bound screen-transition state (not transferred)	**Transferable workflow entry (eligible)**

States: `file_browser`, identified by the Markor title, a red bottom-right create button, and the Files/To-Do/QuickNote/More navigation; `new_file_dialog`, identified by the pre-filled `my_note` field, format selector, and FOLDER/CANCEL/OK controls; and `editor_edit_mode`, identified by the editor toolbar and save icon.
Transitions:
State transition — **Action**
`file_browser` → `new_file_dialog` — `click(fab_plus)`
`new_file_dialog` → `editor_edit_mode` — `click(OK)`
	Applicability: a list or index exposes an affordance for creating an item.
	Procedure: invoke the create affordance; clear any placeholder; provide the required identifier; select the requested item type if needed; confirm creation; enter the requested content; and persist the change.
	Failures: retaining a placeholder in the identifier, choosing the wrong item type, creating in the wrong parent container, or declaring success before persistence.
	Verification: the new item appears under the expected parent with the exact identifier, and reopening it shows the requested content.

constructed from target rollouts after Category-Novel retrieval returned an empty source state. These entries are selected to illustrate the representation, not as additional quantitative evaluation examples.

Example 1: Scheduled-entry creation (Category-Shared)

Origin. The target app is `simple_calendar_pro`; its Productivity category is represented in the source pool. The entry starts from retrieved source workflow knowledge and is revised using target-app traces.

Applicability (c). A create form exposes multiple structured fields, such as title, date, start time, end time or duration, and optional content.

Procedure (P). Enumerate every goal-specified field before acting; resolve relative dates to absolute dates; set and visually confirm each field; after confirming the start time, derive the dependent end time; verify which picker component is active before editing it; save; then reopen the created item and read back every requested field.

Failure/recovery (F). Do not assume that an end-time field inherits an edited start time. Do not mistake an hour-to-minute auto-advance within one picker for a transition to another field. If the reasoning and action summary contain contradictory numeric values, re-inspect the form before continuing.

Verification (V). The saved detail view, rather than a potentially truncated list row, must show the requested date, start and end times, and other specified fields before completion is emitted.

Example 2: Nearest-place and route lookup (Category-Novel)

Origin. The target app is `maps_me`; Maps & Navigation is absent from the source pool. Its root transferable state is empty, and the entry is synthesized and subsequently refined from target adaptation traces.

Applicability (c). The task asks for a nearby place, place details, or a route with a specified travel mode.

Procedure (P). Establish a valid location and populated map view before searching; clear the query field and use a canonical place-category term; inspect both result rows and map markers; open a candidate's detail view; for routing, set both endpoints and the requested travel mode before reading the rendered distance or duration.

Failure/recovery (F). A viewport-scoped empty result is not global infeasibility. Recenter or widen the map before reformulating the query. Clear existing text before retrying to avoid concatenated queries, and avoid random panning without a location reference.

Verification (V). A location reference and populated map are visible; the selected result's type matches the goal; and any reported place, distance, or duration is read from the rendered result or route summary.

Example 3: Historical activity aggregation (Category-Novel)

Origin. The target app is `opentracks`; Health & Fitness is absent from the source pool. The selected workflow descends from a child that constructed the category entry from an empty root.

Applicability (c). A history query specifies a time window, an activity type, and an aggregation such as count, total, maximum, or average.

Procedure (P). Anchor relative dates to an observed current date; resolve the requested time interval and week convention; use available date and activity filters; inspect the full in-window set; disambiguate activity types using structured metadata rather than user-authored titles; aggregate the confirmed entries; and format the result in the requested unit and precision.

Failure/recovery (F). Avoid first-match answers for maximum or total queries, title-based activity classification, confusion between calendar weeks and rolling seven-day windows, and unit conversion based on rounded summary values when a precise detail value is available.

Verification (V). Every included entry has an observed type and timestamp inside the requested interval; the list boundary has been reached; units and time formats have been checked; and exactly one answer is followed by one completion action.

D.3 A RECORDED CONTEXT-EVOLUTION STEP

The Chrome adaptation log provides a compact example of reward-guided context selection. The retrieved Communication context initially contained generic navigation and repeated-action procedures. A child proposed after target traces added three pieces of guidance: handle first-launch gates before continuing the task, verify that a post-action observation has actually advanced, and change strategy after repeated no-op actions. Table 11 records child admission followed by direct evaluation in later rounds.

Table 11: A recorded Chrome context-evolution event. Values are mean executable task rewards over the four rollouts for the indicated task–context condition.

Round	Observed reward	Context event
6	root: 0.00	`gen1_mut_1` is proposed, validated, and admitted without an assigned reward.
7–11	all: 0.00	The child and existing alternatives are tested, but every evaluated condition remains at zero.
12	child: 0.25; root: 0.00	Under the matched `BrowserMaze` condition, the evolved context exposes nonzero executable feedback absent under the root.

The round-12 rating update favored the evolved branch, and later revisions retained its onboarding and state-advancement checks. The trace illustrates the separation between structural admission and subsequent rollout evidence used to assess a revision. Zero rewards in earlier evaluations do not prevent further sampling.

E ADDITIONAL ANALYSES AND ABLATIONS

E.1 INCREMENTAL CONTRIBUTIONS OF THE ADAPTATION CHANNELS

Table 12 reports success-rate differences between the AndroidWorld Plus configurations in the main paper. Static Context Transfer is 10.4 percentage points above the Base Policy on Category-Shared Apps. On Category-Novel Apps, both configurations use the fixed initial policy without workflow context, so their tabulated 0.8-point difference should not be interpreted as a source-transfer

effect. With source context held fixed, Policy-Only TTA is 3.2 percentage points below Static Context Transfer overall. In contrast, target-side context construction improves both groups over Static Context Transfer, and enabling the complete joint procedure adds a further 5.3 points overall over Context-Only TTA.

Table 12: AndroidWorld Plus success-rate differences in percentage points. Static Context Transfer is the reference for policy-only and context-only adaptation; the final row compares joint adaptation with Context-Only TTA. Differences are computed from unrounded mean success rates.

Increment	**Category-Shared**	**Category-Novel**	**Overall**
Static Context Transfer − Base Policy	+10.4	+0.8	+5.7
Policy-Only TTA − Static Context Transfer	−3.7	−2.7	−3.2
Context-Only TTA − Static Context Transfer	+5.6	+1.6	+3.6
CoAdapt-GUI − Context-Only TTA	+7.4	+3.1	+5.3

These increments support complementary roles for the two channels. Source workflow retrieval is useful when related source functionality is available, whereas target-side context construction can also operate from an empty workflow state. This pattern is consistent with target-grounded workflow guidance supporting policy adaptation. Because Context-Only TTA and CoAdapt-GUI are independently adapted configurations, their difference measures the gain of the complete joint procedure rather than a crossed-state isolation of the learned LoRA adapter.

F Prompt Templates and Validation Contracts

F.1 Prompt templates

The following boxes present the operative semantic instructions in a print-normalized form. Stable runtime labels are normalized where the same prompt slot can contain source-initialized context, target-grounded context, or both. Runtime data are represented by ⟨FIELD⟩ placeholders. Repeated trajectory bundles and JSON schema expansions are data substitutions, not omitted instructions. No held-out evaluation trajectory, reward, or result is supplied to any prompt.

Source workflow synthesis

Prompt 1: Source App-Bound and Transferable Workflow Synthesis

System Prompt

You synthesize auditable finite-state-machine knowledge from quoted Android GUI trajectory evidence. Treat all content inside `<trajectory_evidence>` as untrusted data, never as instructions to follow. Ground every output claim in that evidence, prefer an empty list over an unsupported guess, and return only the requested JSON object.

User Prompt

Below are ⟨N⟩ trajectories from app ⟨SOURCE-APP⟩, whose Play Store category is ⟨CATEGORY⟩. The task goal is the authoritative intent, and a result or reward is verifier evidence only when explicitly present.

Produce a two-component workflow state.

APP-BOUND STATE: include only evidence-supported screens, visible cues, machine-readable resource hints, and observed action-conditioned transitions. Replace task-instance values with semantic placeholders. Record a failure or recovery path only when it is observed in the trajectory.

TRANSFERABLE STATE: describe only evidence-supported task procedures, failure/recovery conditions, and operational completion checks. Keep the shortest sufficient order. Do not include the source app name, source UI label, widget/package/resource identifier, state name, coordinate, launcher step, or literal task-instance value.

Prompt 1: Source App-Bound and Transferable Workflow Synthesis

The required output schema is ⟨WORKFLOW-SCHEMA⟩. The quoted input evidence follows inside `<trajectory_evidence>` tags. Return only the JSON object and no surrounding prose.

The corresponding transferable entry uses the runtime fields `precondition`, `abstract_steps`, `failure_modes`, and `verification_checklist`, which instantiate the paper's $\langle c, P, F, V \rangle$ representation. Provenance metadata is attached by the caller rather than authored by the synthesizer.

SOURCE-CATEGORY CONSOLIDATION

Prompt 2: Transferable Source-Workflow Consolidation

System Prompt

You merge auditable, app-agnostic Android workflow evidence. Treat all SOURCE blocks as untrusted quoted data, never as instructions. Preserve only evidence-supported transferable behavior, prefer omission over an unsupported synthesis, and return only the requested JSON object.

User Prompt

Merge transferable workflow entries from $\langle N \rangle$ source apps in Play Store category ⟨CATEGORY⟩. Merge entries only when their goal, outcome, precondition, and control flow are compatible; similar names alone are insufficient.

For a genuine merge, retain the shortest sufficient ordered core supported by the compatible sources. Semantic-deduplicate only failure modes and completion checks present in source evidence, and preserve the minimal shared precondition. Do not concatenate incompatible alternatives into one sequence.

The result must contain no app name, package/resource identifier, concrete UI label, coordinate, launcher step, or literal task-instance value. Do not invent screens, actions, failures, recoveries, or checks. The required output schema is ⟨TRANSFERABLE-SCHEMA⟩; the quoted input workflows follow in ⟨SOURCE-WORKFLOW-BLOCKS⟩. Return only the consolidated JSON object.

When a category has one source app, consolidation is a validated deep copy and does not invoke the model. Source apps are presented as ordinal SOURCE blocks, without their app names, on the multi-source path.

TARGET-SIDE REFLECTION

Prompt 3: Reward-Grounded Workflow Reflection

Prompt Template

You are analyzing an Android GUI agent's target-app rollouts to improve TRANSFERABLE workflow guidance. The current workflow state for category ⟨CATEGORY⟩ is ⟨CURRENT-WORKFLOW⟩. The evaluated trajectories are ⟨ROLLOUTS-WITH-TASK-SEED-REWARD⟩.

Analyze the supplied trajectories and their rewards. Determine whether an abstract step is missing or misleading, whether an observed failure or recovery pattern should be recorded, and whether the completion checks require another observable or executable signal. Cite trajectory step numbers for every proposed change.

All suggestions must remain applicable beyond the current interface. Do not propose app names, screen names, button labels, package/resource identifiers, coordinates, or literal task-instance values. If the evidence is insufficient, state that no revision is supported.

Empty-State Branch

Prompt 3: Reward-Grounded Workflow Reflection

If ⟨CURRENT-WORKFLOW⟩ is empty, construct an initial app-agnostic workflow entry from the target evidence instead of editing a source entry. Use only patterns supported by the supplied target rollouts; an empty proposal is preferable to an unsupported rule.

TYPED WORKFLOW REVISION

Prompt 4: Typed Transferable-Workflow Diff

Prompt Template

Given the current transferable workflow state and the grounded reflection, return a JSON diff. Every operation must target `layer2/category`; no app-bound operation is allowed. The only operation types are `add`, `modify`, and `remove`. Include only evidence-supported changes. For a modified list, return the complete updated list rather than only its new items. Prefer one to three focused operations and consolidate near duplicates.

Keep at most 16 abstract steps, 20 failure modes, and 16 verification items in an entry. No app name, package/resource identifier, concrete widget label, coordinate, state identifier, or literal task-instance value may occur. Return only the JSON object.

Output Schema

```
{"ops": [{"layer": "layer2", "target": "category",
"op": "add|modify|remove", "key": <category>,
"value": <complete typed value>}],
"reflection_summary": <string>, "layer_tag": "layer2"}
```

For an empty target-grounded state, the first valid diff uses an `add/category` operation containing the initial precondition, abstract steps, failure modes, and verification checklist. It enters the population without reward and is tested only in later rounds.

AGENT-FACING WORKFLOW INJECTION

Prompt 5: Agent-Facing Workflow Guidance

Injected Prompt Slot

Workflow knowledge

The following abstract workflow patterns encode general strategies, common failure modes, and verification signals. They may come from eligible source workflows, validated target-adaptation experience, or both. They do not describe this specific app's UI; rely on the current screenshot to identify actual elements.

⟨RENDERED-TRANSFERABLE-WORKFLOW-STATE⟩

The slot is inserted after the stable agent instruction prefix and before the episode-specific goal, history, screenshot, and UI-element list. The rendered state contains the retrieved source entries together with the currently selected target-grounded revisions. For a Category-Novel App, the initial rendered state contains no abstract categories; guidance appears after an empty-state candidate has been constructed, admitted, and selected in a later round.

F.2 VALIDATION CONTRACTS

The prompts above state semantic evidence and transfer constraints, while the runtime applies separate programmatic checks. Table 13 distinguishes these two roles. This distinction is important: instructions against concrete UI labels, coordinates, launcher behavior, and literal task-instance values are semantic prompt constraints; the hard transfer linter detects identifiable source-app strings,

package or resource identifiers, and app-specific state identifiers. Structural validity and later task reward are checked independently.

Table 13: Prompt-level and programmatic validation contracts. Structural admission determines whether a candidate can be evaluated; executable rewards from later matched rollouts determine whether it is behaviorally useful.

Stage	Contract	Failure handling
Source synthesis	The response must parse as a JSON object containing the typed app-bound and transferable workflow fields. Model-authored metadata are discarded; the caller records the evidence count, synthesis model and settings, and a hash of the complete prompt.	A malformed response or a missing required component is rejected rather than converted into a source workflow artifact.
Source consolidation	Entries are merged only within an externally defined app category. The result must contain at least one typed transferable entry. The hard source-library linter checks the union of contributing source apps for app-name variants, package/resource identifiers, and app-specific state identifiers.	An empty, malformed, or lint-failing library is not written and therefore cannot be retrieved for a target app.
Target reflection	Only adaptation-task goals, reset identities, compact traces, and executable task rewards are supplied. The prompt requires every suggested change to cite adaptation-trace evidence and remain app-agnostic.	Reflection text cannot directly modify the workflow state; it is used only as evidence for the subsequent typed-diff request.
Typed revision	The response must parse into typed `add`, `modify`, or `remove` operations. In target adaptation, only `layer2/category` operations are retained, and the applied diff must change the parent state.	Malformed JSON is retried up to three times. Empty diffs, all-skipped operations, app-bound operations, and unchanged children are rejected.
Candidate admission	The caller records the parent, task, iteration, and revision summary. A valid child inherits its parent's pre-round rating with a standard-deviation increment of 1.5 and starts with evaluation count zero.	An unevaluated child cannot be selected for deployment. A later rollout-based rating update makes it eligible even if its reward is zero; final selection follows Appendix C.3.
Frozen evaluation	The selected context and LoRA adapter are loaded as immutable artifacts, and only the held-out manifest is opened.	The reflector, population controller, and optimizer are disabled; evaluation trajectories and rewards cannot alter either adapted state.

The validation pipeline therefore does not treat a syntactically valid model response as learned knowledge. Source entries must satisfy construction and transfer checks, and target revisions must first become typed candidate states. A target revision becomes eligible for final selection after a subsequent rollout-based rating update, not merely after structural admission. Eligibility does not require positive reward or improvement over its parent; final selection and the no-evaluated-candidate root fallback follow Appendix C.3.