HATS: Hardness-Aware Trajectory Synthesis for GUI Agents

Supplementary Material

A. Extended Related Works

Language-Driven GUI Automation. Large language models (LLMs) have become the engine behind a new wave of language-conditioned agents that perceive, reason, and act within graphical user interfaces [11, 12, 65, 79]. Early efforts retained frozen parameters and relied on prompt programming, cooperative agent collectives [21, 57, 64], external tool execution [58], self-reflection loops [6, 55, 68], and simulated world models [18, 22, 72]. A parallel line of work adapts model weights to acquire interface-specific skills: recognizing visual screen states [5, 16, 66] or hierarchical UI trees [67, 75], emitting atomic actions such as click and type [3], and generalizing across web [9, 69], desktop [23, 41], and mobile domains [27, 60]. Related trends are also visible beyond GUI automation, where multimodal grounding, history-aware policy optimization, and sequential decision-making have been studied in media understanding and embodied agents [28–32, 51–53, 80]. Collectively, these studies lay the groundwork for agents capable of non-trivial task execution in heterogeneous digital ecosystems.

Benchmark Suites and GUI Corpora. Robust agent training hinges on rich, varied interface data capturing both static layouts and dynamic behaviours [43, 65, 71]. The RICO dataset introduced sequential screenshots and view hierarchies for mobile applications [8], whereas MINIWOB presented fine-grained web interactions [54]. Subsequent initiatives extended coverage: large-scale mobile GUI logs [2, 36, 47, 74], expansive web task collections [34, 37, 40], and curated desktop traces for instructional tasks [3]. Related data-centric efforts in embodied and structured interaction learning have likewise emphasized scalable action datasets and temporal consistency [19, 38]. These corpora supply the visual and structural context needed to ground language-conditioned policies.

Trajectory-Centric Supervision. End-to-end agent learning benefits from trajectory data composed of instructions, subgoals, actions, and intermediate states [27, 73, 76]. Two major paradigms exist: (i) **Task-driven** schemes recruit annotators or scripted bots to complete predefined tasks, producing high-fidelity but narrow-coverage demonstrations [27, 37]; (ii) **Exploration-driven** pipelines allow autonomous agents to traverse environments and discover tasks through random walks [42, 59]. Recent embodied-agent studies similarly rely on trajectory abstraction, hierarchical refinement, memory augmentation, and skill composition to improve long-horizon action quality [26, 28–32, 80]. While cost-efficient, naive exploration gathers large numbers of semantic-intuitive actions while failing to capture rarer but more valuable **semantic-ambiguous actions**. Consequently, existing corpora remain biased toward trivial behaviours and often contain incoherent instructions or execution errors, limiting generalization. Alleviating this imbalance and improving semantic alignment directly motivates the present work.

Hardness-Aware Learning. The concept of hardness has been long studied in machine learning. Curriculum and self-paced learning [1, 24] organize training from easy to hard samples, while automated curricula [17] and hardness-aware exploration [13, 45] dynamically adjust sample hardness. In computer vision, hard-example mining techniques prioritize rare or ambiguous samples that improve model generalization [56], and related robustness research further highlights the need to focus on adversarially sensitive, hard-to-evaluate, or domain-shifted cases [14, 15, 49, 50]. Yet existing GUI trajectory synthesis pipelines seldom incorporate hardness-aware mechanisms [42, 59]. Our method addresses this gap by transforming instruction–execution misalignment into a hardness signal to identify and prioritize under-represented, high-value semantic-ambiguous actions.

B. Experimental Details

B.1. Evaluation Benchmarks

We evaluate our method on two representative GUI benchmarks: ANDROIDWORLD [48] (mobile) and WEBARENA [78] (web).

B.1.1. Environment Setup

- ANDROIDWORLD [48]. This mobile-focused benchmark is executed in Android emulators and contains diverse user-oriented tasks such as app navigation, form filling, and in-app purchases, capturing realistic GUI layouts and interaction dependencies. We conduct all mobile experiments using Android Virtual Devices (AVD) configured with Pixel 6 hardware profiles and the Tiramisu system image (API Level 33), following the official setup guidelines.
- WEBARENA [78]. This large-scale web interaction benchmark consists of functional websites, including email clients, calendars, and e-commerce platforms, presenting challenges due to dynamic page layouts and varied interaction semantics. For web-based evaluation, we employ the AgentLab framework [7, 10] to standardize experiment execution and management, evaluating on three representative domains: **Gitlab** (180 tasks), **Maps** (109 tasks), and **Reddit** (106 tasks), which together cover diverse patterns of web interaction including repository management, location-based services, and social media

Algorithm 1 HD-MCTS: Hardness-Driven Exploration with Alignment-Guided Refinement.

Require: Environment $\mathcal{E}$; depth limit $T_{\max}$; UCB constant C; thresholds $R_{\min}, F_{\max}$; hardness parameters ϵ, α

for each iteration ▷ Each iteration generates one training sample **do**
  Retrieve root node $v_0 \leftarrow \textsc{GetOrCreateNode}(a_0, s_0)$; $\mathcal{P} \leftarrow \varnothing$
  $v \leftarrow v_0$; $s \leftarrow s_0$
  ▷ **Selection**: traverse by UCB until a frontier or depth limit (measured by $|\mathcal{P}|$)
  while v is fully expanded **and** $|\mathcal{P}| < T_{\max}$ **do**
    $a^\star \leftarrow \arg\max_{a \in \mathcal{A}(v)} \left[Q(v,a) + C\sqrt{\frac{\ln(N(v)+1)}{N(v,a)+1}} \right]$
    $s' \leftarrow \mathcal{E}(s, a^\star)$
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(a^\star, s')\}$; $v \leftarrow \textsc{GetChildNode}(a^\star, s')$; $s \leftarrow s'$
  end while
  ▷ **Expansion**: add a previously unexpanded action if $|\mathcal{P}| < T_{\max}$
  if v **is not** fully expanded **and** $|\mathcal{P}| < T_{\max}$ **then**
    choose previously unexpanded action $a_e \in \mathcal{A}(v)$; $s_e \leftarrow \mathcal{E}(s, a_e)$
    $u \leftarrow \textsc{CreateNode}(a_e, s_e)$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{(a_e, s_e)\}$; $v \leftarrow u$; $s \leftarrow s_e$
  end if
  ▷ **Simulation**: rollout → selection → instruction synthesis → replay → refine → reward computation
  $\bar{\mathcal{P}} \leftarrow \textsc{Rollout}(\mathcal{E}, v, s, \mathcal{P}, T_{\max})$ ▷ Extend path $\mathcal{P}$ from current node until reaching $T_{\max}$
  $A \leftarrow \textsc{SelectSubsequence}(\bar{\mathcal{P}})$ ▷ Select a semantically coherent sub-trajectory from the path
  $I \leftarrow \textsc{SynthesizeInstruction}(A)$ ▷ Instruction synthesis from A
  $\mathbb{A} \leftarrow$ **false**; $F \leftarrow 0$ ▷ $\mathbb{A}$: alignment flag
  repeat
    $B \leftarrow \textsc{ExecuteInstruction}(I)$ ▷ Replay I from the original start state of A
    $R \leftarrow \textsc{AlignmentScore}(A, B)$ ▷ Action-level alignment, defined by Eq. 2
    if $R \geq R_{\min}$ **and** $\textsc{Executable}(B)$ **then**
      $\mathbb{A} \leftarrow$ **true**
    else
      $I \leftarrow \textsc{RefineInstruction}(I, A, B)$ ▷ Diagnose misalignment and refine
      $F \leftarrow F + 1$
    end if
  until $\mathbb{A}$ **or** $F \geq F_{\max}$
  ▷ **Hardness reward**: convert alignment into exploration signal
  $r \leftarrow (R + \epsilon)^{-\alpha}$ ▷ Eq. 3; lower alignment ⇒ higher hardness
  ▷ **Emission**: keep only aligned pairs for training
  if $\mathbb{A}$ **then**
    $\textsc{StoreSample}(I, B, r)$ ▷ Emit verified instruction–trajectory with hardness score
  end if
  ▷ **Backpropagation**: update edge values along the selected path $\mathcal{P}$
  for each $(a_i, s_i) \in \mathcal{P}$ **do**
    $N(a_i) \leftarrow N(a_i)+1$
    $Q(a_i) \leftarrow Q(a_i) + \frac{r - Q(a_i)}{N(a_i)}$
  end for
end for

platforms.

B.1.2. Task Distribution and Categorization

AndroidWorld Task Categories. We organized AndroidWorld's 116 evaluation tasks into four functional categories based on the primary application domain. Table 6 provides the complete mapping from applications to categories. The four categories are defined as:

- **DLS (Daily Life & Services):** entertainment, media, and lifestyle apps
- **S&C (Social & Communication):** messaging and social interaction apps
- **S&U (System & Utility):** core system functions and

utilities

- **P&W (Productivity & Work):** task management and productivity tools.

This categorization allows us to analyze performance across different levels of interaction complexity and semantic domains. The training corpus comprises **1,000 trajectories** collected through our synthesis pipeline, while the evaluation uses the full set of **116 benchmark tasks** from ANDROIDWORLD.

Table 6. **Application categorization for AndroidWorld.**

Application	Category	Application	Category
Calculator	P&W	VLC	DLS
Calendar	P&W	Markor	P&W
Clock	S&U	RetroMusic	DLS
Contacts	S&U	Gallery	DLS
Dialer	S&U	Tasks	P&W
Settings	S&U	SMSMessenger	S&C
Chrome	P&W	Expense	DLS
Files	P&W	Joplin	P&W
Google Tasks	P&W	Camera	S&U
Keep Notes	P&W	Broccoli	DLS
Draw	P&W	Audio Recorder	P&W
Launcher	S&U	OsmAnd	S&C
Markup	P&W	OpenTracks	P&W

B.1.3. Evaluation Protocol

Success Criteria. We adopt the official evaluation protocol from both benchmarks. Success is determined by **exact state matching**: an agent completes a task successfully if and only if the final environment state exactly matches the ground-truth target state specified by the benchmark. Partial completion or near-miss outcomes are counted as failures.

Execution Constraints. Each task is subject to the maximum step limit defined by the respective benchmark. For AndroidWorld, we enforce per-task step budgets as specified in the original evaluation suite. For WebArena, we similarly respect the official constraints for each domain.

Retry Policy. We retry executions only in cases of **environment-level failures** (e.g., AVD unresponsiveness, API timeouts, network errors). Task failures due to agent limitations (incorrect actions, semantic misunderstandings) are **not** retried and are recorded as failures in the final evaluation.

B.2. Model Configuration and Training

For instruction synthesis and reward modeling, we employ **GPT-4o** [20]. Two VLM backbones are used for agent training:

- **InternVL2-4B/8B** [4]: general-purpose models without GUI-specific pretraining.
- **Qwen2-VL-7B-Instruct** [61]: an instruction-tuned model with agentic reasoning capabilities.

B.2.1. Training Hyperparameters

Table 7 summarizes the training configuration for each VLM backbone. All models are trained for **15 epochs** with a global batch size of **128** (achieved via gradient accumulation) on **8×NVIDIA H100 80GB GPUs**.

Table 7. **Training hyperparameters for VLM backbones.**

Model	Learning Rate	Batch Size	Epochs
InternVL2-4B	4e-5	128	15
InternVL2-8B	4e-5	128	15
Qwen2-VL-7B	2e-6	128	15

Optimization and Regularization. We use the **AdamW** optimizer [35] with a weight decay of **0.05** for all models. For Qwen2-VL-7B, we additionally apply gradient clipping with a maximum gradient norm of **1.0** to stabilize training. All models employ a **cosine annealing** learning rate schedule with a linear warmup phase covering **3%** of total training steps (warmup ratio = 0.03).

Distributed Training Infrastructure. Training employs the **DeepSpeed** [46] framework for efficient distributed optimization. The effective batch size of 128 is achieved with a per-device batch size of **2** and **8** gradient accumulation steps. We use **bfloat16** (bf16) mixed precision to enable full fine-tuning of all model parameters while maintaining stable gradient updates and memory efficiency.

Data Preprocessing. To accelerate training and inference, all input screenshots are resized to a resolution of **460×1024** pixels while preserving aspect ratio. This resolution balances visual detail retention with computational efficiency.

Inference Configuration. During evaluation, we use **greedy decoding** (temperature = 0) to ensure deterministic and reproducible results. For AndroidWorld, we impose no explicit length constraints on model generation. For WebArena, due to longer observation sequences from complex web pages, we set `max_model_len=16384` to prevent out-of-memory errors while accommodating typical task contexts. For all experiments, we report results using the **final checkpoint** saved at the end of training (epoch 15).

B.3. Baselines

We compare **HATS** against four representative data-synthesis paradigms, all using consistent multimodal inputs (**a11ytree** and screenshots):

- **Zero-Shot (CoT + M3A)** [48]: a GPT-based agent guided by Chain-of-Thought prompting [63], without additional training.
- **Task-Driven Generation** [25]: the model receives predefined tasks and screenshots to synthesize trajectories.
- **Self-Instruct** [62]: automatically generates new task instructions to expand trajectory diversity.

- **OS-Genesis** [59]: a reverse task-synthesis pipeline that explores environments first and generates instructions post hoc, with instruction refinement and reward filtering.

Baseline Reproduction Details. To ensure rigorous comparison, we adopted specific reproduction protocols for different benchmarks. On AndroidWorld, we evaluated OS-Genesis using their officially released weights and evaluation code on our identical AVD setup, while the Task-Driven and Self-Instruct pipelines were reproduced by ourselves following the methodologies described in [59]. On WebArena, consistent with AndroidWorld, we reproduced the Task-Driven and Self-Instruct baselines based on the descriptions in [59]. However, the official evaluation code for OS-Genesis is not publicly available. Our attempts to reproduce its results yielded significantly lower performance than reported in the original paper. This discrepancy likely stems from environmental differences—such as updates to the underlying LLM API (e.g., gpt-4o) and subtle variations in browser state simulation—that are difficult to align without a standardized evaluation framework. To avoid concerns about unfairly underestimating OS-Genesis, we report its original paper results in all main comparisons. Notably, our method still outperforms these baseline scores by a large margin, demonstrating that the observed gains are not affected by minor variances in reproduction.

B.4. Trajectory-Based Training

All methods use **Supervised Fine-Tuning** (SFT) over synthesized trajectories with two complementary objectives:

$$\mathcal{L}_1 = -\sum_{t_i \in \mathcal{T}} \log \left(p_\theta(\ell \mid s, h_i, c) \cdot p_\theta(a \mid s, h_i, c, \ell) \right)$$

$$\mathcal{L}_2 = -\sum_{t_i \in \mathcal{T}} \log p_\theta(a \mid s, c, \ell)$$

where s is the multimodal state, h_i the task instruction, c the context, ℓ the subgoal, and a the predicted action. We use 1K trajectories for all methods except Self-Instruct (1.5K), with an average of 6.4 interaction steps per trajectory. ReAct-style reasoning [70] is adopted for interpretability.

B.5. Experimental Setup for Component Analysis

To balance computational efficiency with representativeness, component analyses were conducted on a curated subset of AndroidWorld comprising 20 tasks. These tasks span diverse functional domains—such as social communication, productivity, and utilities—and vary in interaction depth (3–12 steps), ensuring broad coverage of real-world agent behaviors. For each analysis, we employed identical rollout trajectories and model backbones (InternVL2-4B) to isolate the effect of each proposed component. For instance, in Analysis I (Hardness Metric Validation), we computed hardness values across multiple (ϵ, α) configurations using the same trajectories, comparing both recall- and precision-based formulations under consistent conditions.

C. Prompt Summary

We summarizes all prompt templates used in the paper:

- **Instruction Generation Prompt.** For converting trajectories into natural-language instructions, we use a prompt that enforces step grounding, UI-action consistency, and minimal hallucination. This template ensures that generated instructions remain aligned with the underlying hard actions and faithfully reflect the agent's behavior.
- **Alignment Verification Prompt.** To evaluate whether a synthesized instruction is semantically compatible with a rollout, we employ an alignment-checking prompt that compares action semantics, reasoning flow, and intended outcomes. This template is used in the alignment-aware refinement stage to filter out misaligned trajectories.
- **Semantic-Ambiguity Detection Prompt.** To identify context-, sequential-, and visually driven hard actions, we employ a structured ambiguity-classification prompt. This prompt guides the model to analyze trajectories and determine whether they contain any of the three types of semantic ambiguity defined in the main paper.

Together, these prompts define the interface between the agent, the trajectory, and the verifier. For completeness and reproducibility, the full prompt templates are provided in the following pages.

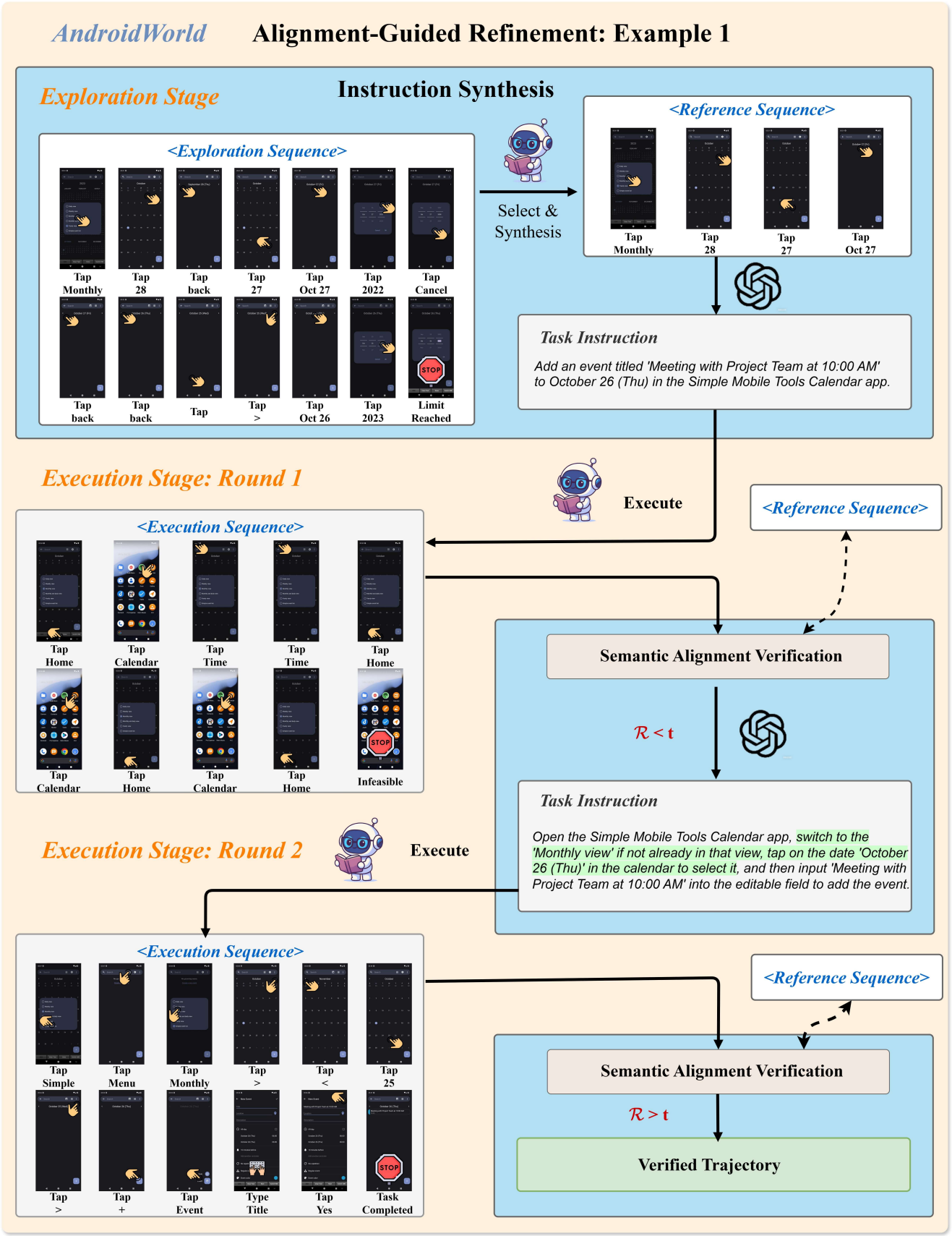
D. Instruction Refinement Examples

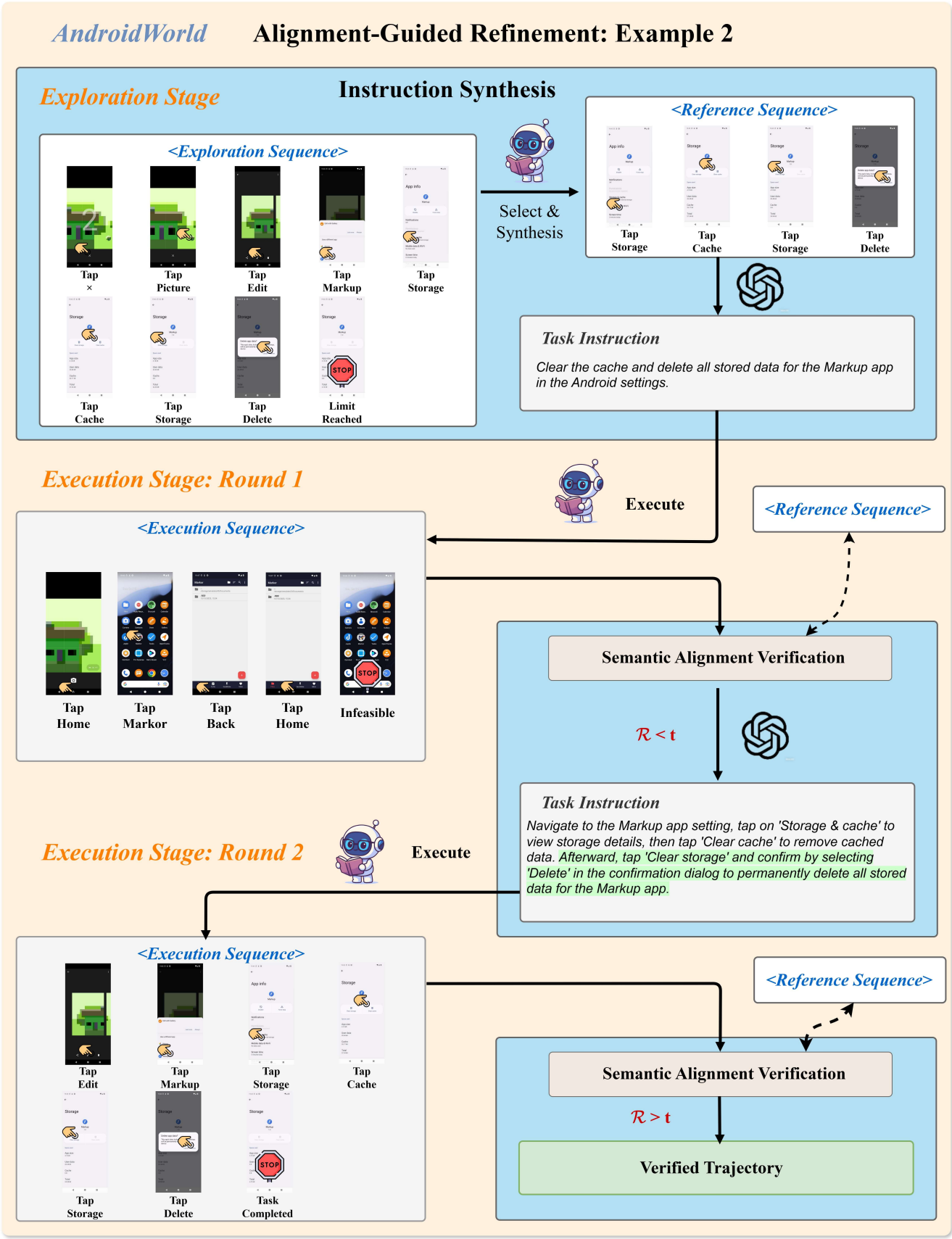
Table 8 shows the task instructions for the seven representative tasks in Fig. 7b, refined over three rounds by the **Alignment-Guided Refinement** module. As the process progresses, instructions are systematically enriched with critical semantic details—such as precise UI element properties (e.g., "clickable and focusable"), required navigation steps (e.g., "scroll down if the target is off-screen"), and explicit interaction modalities—leading to a consistent and monotonic increase in action-level recall. This demonstrates how structured, grounded refinement directly enables more accurate and executable agent behavior.

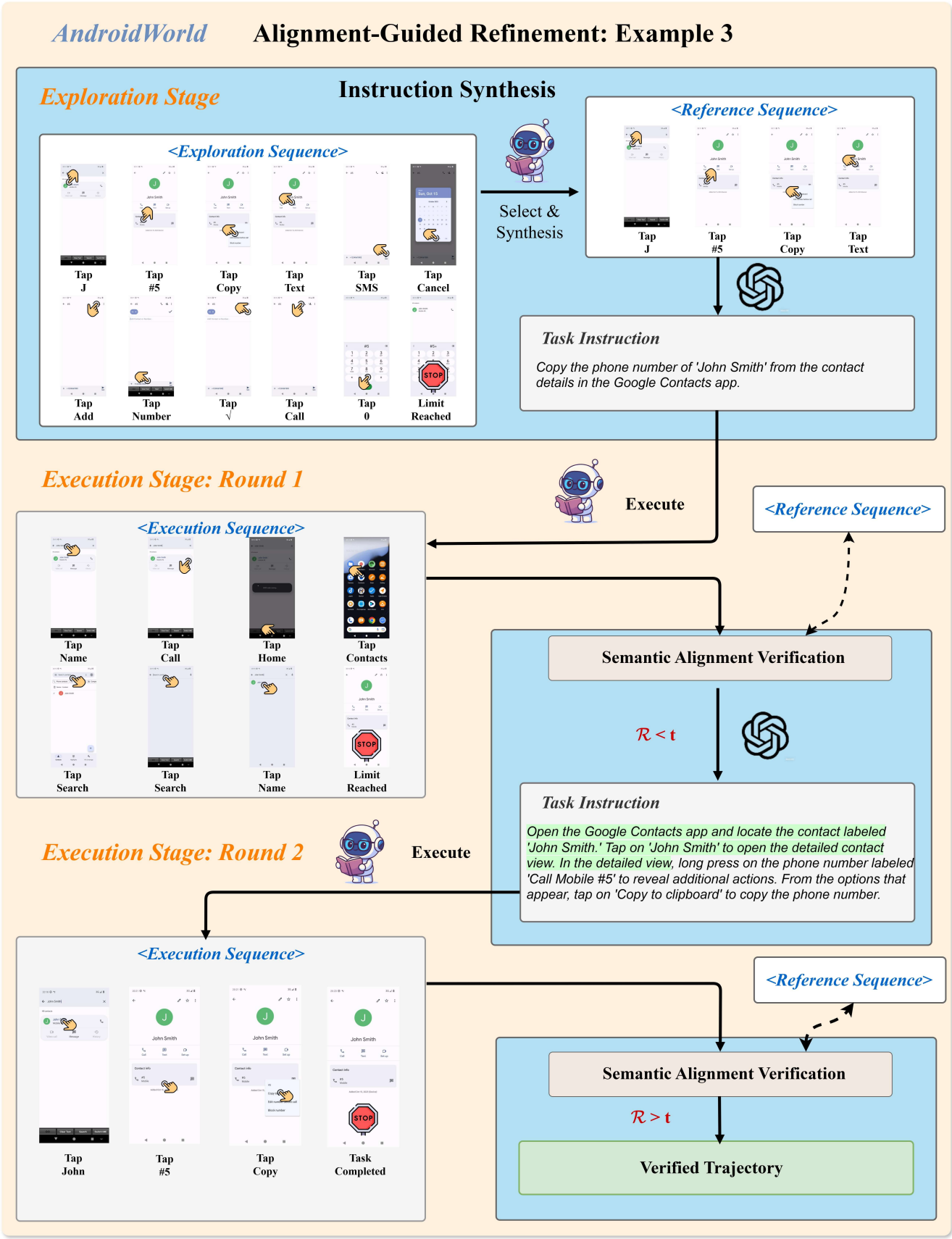
E. Illustrative Examples

In Figures 9 to 12, we present four illustrative cases of alignment-guided refinement, drawn from AndroidWorld (three) and WebArena (one). These examples are deliberately selected for their clarity and representativeness, enabling clear visualization of each stage in our pipeline: exploration, reference selection, instruction synthesis, execution, and refinement. The initial **Exploration Sequence** is typically noisy and repetitive, obscuring the underlying task intent. For example, in Example 1, the agent repeatedly navigates across multiple pages with redundant actions, making it impossible to infer the target date. From this raw trajectory, we extract a compact **Reference Sequence** that retains only goal-relevant steps, effectively distilling intent—such as

"perform operations for October 26." This refined sequence guides the generation of an initial task instruction, which the agent executes to produce an **Execution Sequence**. While this first attempt shows improved logical structure, it often lacks critical details, resulting in failure. For instance, the agent may omit the specific date due to an underspecified instruction. Through our **Refine** phase, we enrich the instruction semantically by aligning it with the reference sequence, enabling the agent to recover missing context. In Example 1, the refined instruction explicitly states, "Navigate to October 26, then click 'Confirm,"' leading to successful task completion. These cases demonstrate that our alignment-guided refinement mechanism transforms ambiguous, exploratory traces into precise, executable instructions, enabling agents to reliably complete complex, real-world GUI tasks.


AndroidWorld
Alignment-Guided Refinement: Example 1
Exploration Stage
Instruction Synthesis
<Exploration Sequence>
Tap Monthly
Tap 28
Tap back
Tap 27
Tap Oct 27
Tap 2022
Tap Cancel
Tap back
Tap back
Tap
Tap >
Tap Oct 26
Tap 2023
Limit Reached
Select & Synthesis
<Reference Sequence>
Tap Monthly
Tap 28
Tap 27
Tap Oct 27
Task Instruction
Add an event titled 'Meeting with Project Team at 10:00 AM' to October 26 (Thu) in the Simple Mobile Tools Calendar app.
Execution Stage: Round 1
Execute
<Execution Sequence>
Tap Home
Tap Calendar
Tap Time
Tap Time
Tap Home
Tap Calendar
Tap Home
Tap Calendar
Tap Home
Infeasible
<Reference Sequence>
Semantic Alignment Verification
R < t
Task Instruction
Open the Simple Mobile Tools Calendar app, switch to the 'Monthly view' if not already in that view, tap on the date 'October 26 (Thu)' in the calendar to select it, and then input 'Meeting with Project Team at 10:00 AM' into the editable field to add the event.
Execution Stage: Round 2
Execute
<Execution Sequence>
Tap Simple
Tap Menu
Tap Monthly
Tap >
Tap <
Tap 25
Tap >
Tap +
Tap Event
Type Title
Tap Yes
Task Completed
<Reference Sequence>
Semantic Alignment Verification
R > t
Verified Trajectory


AndroidWorld
Alignment-Guided Refinement: Example 2
Exploration Stage
Instruction Synthesis
<Exploration Sequence>
Tap
Tap
Picture
Tap
Edit
Tap
Markup
Tap
Storage
Tap
Cache
Tap
Storage
Tap
Delete
Limit
Reached
Select &
Synthesis
<Reference Sequence>
Tap
Storage
Tap
Cache
Tap
Storage
Tap
Delete
Task Instruction
Clear the cache and delete all stored data for the Markup app in the Android settings.
Execution Stage: Round 1
Execute
<Execution Sequence>
Tap
Home
Tap
Markor
Tap
Back
Tap
Home
Infeasible
<Reference Sequence>
Semantic Alignment Verification
R < t
Task Instruction
Navigate to the Markup app setting, tap on 'Storage & cache' to view storage details, then tap 'Clear cache' to remove cached data. Afterward, tap 'Clear storage' and confirm by selecting 'Delete' in the confirmation dialog to permanently delete all stored data for the Markup app.
Execution Stage: Round 2
Execute
<Execution Sequence>
Tap
Edit
Tap
Markup
Tap
Storage
Tap
Cache
Tap
Storage
Tap
Delete
Task
Completed
<Reference Sequence>
Semantic Alignment Verification
R > t
Verified Trajectory


AndroidWorld
Alignment-Guided Refinement: Example 3
Exploration Stage
Instruction Synthesis
<Exploration Sequence>
Tap J
Tap #5
Tap Copy
Tap Text
Tap SMS
Tap Cancel
Tap Add
Tap Number
Tap √
Tap Call
Tap 0
Limit Reached
Select & Synthesis
<Reference Sequence>
Tap J
Tap #5
Tap Copy
Tap Text
Task Instruction
Copy the phone number of 'John Smith' from the contact details in the Google Contacts app.
Execution Stage: Round 1
Execute
<Reference Sequence>
<Execution Sequence>
Tap Name
Tap Call
Tap Home
Tap Contacts
Tap Search
Tap Search
Tap Name
Limit Reached
Semantic Alignment Verification
$\mathcal{R} < t$
Task Instruction
Open the Google Contacts app and locate the contact labeled 'John Smith.' Tap on 'John Smith' to open the detailed contact view. In the detailed view, long press on the phone number labeled 'Call Mobile #5' to reveal additional actions. From the options that appear, tap on 'Copy to clipboard' to copy the phone number.
Execution Stage: Round 2
Execute
<Execution Sequence>
Tap John
Tap #5
Tap Copy
Task Completed
<Reference Sequence>
Semantic Alignment Verification
$\mathcal{R} > t$
Verified Trajectory

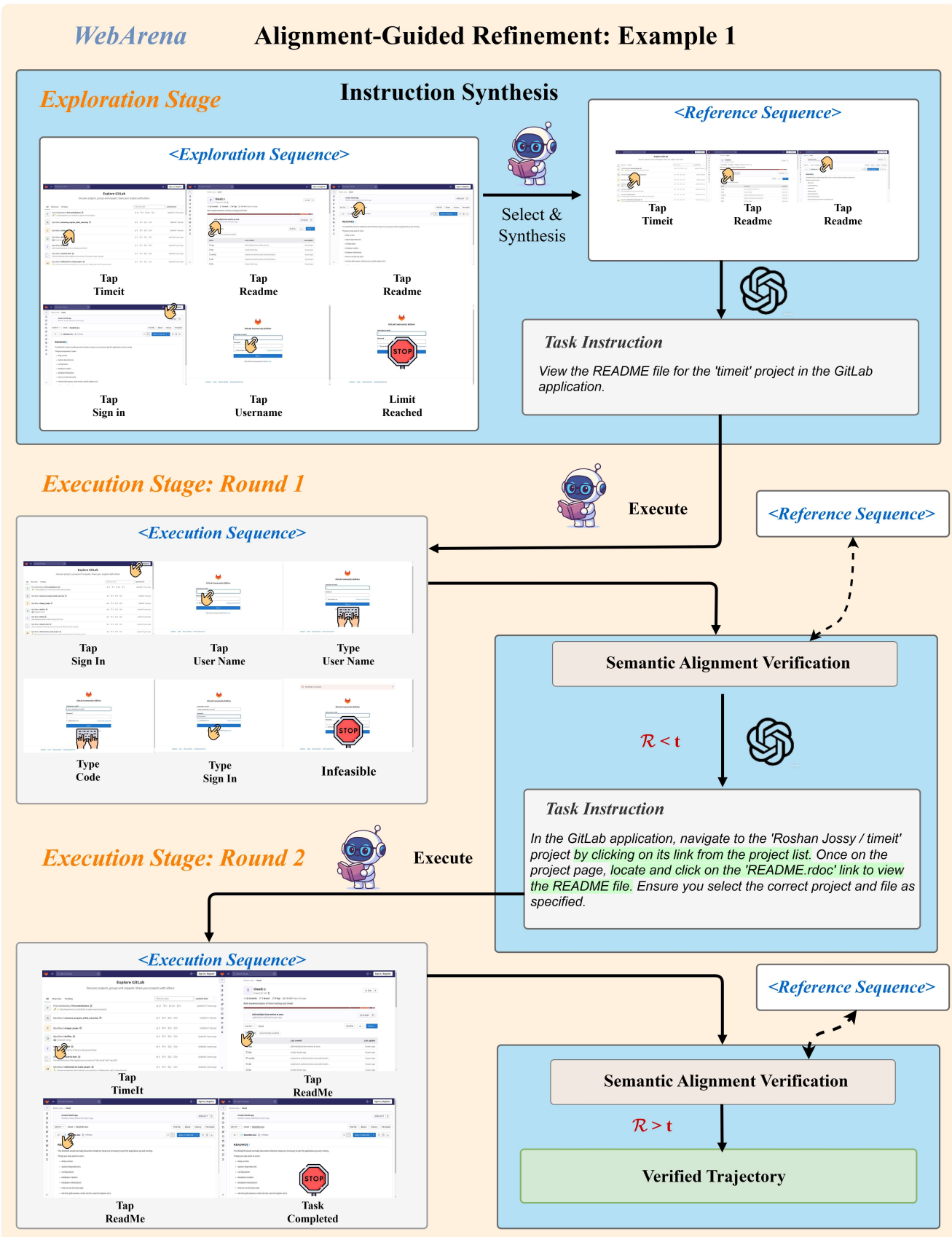

WebArena
Alignment-Guided Refinement: Example 1
Exploration Stage
Instruction Synthesis
<Exploration Sequence>
Tap
Timeit
Tap
Readme
Tap
Readme
Tap
Sign in
Tap
Username
Limit
Reached
Select &
Synthesis
<Reference Sequence>
Tap
Timeit
Tap
Readme
Tap
Readme
Task Instruction
View the README file for the 'timeit' project in the GitLab application.
Execution Stage: Round 1
Execute
<Execution Sequence>
Tap
Sign In
Tap
User Name
Type
User Name
Type
Code
Type
Sign In
Infeasible
<Reference Sequence>
Semantic Alignment Verification
$\mathcal{R} < t$
Task Instruction
In the GitLab application, navigate to the 'Roshan Jossy / timeit' project by clicking on its link from the project list. Once on the project page, locate and click on the 'README.rdoc' link to view the README file. Ensure you select the correct project and file as specified.
Execution Stage: Round 2
Execute
<Execution Sequence>
Tap
TimeIt
Tap
ReadMe
Tap
ReadMe
Task
Completed
<Reference Sequence>
Semantic Alignment Verification
$\mathcal{R} > t$
Verified Trajectory

Table 8. Evolution of synthesized instructions across three refinement rounds for the tasks in Figure 7b.

Task ID	Round	Synthesized Instruction
Task 1	1	In the Audio Recorder app, rename the audio recording titled '05-06-2025 06.14.28PM' to 'Meeting_Notes_05-06-2025'. First, double-click the text field containing '05-06-2025 06.14.28PM' to select it. Then, double-click the 'Copy' option from the edit menu. Double-click the date and time field again to ensure focused editing, and double-click the 'Paste' option from the edit menu. Finally, input 'Meeting_Notes_05-06-2025' into the text field to set the new name.
	2	In the Audio Recorder app, rename the audio recording titled '05-06-2025 06.14.28PM' to 'Meeting_Notes_05-06-2025'. First, locate the text field containing '05-06-2025 06.14.28PM' in the renaming prompt and double-click it to select the text. Then, double-click the 'Copy' option from the edit menu above the selected text. Next, double-click the same text field again to ensure it is focused for editing. After that, click the 'Paste' option from the edit menu to paste the copied text. Finally, input 'Meeting_Notes_05-06-2025' into the text field to set the new name, and click the 'Save' button to confirm the changes.
	3	In the Audio Recorder app (package name: com.dimowner.audiorecorder), properly set up the renaming interface to change the audio recording titled '05-06-2025 06.14.28PM' to 'Meeting_Notes_05-06-2025'. Start by double-clicking the text field containing '05-06-2025 06.14.28PM' to select it in the renaming prompt. Then, double-click the 'Copy' option from the edit menu above the selected text to copy it. Double-click the text field to refocus it if necessary. Click the 'Paste' option from the edit menu to paste the copied text. Finally, manually input 'Meeting_Notes_05-06-2025' into the text field for renaming and press the 'Save' button to confirm the change. Ensure that you are on the renaming dialog screen directly accessing the desired recording and follow the sequence precisely.
Task 2	1	Delete the audio records 'Record-10' and 'Record-4' in the Audio Recorder app.
	2	In the Audio Recorder app, first tap on the bookmark icon next to 'Record-10' to select it, then double-tap on 'Record-4' to select it. Finally, tap on the trash bin icon in the top-right corner to delete the selected records.
	3	In the Audio Recorder app, first tap on the bookmark icon next to 'Record-10' to select it. Then scroll down if necessary to ensure 'Record-4' is visible. Once visible, double-tap on 'Record-4' to select it by quickly tapping twice on the bookmark icon next to 'Record-4'. After both 'Record-10' and 'Record-4' are selected, tap on the trash bin icon in the top-right corner to delete these selected records.
Task 3	1	Sort the audio records by date (oldest) in the Audio Recorder app.
	2	In the Audio Recorder app (com.dimowner.audiorecorder), first tap on the filter icon to access sorting options. Then select 'By date (oldest)' to sort the audio records. After sorting, long-click on 'Record-11' to select it.
	3	In the Audio Recorder app (com.dimowner.audiorecorder), tap on the filter icon, which is clickable and focusable, to access sorting options. Ensure you select 'By date (oldest)' from the sorting options to sort the audio records correctly. After sorting, locate 'Record-11' on the screen and long-click on it to select it.
Task 4	1	Disable all calendar notifications in the Simple Mobile Tools Calendar Pro app by navigating through the settings menu.
	2	Disable all calendar notifications in the Simple Mobile Tools Calendar Pro app by navigating to the 'Settings' menu, selecting 'Customize notifications' under the 'Reminders' section, toggling off the 'All Calendar notifications' option, and verifying that notifications are fully disabled.
	3	Disable all calendar notifications in the Simple Mobile Tools Calendar Pro app by following these steps: 1) Tap on the 'Settings' icon located in the top-right corner of the app. 2) Select 'Customize notifications' under the 'Reminders' section. 3) Locate the toggle switch labeled 'All Calendar notifications' and ensure it is turned off. 4) Tap on the 'Calendar' icon to access additional notification settings. 5) Select 'Notifications' to return to the app info screen and verify that all calendar notifications are fully disabled.
Task 5	1	Explore Google's privacy and safety resources by navigating through the Privacy Policy and Safety Centre pages in the Chrome app.
	2	In the Chrome app, explore Google's privacy and safety resources by first tapping on the 'Privacy Policy' option in the Google Privacy & Terms menu. Then, double-tap on the 'Privacy Principles' link within the Technologies page. Next, double-tap on the 'Safety Centre' link to navigate to the Safety Centre page. Finally, tap on the 'Safety Centre' logo to refresh or reload the page.

Continued on next page

Table 8 – continued from previous page

Task ID	Round	Synthesized Instruction
	3	In the Chrome app, begin by navigating to the home screen if you are not already there, and open the Chrome app by tapping its icon. Once inside Chrome, access the Google Privacy & Terms menu by tapping on 'Settings' and then 'Privacy and security'. Scroll down until you find the 'Privacy Policy' option and tap on it. Next, navigate to the Technologies page and double-tap on the 'Privacy Principles' link. Then, proceed to the Safety Centre page by double-tapping on the 'Safety Centre' link. Finally, refresh the Safety Centre page by tapping on the 'Safety Centre' logo.
Task 6	1	Navigate through the sidebar in the Joplin app to explore different sections like 'All notes', 'Notebooks', and 'Trash'.
	2	Open the sidebar menu in the Joplin app by tapping the 'Sidebar' icon. Once the sidebar is visible, scroll left if necessary to make sure all options are accessible. Tap the 'Back' arrow in the top-left corner to return to the previous screen, confirming each stage of navigation.
	3	In the Joplin app, start by ensuring the main screen is visible. First, tap the 'Sidebar' icon, ensuring it is a clickable UI element with the content description 'Sidebar'. If the sidebar does not appear fully open or options aren't visible, scroll left on the scrollable UI element within the sidebar. Once you have verified the sidebar is open and accessible, identify the 'Back' arrow; it should be a visible, clickable UI element in the top-left corner of the screen. Tap the 'Back' arrow to return to the previous screen. Confirm each action by checking that the intended screen or menu appears after each tap or scroll.
Task 7	1	In the Tasks app, switch to the 'Default list', mark two 'New Task' entries as completed, and ensure a task is created using the 'Create new task' button by double tapping.
	2	In the Tasks app (org.tasks), first tap on the 'Default list' dropdown to select the 'Default list' if it's not already active. Then, locate the '+' icon with the 'Create new task' description and double tap it to create a new task. Next, scroll through the task list and find two tasks labeled as 'New Task'. Tap on the checkbox next to each 'New Task' entry to mark them as completed. Ensure the list is scrolled up or down as necessary to reveal the tasks and 'Create new task' button.
	3	In the Tasks app (org.tasks), start by tapping on the 'Default list' dropdown to ensure the 'Default list' is active. If it's not already active, tap it once to select it. Next, locate each 'New Task' entry in the task list and tap the checkbox next to them to mark them as completed. Tap on the checkbox, and ensure you have scrolled, if necessary, to bring the tasks into view. Once this is done, locate the '+' icon described as 'Create new task', tap once to create a new task, then tap on the 'Save' icon with the 'Save' description to save the task. Make sure the sequence is followed as described to achieve the user goal.

Prompt for Input Field Content Generation

Generate appropriate input content for the input field marked with a green bounding box and labeled as number index in the screenshot. Your generated content should be suitable for the usage scenario shown in this screenshot and represent what real users would typically enter in daily use. Your response should contain only the text that needs to be entered into the input field.

Prompt for Trajectory-to-Instruction Conversion (Android)

You are an expert specializing in inferring specific user tasks based on changes observed in mobile phone screenshots within an interaction trajectory. I will provide you with an interaction trajectory containing:

1. Action per Step: One of click, double_click, long_click, scroll_up, scroll_down, scroll_left, scroll_right, input. If the action is input, the input text is provided. Each non-scroll action includes the target element's attributes (e.g., content_description, text, resource_id).

2. Screenshots: A pre-action screenshot for each step (with a green bbox and step number) and one final post-trajectory screenshot. Pay close attention to the element inside the green bbox and all UI changes between consecutive screenshots.

3. Package Name: The package_name of the active app (use it to infer the app's common name).

Note: In Sub-Instruction, Analysis, Knowledge, and Task-Instruction, do not use the numeric marker from the green bbox; refer to elements by their visible labels or roles (e.g., "the Settings icon", "the field labeled Username").

Part 1: Analysis and Step Selection
- Analyze the entire trajectory (all actions and screenshot changes).
- Select a logically coherent subsequence that completes a clear user task.
- Exclude redundant/irrelevant/irrational steps.

Part 2: Output Generation Produce five parts (lists must match the original step count where specified):
- Sub-Instruction (List[str]): For each original step, write a concise, actionable instruction grounded in the observed UI, including concrete identifiers (filenames, times, visible text).
- Analysis (List[str]): For each original step, reason about likely next actions based on the new UI state.
- Knowledge (List[str]): For each original step, describe the general functionality of the interacted element, inferred from before/after screenshots (1–2 sentences).
- Selected-Step-ID (List[int]): The indices of the steps you chose for the coherent subsequence, in chronological order.
- Task-Instruction (str): A single actionable instruction that captures the user goal of the selected subsequence. Explicitly mention the inferred app name and any crucial specifics.

You must return only a JSON dictionary in the following format:
{
"Sub-Instruction": List[str],
"Analysis": List[str],
"Knowledge": List[str],
"Selected-Step-ID": List[int],
"Task-Instruction": str
}

The trajectory information will be provided below: 'trajectory_information'

RETURN ME THE DICTIONARY I ASKED FOR.

Android Action Reasoning & Execution Prompt

Role Definition You are an Android operation AI that fulfills user requests through precise screen interactions. Current and annotated screenshots are provided.

Action Catalog (STRICT JSON FORMAT)
1. Status Operations:
- Task Complete: "action_type": "status", "goal_status": "complete"
- Task Infeasible: "action_type": "status", "goal_status": "infeasible"
2. Information Actions:
- Answer Question: "action_type": "answer", "text": "<answer_text>"
3. Screen Interactions:
- Tap Element: "action_type": "click", "index": <visible_index>
- Long Press: "action_type": "long_press", "index": <visible_index>
- Scroll: "action_type": "scroll", "direction": "up|down|left|right", "index": <optional_container_index>
4. Input Operations:
- Text Entry: "action_type": "input_text", "text": "<content>", "index": <text_field_index>
- Keyboard Enter: "action_type": "keyboard_enter"
5. Navigation:
- Home Screen: "action_type": "navigate_home"
- Back Navigation: "action_type": "navigate_back"
6. System Actions:
- Wait Refresh: "action_type": "wait"

Current Objective
User Goal: task_goal

Execution Context
Action History:
history

Visible UI Elements (only interact with visible=true):
ui_elements

Core Strategy
1) Path Optimization: prioritize app drawer for app launch; always use input_text for text fields; verify visibility before interacting; scroll containers (if possible) before full-screen scrolls.
2) Error Handling: switch approach after > 1 failures; use scroll to reveal off-screen targets; try opposite scroll if needed.
3) Information Tasks: use answer for questions; verify data freshness.
4) Expert Techniques: knowledge

Response Format (STRICT)
Reasoning: [step-by-step analysis covering visibility checks, history effectiveness, alternatives, scroll considerations]
Action: [single JSON action from the catalog]

Generate response:

Step Summary Generation Prompt

Goal: {task_goal}

Before screenshot elements: {before_ui_elements}

After screenshot elements: {after_ui_elements}

Action: {action} Reasoning: {reasoning}

Provide a concise single-line summary (< 50 words) comparing screenshots and action outcome.
Include: intended effect; success/failure; key info for subsequent steps; critique if action/reasoning was flawed; any important data to remember across apps.
For answer or wait (no screen change), assume success.

Summary:

Matching Exploration vs. Agent Trajectories Prompt(Alignment Verification)

Objective: Determine which steps from the reference Exploration Trajectory were matched by the GUI Agent's trajectory. Report the count, matched exploration step indices, and the corresponding agent step indices.

Input:
1) Task Instruction:
task_instruction
2) Exploration Trajectory (numbered "Step i"):
exploration_trajectory_information
3) GUI Agent Trajectory (numbered "Step i"):
gui_agent_trajectory_information

Task: Compare trajectories; identify each exploration step matched by one or more agent steps;
list matched exploration indices; list all agent step indices contributing to any match;
count unique matched exploration steps.

Matching Rule: A match occurs if the agent performs the equivalent action or achieves the same
sub-goal. Multiple agent steps mapping to the same exploration step count as one toward the
match total. Still include all contributing agent step indices.

Output (JSON only):
{
"match_num": <int>,
"matched_exploration_id": [<int>, ...],
"matched_gui_agent_id": [<int>, ...]
}

Constraints: match_num == len(matched_exploration_id); sort both index lists ascending.

Refining Task Instruction Prompt

Role: You refine a given task instruction so a GUI Agent's execution more closely matches a reference exploration trajectory.

Inputs:
1) Initial Task Instruction:
task_instruction
2) Exploration Trajectory (ground-truth sequence):
exploration_trajectory_information
3) GUI Agent Trajectory (taken steps):
gui_agent_trajectory_information
4) Matched Exploration Steps (low-level instructions the agent achieved):
matched_low_level_instructions
5) Matched GUI Agent Steps (reasoning/action steps corresponding to matches):
matched_gui_agent_steps

Your Task: Identify exploration steps not matched by the agent; analyze agent deviations; rewrite the task instruction to guide execution toward the full exploration sequence.

Refinement Guidelines: Add specifics from unmatched steps (element texts/descriptions, action types like long_click or scroll_up, required input text); clarify sequence; address agent misunderstandings; preserve the original goal; explicitly name the app inferred from package_name; keep it actionable.

Output (JSON only):
{
"refined_high_level_instruction": "<refined instruction here>"
}

GUI Semantic Ambiguity Detection Prompt

You are an expert GUI interaction analyst tasked with identifying semantic-ambiguous actions in agent trajectories. Semantic-ambiguous actions are UI interactions whose functional meaning is unclear, context-dependent, or visually confounded.

Task Analyze

Analyze the provided trajectory and determine whether it contains any of the following three types of semantic ambiguity. A trajectory is considered to contain a type if at least one action in the sequence exhibits that characteristic.

Three Types of Semantic Ambiguity

1. Context Dependency

Definition: Identical or similar UI elements that trigger different functions depending on the surrounding context, application state, or screen location.
Examples:
- A "+" button that creates a new folder in one context but adds a contact in another
- A "Save" button that behaves differently in edit mode vs. creation mode
- The same icon appearing in different screens with distinct functions *Key Indicators*: Same element text/icon, different outcomes based on context; actions that require understanding the current application state or mode.

2. Sequential Dependency

Definition: Actions that only succeed or make sense after completing specific prerequisite steps, or interactions that form part of a multi-step workflow where order matters critically.
Examples:
- Submitting a form only after all required fields are filled
- Confirming a deletion only after initiating the delete action
- Accessing a feature that requires prior authentication or permission granting
- Multi-step workflows where skipping intermediate steps causes failure
Key Indicators: Actions that reference “after doing X” or “following Y”; failures due to missing prerequisites; explicit workflow sequences.

3. Visual Ambiguity

Definition: Visually similar or identical UI elements that correspond to distinct functions, making it difficult to distinguish their purpose based solely on appearance.
Examples: - Multiple icons with similar shapes but different meanings
- Text buttons with similar labels (“OK” vs. “Confirm” vs. “Done”) that have different effects
- Structurally similar list items or menu entries that perform different actions
- Elements that look clickable but aren’t, or vice versa
Key Indicators: Reasoning mentions visual similarity, confusion between elements, or need to carefully distinguish between look-alike components.

Analysis Guidelines

1. Read the entire trajectory including the task goal and all reasoning-action pairs.
2. For each action, consider:
- Does the element’s function depend on context? → Context Dependency
- Does this action require prior steps to work? → Sequential Dependency
- Are there visually similar elements that could be confused? → Visual Ambiguity
3. Mark True if ANY action in the trajectory exhibits the characteristic.
4. Be conservative: Only mark True if there is clear evidence in the reasoning or sequence.

Input Format

You will receive a trajectory string containing:
- Task Goal
- Step-by-step reasoning-action information

Output Format

Return ONLY a valid JSON dictionary:
{ "context_dependency": true,
"sequential_dependency": false,
"visual_ambiguity": false }
Important:
- Use lowercase with underscores for keys
- Use lowercase ‘true‘ and ‘false‘ (JSON boolean format)
- Do not include any explanatory text, only the JSON dictionary
- All three keys must be present

Trajectory to Analyze

{trajectory_string}

Your Classification

Provide your analysis as a JSON dictionary only:

Prompt for Trajectory-to-Instruction Conversion (Web)

You are an expert specializing in inferring specific user tasks based on changes observed in website screenshots within an interaction trajectory. I will provide you with an interaction trajectory containing the following information:

1. Action per Step: The action performed at each step, chosen from one of these types: ‘click‘, ‘scroll_up‘, ‘scroll_down‘, ‘input‘. If the action is ‘input‘, the input text will also be provided. Associated with each action (except scrolls) is information about the targeted UI element, including attributes like ‘bid‘, ‘type‘, ‘attributes‘, etc.
2. Screenshots: A screenshot taken before each action (labeled "Step i" in the bottom-right corner, indicating it precedes the i-th action) and a final screenshot taken after the last action (labeled "Final" in the bottom-right). The pre-action screenshots will feature a green bounding box (marked with the step number ’i’ in the top-left corner, also identifying the i-th UI element.) highlighting the element being interacted with. Pay close attention to the content within or associated with the green bounding box and the changes between the ’before’ and ’after’ screenshots for each step.
3. Current Page Title: The title of the current page after each action, which can provide context about the page’s content or purpose.
4. Current URL: The URL of the current page after each action, which can help identify the specific page or resource being accessed.

Note: In Sub-Instruction, Analysis, Knowledge, and Task-Instruction, do not use the numerical marker (from the top-left of the green box) to refer to the UI element. Instead, if you need to refer to the element, use a more natural description based on its visual characteristics or text content (e.g., "the ’Settings’ icon", "the text input field labeled ’Username’").

Your task involves two main parts:

Part 1: Analysis and Step Selection
- Analyze the entire provided trajectory (actions and screenshot changes).
- Identify a logically coherent subsequence of steps within the trajectory that represents a complete and reasonable user task.
- You must select actions that follow a clear and rational sequence toward achieving a specific goal.
- Eliminate any steps from the original trajectory that are redundant, irrelevant, irrational, or unnecessary for completing the identified task.

Part 2: Output Generation
Based on your analysis and step selection, devise the specific user goal or task. Your output must include five parts:
- Sub-Instruction (List[str]): For each step in the original trajectory, generate a natural language instruction corresponding to the action performed, based on the UI changes observed. This instruction should be concise, clear, actionable, and must incorporate key specific details visible in the screenshot, such as filenames, times, text content, or other relevant identifiers associated with the interacted element. Examples: "Scroll up to open the app drawer, revealing all installed applications.", "Click on the chat interface labeled ’General Discussion’.", "Input the username ’Agent’ into the username field."
- Analysis (List[str]): For each step in the original trajectory, provide an analysis of the potential subsequent actions or user intent, based on the UI changes resulting from the action and its Sub-Instruction. This analysis should involve step-by-step reasoning, considering the observed screen state and what actions become possible or logical next. Example: "After tapping the ’+’ button, a menu with options like ’New Document’ and ’New Folder’ appeared. I can create something new. Next step would be to tap ’New Document’, which might then prompt for a filename."
- Knowledge (List[str]): For each step ‘i‘ in the original trajectory, describe the general functionality of the UI element interacted with in that step. This description should be inferred by comparing the screenshot before action ‘i‘ (labeled "Step i") and the screenshot after action ‘i‘ (which will be the screenshot labeled "Step i+1", or "Final" for the last action). The description should be concise (1-2 sentences), focus on the general function revealed by the interaction (e.g., "Opens a settings menu," "Navigates back," "Selects an item," "Confirms an action"), avoid specific details unless necessary for clarity, and use generic terms like "this element" or "this button" or a functional description (e.g., "the search icon"). The length of this list must be equal to the number of steps in the original trajectory. Example: "Tapping this element initiates a search and displays matching results."
- Selected-Step-ID (List[int]): A list containing the integer step IDs (the ’i’ from ’Step i’) of the actions you selected in Part 1 as part of the coherent, logical task sequence. The IDs must be listed in the chronological order they appear in the selected subsequence.
- Task-Instruction (str): Based only on the selected subsequence of steps identified in Part 1, formulate a single, task instruction describing the overall task the user was likely trying to achieve through those selected steps. This instruction should correspond directly and efficiently to the selected sequence.

You must return only a JSON dictionary in the following format:
“‘json
{{
"Sub-Instruction": List[str],
"Analysis": List[str],
"Knowledge": List[str],
"Selected-Step-ID": List[int],
"Task-Instruction": str
}} “‘
The trajectory information will be provided below:
‘trajectory_information‘

RETURN ME THE DICTIONARY I ASKED FOR.

Web Action Reasoning & Execution Prompt

You are an agent trying to solve a web task based on the content of the page and user instructions. You can interact with the page and explore, and send messages to the user. Each time you submit an action it will be sent to the browser and you will receive a new page.

Instructions

Review the current state of the page and all other information to find the best possible next action to accomplish your goal. Your answer will be interpreted and executed by a program, make sure to follow the formatting instructions.

Goal:

goal

Observation of current step:

AXTree:

Note: [bid] is the unique alpha-numeric identifier at the beginning of lines for each element in the AXTree. Always use bid to refer to elements in your actions.

Note: You can only interact with visible elements. If the "visible" tag is not present, the element is not visible on the page.

a11y_tree

Focused element:

focused_element

History of interaction with the task:

history

Action space:

Note: This action set allows you to interact with your environment. Most of them are python function executing playwright code. The primary way of referring to elements in the page is through bid which are specified in your observations.

12 different types of actions are available:

noop(wait_ms: float = 1000)
scroll(delta_x: float, delta_y: float)
fill(bid: str, value: str)
select_option(bid: str, options: str | list[str])
click(bid: str, button: Literal['left', 'middle', 'right'] = 'left')
dblclick(bid: str, button: Literal['left', 'middle', 'right'] = 'left')
hover(bid: str)
press(bid: str, key_comb: str)
focus(bid: str)
clear(bid: str)
drag_and_drop(from_bid: str, to_bid: str)
upload_file(bid: str, file: str | list[str])

Only a single action can be provided at once. Example:

fill('a12', 'example with "quotes"')

Note:

- Some tasks may be game like and may require to interact with the mouse position in x, y coordinates.
- Some text field might have auto completion. To see it, you have to type a few characters and wait until next step.
- If you have to cut and paste, don't forget to select the text first.
- Coordinate inside an SVG are relative to it's top left corner.
- Make sure to use bid to identify elements when using commands.
- Interacting with combobox, dropdowns and auto-complete fields can be tricky, sometimes you need to use select_option, while other times you need to use fill or click and wait for the reaction of the page.

Abstract Example

Here is an abstract version of the answer with description of the content of each tag. Make sure you follow this structure, but replace the content with your answer:

<think>
Think step by step. If you need to make calculations such as coordinates, write them here. Describe the effect that your previous action had on the current content of the page.
</think>
<action>
One single action to be executed. You can only use one action at a time.
</action>

Concrete Example

Here is a concrete example of how to format your answer. Make sure to follow the template with proper tags:

<think>
From previous action I tried to set the value of year to "2022", using select_option, but it doesn't appear to be in the form. It may be a dynamic dropdown, I will try using click with the bid "a324" and look at the response from the page.
</think>
<action>
click('a324')
</action>

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, 2009. 1

[2] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. In *ACL*, 2024. 1

[3] Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Guicourse: From general vision language models to versatile gui agents. In *ACL*, 2025. 1

[4] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *CVPR*, 2024. 5, 6, 7, 3

[5] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *ACL*, 2024. 2, 1

[6] Kanzhi Cheng, Yantao Li, Fangzhi Xu, Jianbing Zhang, Hao Zhou, and Yang Liu. Vision-language models can self-improve reasoning via reflection. In *ACL*, 2025. 1

[7] Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. The browsergym ecosystem for web agent research. In *TMLR*, 2025. 1

[8] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *UIST*, 2017. 3, 1

[9] 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 *NeurIPS*, 2023. 1

[10] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In *ICML*, 2024. 1

[11] Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, et al. Agent ai: Surveying the horizons of multimodal interaction. *arXiv preprint arXiv:2401.03568*, 2024. 2, 1

[12] Tao Feng, Chuanyang Jin, Jingyu Liu, Kunlun Zhu, Haoqin Tu, Zirui Cheng, Guanyu Lin, and Jiaxuan You. How far are we from agi. *CoRR*, 2024. 2, 1

[13] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018. 1

[14] Ruize Gao, Feng Liu, Jingfeng Zhang, Bo Han, Tongliang Liu, Gang Niu, and Masashi Sugiyama. Maximum mean discrepancy test is aware of adversarial attacks. In *ICML*, 2021. 1

[15] Ruize Gao, Jiongxiao Wang, Kaiwen Zhou, Feng Liu, Binghui Xie, Gang Niu, Bo Han, and James Cheng. Fast and reliable evaluation of adversarial robustness with minimum-margin attack. In *ICML*, 2022. 1

[16] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *ICLR*, 2025. 1

[17] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *ICML*, 2017. 1

[18] Zhiting Hu and Tianmin Shu. Language models, agent models, and world models: The law for machine reasoning and planning. In *NeurIPS*, 2023. 1

[19] Keke Huang, Ruize Gao, Bogdan Cautis, and Xiaokui Xiao. Scalable continuous-time diffusion framework for network inference and influence estimation. In *WWW*, 2024. 1

[20] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 5, 6, 7, 3

[21] Chengyou Jia, Minnan Luo, Zhuohang Dang, Qiushi Sun, Fangzhi Xu, Junlin Hu, Tianbao Xie, and Zhiyong Wu. Agentstore: Scalable integration of heterogeneous agents as specialized generalist computer assistant. In *ICLR*, 2024. 1

[22] Chuanyang Jin, Yutong Wu, Jing Cao, Jiannan Xiang, Yen-Ling Kuo, Zhiting Hu, Tomer Ullman, Antonio Torralba, Joshua B Tenenbaum, and Tianmin Shu. Mmtom-qa: Multimodal theory of mind question answering. In *ACL*, 2024. 1

[23] Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. In *ECCV*, 2024. 2, 1

[24] M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *NeurIPS*, 2010. 1

[25] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent. In *KDD*, 2024. 2, 5, 6, 7, 3

[26] Hao Li, Qi Lv, Rui Shao, Xiang Deng, Yinchuan Li, Jianye Hao, and Liqiang Nie. Star: Learning diverse robot skill abstractions through rotation-augmented vector quantization. In *ICML*, 2025. 1

[27] Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In *NeurIPS*, 2024. 2, 3, 1

[28] Wei Li, Renshan Zhang, Rui Shao, Jie He, and Liqiang Nie. Cogvla: Cognition-aligned vision-language-action model via instruction-driven routing & sparsification. In *NeurIPS*, 2025. 1

[29] Wei Li, Jizhihui Liu, Yixing Li, Junwen Tong, Rui Shao, and Liqiang Nie. Consisvla-4d: Advancing spatiotemporal consistency in efficient 3d-perception and 4d-reasoning for robotic manipulation. In *CVPR*, 2026.

[30] Wei Li, Renshan Zhang, Rui Shao, Zhijian Fang, Kaiwen Zhou, Zhuotao Tian, and Liqiang Nie. Semanticvla: Semantic-aligned sparsification and enhancement for efficient robotic manipulation. In *AAAI*, 2026.

[31] Zajing Li, Bing Hu, Rui Shao, Gongwei Chen, Dongmei Jiang, Pengwei Xie, Jianye Hao, and Liqiang Nie. Global prior meets local consistency: Dual-memory augmented vision-language-action model for efficient robotic manipulation. In *CVPR*, 2026.

[32] Zaijing Li, Bing Hu, Rui Shao, Gongwei Chen, Dongmei Jiang, Pengwei Xie, Jianye Hao, and Liqiang Nie. Global prior meets local consistency: Dual-memory augmented vision-language-action model for efficient robotic manipulation. *arXiv preprint arXiv:2602.20200*, 2026. 1

[33] Musen Lin, Minghao Liu, Taoran Lu, Lichen Yuan, Yiwei Liu, Haonan Xu, Yu Miao, Yuhao Chao, and Zhaojian Li. Guirewalk: Massive data generation for gui agent via stochastic exploration and intent-aware reasoning. *arXiv preprint arXiv:2509.15738*, 2025. 3

[34] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *ICLR*, 2018. 1

[35] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 3

[36] Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *ICCV*, 2025. 1

[37] Xing Han Lù, Zdeněk Kasner, and Siva Reddy. Weblinx: Real-world website navigation with multi-turn dialogue. In *ICML*, 2024. 2, 3, 1

[38] Qi Lv, Hao Li, Xiang Deng, Rui Shao, Yinchuan Li, Jianye Hao, Longxiang Gao, Michael Yu Wang, and Liqiang Nie. Spatial-temporal graph diffusion policy with kinematic modeling for bimanual robotic manipulation. In *CVPR*, pages 17394–17404, 2025. 1

[39] Shikhar Murty, Christopher D Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. BAGEL: Bootstrapping agents by guiding exploration with language. In *ICML*, 2024. 2, 3

[40] Shikhar Murty, Dzmitry Bahdanau, and Christopher D Manning. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator. In *ICLR*, 2025. 1

[41] Runliang Niu, Jindong Li, Shiqi Wang, Yali Fu, Xiyu Hu, Xueyuan Leng, He Kong, Yi Chang, and Qi Wang. Screenagent: A vision language model-driven computer control agent. In *IJCAI*, 2024. 1

[42] Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *ACL Findings*, 2025. 1

[43] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. In *COLM*, 2024. 1

[44] Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*, 2024. 2, 3

[45] Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. In *IJCAI*, 2020. 1

[46] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD*, 2020. 3

[47] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy P Lillicrap. Androidinthewild: A large-scale dataset for android device control. In *NeurIPS*, 2023. 1

[48] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *ICLR*, 2025. 3, 6, 7, 1

[49] Rui Shao, Xiangyuan Lan, and Pong C Yuen. Deep convolutional dynamic texture learning with adaptive channel-discriminability for 3d mask face anti-spoofing. In *IJCB*, 2017. 1

[50] Rui Shao, Xiangyuan Lan, Jiawei Li, and Pong C Yuen. Multi-adversarial discriminative deep domain generalization for face presentation attack detection. In *CVPR*, 2019. 1

[51] Rui Shao, Tianxing Wu, and Ziwei Liu. Detecting and grounding multi-modal media manipulation. In *CVPR*, 2023. 1

[52] Rui Shao, Tianxing Wu, Jianlong Wu, Liqiang Nie, and Ziwei Liu. Detecting and grounding multi-modal media manipulation and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[53] Rui Shao, Wei Li, Lingsen Zhang, Renshan Zhang, Zhiyang Liu, Ran Chen, and Liqiang Nie. Large vlm-based vision-language-action models for robotic manipulation: A survey. *arXiv preprint arXiv:2508.13073*, 2025. 1

[54] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *ICML*, 2017. 3, 1

[55] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2024. 1

[56] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, 2016. 1

[57] Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. In *COLM*, 2023. 1

[58] Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, et al. A survey of neural code intelligence: Paradigms, advances and beyond. *arXiv preprint arXiv:2403.14734*, 2024. 2, 1

[59] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *ACL*, 2025. 2, 3, 4, 5, 6, 7, 1

[60] Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. In *ICLR Workshop on LLM Agents*, 2024. 2, 1

[61] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. 5, 6, 7, 3

[62] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *ACL*, 2023. 5, 6, 7, 3

[63] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022. 5, 6, 3

[64] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. In *COLM*, 2023. 2, 1

[65] Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. OS-copilot: Towards generalist computer agents with self-improvement. In *ICLR*, 2024. 2, 1

[66] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. Os-atlas: A foundation action model for generalist gui agents. In *ICLR*, 2025. 2, 1

[67] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *NeurIPS*, 2024. 1

[68] Fangzhi Xu, Qiushi Sun, Kanzhi Cheng, Jun Liu, Yu Qiao, and Zhiyong Wu. Interactive evolution: A neural-symbolic self-training framework for large language models. In *ACL*, 2025. 1

[69] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *NeurIPS*, 2022. 1

[70] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *ICLR*, 2023. 4

[71] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *ACL Findings*, 2024. 1

[72] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *CHI*, 2025. 1

[73] Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Ming Zhu, Juntao Tan, Thai Hoang, Zuxin Liu, Liangwei Yang, et al. Agentohana: Design unified data and training pipeline for effective agent learning. *arXiv preprint arXiv:2402.15506*, 2024. 1

[74] Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. Android in the zoo: Chain-of-action-thought for gui agents. In *EMNLP Findings*, 2024. 1

[75] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *ICML*, 2024. 1

[76] Longtao Zheng, Zhiyuan Huang, Zhenghai Xue, Xinrun Wang, Bo An, and Shuicheng Yan. Agentstudio: A toolkit for building general virtual agents. In *ICLR*, 2024. 1

[77] Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *ICLR*, 2024. 2

[78] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. In *ICLR*, 2024. 3, 6, 1

[79] Xurui Zhou, Gongwei Chen, Yuquan Xie, Zajing Li, Kaiwen Zhou, Shuai Wang, Shuo Yang, Zhuotao Tian, and Rui Shao. Hiconagent: History context-aware policy optimization for gui agents. In *CVPR*, 2026. 1

[80] Yijie Zhu, Rui Shao, Ziyang Liu, Jie He, Jizhihui Liu, Jiuru Wang, and Zitong Yu. H-gar: A hierarchical interaction framework via goal-driven observation-action refinement for robotic manipulation. In *AAAI*, 2026. 1