ATLASACT: RETRIEVAL-CONSTRAINED GROUNDING AND STRUCTURE-VALIDATED MEMORY FOR MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal large language models have enabled GUI agents to perform complex mobile tasks, yet agents may identify the intended control correctly while still failing to localize it precisely, and recurring interactions incur repeated model decisions. We propose ATLASACT, a framework that uses accessibility (A11y) structure as a shared executable interface for action grounding and interaction reuse. First, for element-targeted actions, the Actor specifies a target semantically, and retrieval-constrained grounding resolves it against multimodal A11y-node representations. A confidence router directly executes the top-ranked node when retrieval confidence is high and invokes the Grounder only when disambiguation is needed, reducing unnecessary Grounder calls. Second, structure-validated memory maintains Tips as decision guidance and Shortcuts as reusable action sequences. Shortcuts bypass per-step Actor and Grounder calls, while A11y-based entry and execution checks return control to the Actor upon structural mismatch. Experiments on four mobile GUI benchmarks demonstrate improvements in grounding and end-to-end task execution. Memory ablations show that checked Shortcuts reduce average decision steps by 10.4% relative to Tips alone on a fixed task subset, without lowering the observed overall success rate. With Qwen3.8-27B and interaction memory, ATLASACT achieves success rates of 67.67% on AndroidWorld and 95.40% on AndroidDaily, outperforming all evaluated baselines on both benchmarks. Our code is available at `https://anonymous.4open.science/r/AtlasAct-A003`.

1 INTRODUCTION

Multimodal large language models (MLLMs) have enabled GUI agents to perform increasingly complex mobile tasks Zhang et al. (2025); Wang et al. (2024); Lin et al. (2025); Qin et al. (2025); Xu et al. (2025). Yet reliable execution requires translating semantic intentions into precise actions as interfaces change, while familiar interaction sequences often incur repeated model decisions. Mobile GUI structure offers useful signals for both problems: interactive elements provide discrete action targets, and recurring interface states provide observable conditions for reusing previously acquired actions.

First, specifying an intended target does not ensure that a pointer action reaches it. Many visual GUI agents directly predict screen coordinates from instructions and screenshots Cheng et al. (2024); Gou et al. (2025); Wu et al. (2024); Qin et al. (2025), while fine-grained grounding remains challenging for small or closely spaced UI elements You et al. (2024), as illustrated in Figure 1(a). In environments exposing accessibility (A11y) metadata, UI nodes provide explicit target regions Rawles et al. (2025), allowing element-targeted grounding to be formulated as node selection. However, marking many actionable nodes with Set-of-Marks (SoM) prompting Yang et al. (2023) can introduce irrelevant candidates and clutter the screenshot You et al. (2024). Building on prior exploration of task-conditioned candidate filtering Hao et al. (2025), we investigate multimodal A11y-node retrieval to construct compact candidate sets while retaining the full screenshot for contextual disambiguation, and avoiding an additional grounding-model call when retrieval is sufficiently confident.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

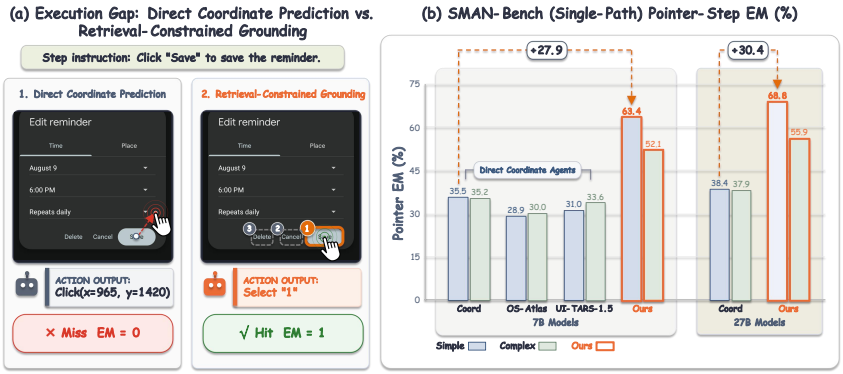

Figure 1: **From Direct Coordinate Prediction to Retrieval-Constrained Grounding.** (a) Direct coordinate prediction may miss the intended control, whereas retrieval-constrained grounding selects the target from a small set of task-relevant UI candidates. (b) Pointer-step exact-match accuracy on SMAN-Bench (Single-Path). This metric is computed only over the pointer-action subset.

Second, reusable action sequences can reduce repeated model decisions, but their applicability can change with the interface. Existing agents reuse guidance, trajectories, and executable routines Wang et al. (2025); Li et al. (2026b); Xiao et al. (2026). A different entry page can invalidate the first action, while dynamic content or unexpected pop-ups can invalidate later steps. Although Mobile-Agent-E uses model-based judgments to assess Shortcut preconditions and action outcomes Wang et al. (2025), current A11y observations provide a basis for explicit structural checks before and during replay.

We propose ATLASACT, a mobile GUI agent framework combining **Retrieval-Constrained Grounding** and **Structure-Validated Memory**. For element-targeted pointer actions, Retrieval-Constrained Grounding decouples semantic target specification from executable target resolution: the Actor produces a natural-language target, which is resolved against a compact set of actionable A11y nodes using multimodal node representations. High-confidence Top-1 matches are executed directly, while the Grounder is invoked only when disambiguation is needed. Following Mobile-Agent-E Wang et al. (2025), Structure-Validated Memory maintains Tips as decision guidance and Shortcuts as reusable action sequences, while augmenting Shortcut reuse with deterministic A11y-based entry and per-step checks. Shortcuts avoid repeated Actor and Grounder calls during normal replay, while failed structural checks return control to the Actor. Together, the two modules use current A11y structure as a shared executable interface for grounding semantic targets and validating reusable routines.

We evaluate ATLASACT on four mobile GUI benchmarks, covering both action-level grounding and end-to-end task execution. With Qwen3.8-27B Qwen Team (2026), ATLASACT improves exact-match accuracy over matched-backbone dense SoM by 10.30 and 6.43 percentage points on the simple and complex single-path splits of SMAN-Bench Xu et al. (2026), respectively. In end-to-end evaluation, ATLASACT with interaction memory achieves success rates of 67.67% on AndroidWorld Rawles et al. (2025) and 95.40% on AndroidDaily Sui et al. (2026), achieving the highest success rates among all evaluated methods on both benchmarks. Memory ablations on AndroidWorld further show that adding structure-validated Shortcuts improves execution efficiency, reducing Actor decision steps by 10.4% on a fixed task subset without reducing overall task success.

Our main contributions are:

- We propose **Retrieval-Constrained Grounding**, which decouples semantic target specification from spatial grounding by resolving natural-language targets to actionable A11y nodes.
- We introduce **Structure-Validated Memory**, which uses A11y-based entry and per-step checks to support reliable Shortcut reuse without repeated Actor and Grounder decisions during normal replay.

- We evaluate ATLASACT on **four mobile GUI benchmarks**, demonstrating consistent improvements in grounding and end-to-end task execution and providing extensive analyses of the proposed design choices.

2 RELATED WORK

2.1 GUI GROUNDING AND CANDIDATE SELECTION

SeeClick, OS-Atlas, and UGround improve screenshot-based GUI grounding through specialized training Cheng et al. (2024); Wu et al. (2024); Gou et al. (2025), while UI-TARS integrates perception, grounding, reasoning, and action generation within a native GUI agent Qin et al. (2025). Structured representations expose explicit targets: SoM overlays visual markers on image regions Yang et al. (2023), while AppAgent and M3A use numbered UI elements derived from interface metadata Zhang et al. (2025); Rawles et al. (2025). However, such representations may still present many elements irrelevant to the current instruction.

Task-conditioned filtering narrows this candidate space. RecAgent combines keyword, semantic, and LLM-based recommendation for mobile interaction Hao et al. (2025), while SeeAct and Prune4Web rank or filter candidate elements in web environments Zheng et al. (2024); Zhang et al. (2026). In our setting, retrieval operates after the Actor specifies an element target. We use multimodal A11y-node representations to retrieve a compact candidate set, executing the top-ranked candidate when retrieval confidence is high and otherwise invoking the Grounder to select from this set.

2.2 INTERACTION MEMORY AND REUSABLE GUI ROUTINES

Mobile GUI agents reuse experience as guidance, trajectories, and executable routines. Mobile-Agent-E maintains Tips and Shortcuts through experience reflection and uses model-based judgments to assess Shortcut applicability and execution outcomes Wang et al. (2025). EchoTrail-GUI retrieves critic-curated trajectories as in-context guidance Li et al. (2026b). SE-GA combines memory retrieval with policy training Jin et al. (2026), while MAGNET maintains procedural workflows and visual element exemplars Sun et al. (2026).

However, model-based admission and outcome assessment do not explicitly verify structural applicability throughout replay. Interface changes can therefore invalidate later actions after a Shortcut has been invoked. To reduce this risk, ATLASACT uses deterministic A11y-based entry and per-step checks to constrain Shortcut invocation and replay.

3 PROBLEM FORMULATION

We formulate mobile GUI interaction as a finite-horizon sequential decision problem. At decision step t, the agent conditions on the screenshot I_t, accessibility information X_t, task goal g, interaction history H_t, and any available interaction memory M, and selects the next operation:

$$a_t = \Pi(I_t, X_t, g, H_t, M), \tag{1}$$

until emitting `finish` or reaching the prescribed step budget. The operation is either an atomic action or the invocation of a reusable Shortcut.

For atomic actions, the Actor determines the action type and, for element-targeted pointer actions, specifies a natural-language target o_t. Other actions are directly parameterized by the Actor.

Let

$$\mathcal{V}_t = \{v_{t,i}\}_{i=1}^{N_t} \tag{2}$$

denote the actionable UI nodes extracted from X_t. For targets represented by these nodes, grounding resolves o_t to a node $\hat{v}_t \in \mathcal{V}_t$ in the current interface.

A Shortcut represents a reusable sequence of atomic actions.

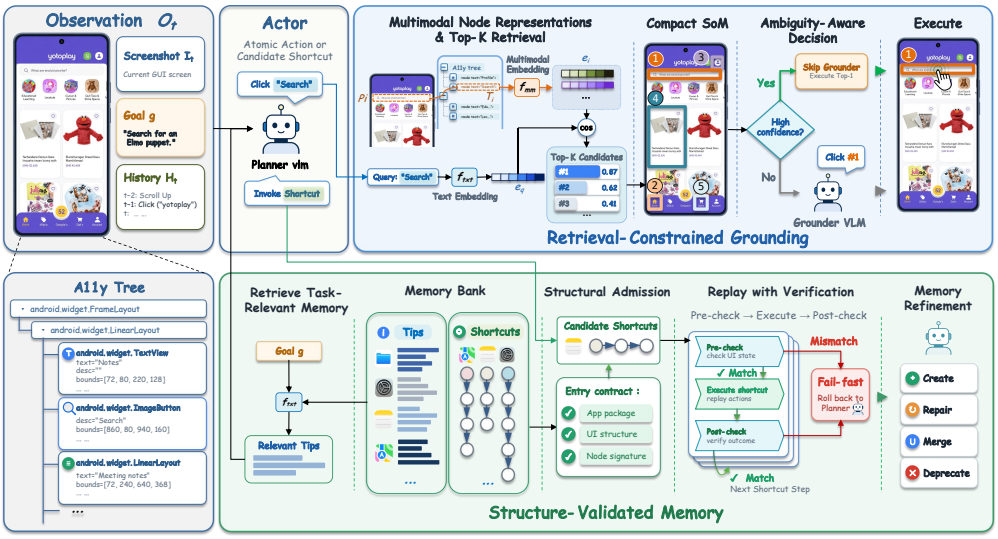

Figure 2: **Overview of ATLASACT.** ATLASACT combines retrieval-constrained grounding with structure-validated memory, using the A11y tree as a shared executable structure for both pointer grounding and Shortcut reuse.

4 METHODS

4.1 OVERVIEW

To improve the reliability of mobile GUI interaction, we propose ATLASACT, a framework that uses current accessibility (A11y) information to constrain action grounding and experience reuse. As shown in Figure 2, ATLASACT consists of **Retrieval-Constrained Grounding** and **Structure-Validated Memory**.

Retrieval-Constrained Grounding uses multimodal node representations to retrieve a compact candidate set from $\mathcal{V}_t$. When retrieval confidence is high, the top-ranked candidate is executed directly; otherwise, a Grounder selects among the retrieved candidates using a compact Set-of-Marks (SoM) screenshot. Structure-Validated Memory maintains Tips as decision guidance and Shortcuts as reusable action sequences Wang et al. (2025). Shortcuts avoid per-step Actor and Grounder calls during normal replay, while structural entry and per-step checks constrain their invocation and execution. A failed required check interrupts replay and returns control to the Actor.

4.2 RETRIEVAL-CONSTRAINED GROUNDING

Task-conditioned candidate retrieval. Given a natural-language target o_t produced by the Actor for an element-targeted click or long-press, we retrieve relevant nodes from the actionable A11y-node set $\mathcal{V}_t$. For each node $v_i \in \mathcal{V}_t$ with bounding box b_i, we extract its screenshot crop $P_i = \mathrm{crop}(I_t, b_i)$ and textual metadata T_i, including available text, content descriptions, and node labels. A shared multimodal embedding function f_{mm} encodes the node and the target description:

$$e_i = f_{\mathrm{mm}}(T_i, P_i), \qquad q_t = f_{\mathrm{mm}}(o_t, \varnothing), \tag{3}$$

where $\varnothing$ indicates that no image is provided for the target query. Thus, node representations can incorporate both textual attributes and local visual appearance in the same embedding space as the target query (Appendix F).

We rank nodes by cosine similarity and retain up to K candidates:

$$s_i = \frac{q_t^\top e_i}{\|q_t\|_2 \|e_i\|_2}, \qquad \mathcal{C}_t = \mathrm{TopK}_{v_i \in \mathcal{V}_t}(s_i). \tag{4}$$

When the Grounder is invoked, only nodes in $\mathcal{C}_t$ are marked on the original screenshot. This preserves global screen context while restricting selection to a compact, task-relevant candidate set.

Confidence-based Grounder invocation. To avoid unnecessary Grounder calls, we estimate the probability that the top-ranked retrieved node matches the intended target:

$$\hat{p}_t = h(\phi_t), \tag{5}$$

where h is a lightweight confidence estimator and ϕ_t summarizes retrieval-score statistics, including top scores, score gaps, score dispersion, and candidate count. Training details are provided in Appendix A.2.

When $\hat{p}_t \geq \tau$, the top-ranked node is selected without invoking the Grounder. Otherwise, the Grounder receives the Actor's step instruction and a compact SoM screenshot over $\mathcal{C}_t$, and returns the index of a selected candidate. In either case, the selected node's bounding-box center provides the coordinates for the Actor-specified click or long-press action. The threshold τ controls Grounder invocation frequency; the resulting accuracy–invocation trade-off is analyzed in Appendix B.

4.3 Structure-Validated Memory

Following Mobile-Agent-E (Wang et al., 2025), we maintain Tips as natural-language guidance for the Actor and Shortcuts as parameterized action sequences. Shortcuts replay multiple atomic actions without per-step Actor or Grounder calls during normal execution. Structure-Validated Memory augments this reuse with deterministic A11y-based checks at Shortcut entry and throughout replay, returning control to the Actor when a required execution condition fails. The memory bank evolves across tasks through post-task reflection.

Memory evolution. For each dataset, we initialize an empty memory bank $\mathcal{M}_1 = \varnothing$ and execute tasks sequentially in the dataset-specified order. Let $\mathcal{M}_i$ denote the Tips and Shortcuts available at the start of task i, accumulated from previously completed tasks in the same dataset. After task i terminates, the memory model updates the bank using its execution trajectory ξ_i, a binary task outcome y_i, and replay-check feedback $\mathcal{F}_i$:

$$\mathcal{M}_{i+1} = \mathrm{Update}_{\mathrm{mem}}\left(\mathcal{M}_i, \xi_i, y_i, \mathcal{F}_i\right). \tag{6}$$

Here, y_i indicates only whether the task was successfully completed. The memory model distills execution experience into Tips and reusable action sequences into Shortcuts. Each Shortcut stores its action sequence, declared content slots, and an entry signature describing its starting interface. Reflection may revise Tips and create, repair, replace, merge, or deprecate Shortcuts. The updated bank is retained for the next task, allowing subsequent tasks to reuse and refine prior experience without changing the Actor or Grounder parameters.

Structural admission. During task execution, relevant Tips are retrieved using the task goal and app name and supplied to the Actor. Shortcut candidates are drawn from the current-app and global memory pools. Each Shortcut S stores an entry signature $\sigma_S = (\mathrm{pkg}_S, \Phi_S)$, where pkg_S denotes its starting application package and Φ_S contains fingerprints of interactive A11y nodes. For nonempty Φ_S, structural coverage under the current fingerprint set Φ_t is

$$\mathrm{cov}(\Phi_S, \Phi_t) = \frac{|\Phi_S \cap \Phi_t|}{|\Phi_S|}. \tag{7}$$

Shortcut candidates are filtered by package compatibility and the coverage threshold γ before being presented to the Actor for possible invocation. Detailed fingerprint construction is provided in Appendix A.4.

Replay checks and recovery. Upon invocation, a memory model fills any declared content slots from the task and current context, after which the Shortcut is expanded into atomic actions. These actions are replayed without per-step Actor or Grounder calls, while A11y-based checks verify execution before and after each step.

Before each action, action-specific pre-checks test whether the stored action remains executable in the current interface. For clicks and long-presses, stored resource identifiers or labels are used to re-resolve the target; missing or ambiguous targets prevent execution.

After each action, post-checks are evaluated on a fresh A11y observation. Strong checks test explicit expected conditions, such as whether typed text appears in an input field, while weak checks use

Table 1: **Single-step grounding performance.** We report Exact Match (EM) and Type Match (TM) on AndroidControl-low (AC-low) and the simple and complex splits of SMAN-Bench. *Finetune* indicates whether the grounding model is fine-tuned for GUI grounding.

Grounding	Method	Backbone Model	Finetune	AC-low		SMAN-Simple		SMAN-Complex	
				EM↑	TM↑	EM↑	TM↑	EM↑	TM↑
Direct Coordinate	Qwen2.5	Qwen2.5-VL-7B	–	83.93	89.53	33.96	88.14	33.59	84.09
	OS-Atlas	Qwen2-VL-7B	✓	88.13	94.67	28.86	86.37	29.95	87.73
	UI-TARS-1.5	Qwen2.5-VL-7B	✓	84.86	**96.92**	31.01	80.04	33.59	81.22
	Qwen3.8	Qwen3.8-27B	–	85.89	94.39	37.50	91.74	36.38	93.23
Dense SoM	AppAgent	Qwen2.5-VL-7B	–	74.39	87.94	50.48	88.41	36.46	78.26
	AppAgent	Qwen3.8-27B	–	84.21	92.24	56.60	87.66	47.63	86.72
Retrieval-Constrained	ATLASACT	Qwen2.5-VL-7B	✓	**88.22**	94.67	59.98	87.55	49.66	88.66
	ATLASACT	Qwen3.8-27B	–	87.85	94.02	**66.90**	**93.24**	**54.06**	**95.35**

coarser transition signals, such as changes in the screen fingerprint. A failed strong check indicates that a required condition was not met, whereas a failed weak check alone is not treated as execution failure.

A failed pre-check or strong post-check terminates replay, disables the Shortcut for the remainder of the episode, and returns control to the Actor on the current interface. Weak-check failures are recorded without interrupting replay. The resulting feedback is retained for post-task memory refinement; episode-level disabling is separate from revising or deprecating a Shortcut in persistent memory. Detailed entry-signature construction and action-specific checking rules are provided in Appendix A.4.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Benchmarks and Metrics. We evaluate ATLASACT on action prediction and grounding, as well as end-to-end task execution. For action-level evaluation, we use AndroidControl Li et al. (2024) under low- and high-level instruction settings, and the Common-Simple and Common-Complex subsets of SMAN-Bench Xu et al. (2026), abbreviated as SMAN-Simple and SMAN-Complex in the tables. Following the evaluation protocols of the respective benchmarks, we report **Exact Match (EM)** and **Type Match (TM)** on AndroidControl and under the single-path setting of SMAN-Bench, and EM together with **Step Efficiency (SE)** under its multi-path setting. EM measures action correctness under the respective benchmark matching rules, while TM measures action-type correctness. Higher EM and TM and lower SE indicate better performance.

For end-to-end evaluation, we use AndroidWorld Rawles et al. (2025) and AndroidDaily Sui et al. (2026), and report task **Success Rate (SR)**. We evaluate ATLASACT with and without cross-task interaction memory. For our framework, we additionally report the average number of decision steps (**Avg. Steps**), with atomic actions within a Shortcut not counted separately. This metric serves as a descriptive proxy for repeated Actor inference and is used only for within-framework analysis, since differences in task success and termination behavior can confound cross-agent comparisons of average step counts. Splits, counts, and matching rules are given in Appendix G.

Baselines. For grounding, we compare against direct-coordinate prediction using Qwen2.5-VL-7B-Instruct Bai et al. (2025), Qwen3.8-27B Qwen Team (2026), OS-Atlas-7B Wu et al. (2024), and UI-TARS-1.5-7B Qin et al. (2025). We also include AppAgent-style dense Set-of-Marks (SoM) grounding Zhang et al. (2025) using Qwen2.5-VL-7B-Instruct and Qwen3.8-27B, matching the grounding backbones of our two configurations. For end-to-end execution, we compare against the four direct-action models above, together with AppAgent, SE-GA Jin et al. (2026), Agent S2 Agashe et al. (2025), and Mobile-Agent-E Wang et al. (2025). We retain the original inference procedure of

Table 2: **Multi-step grounding performance.** We report Exact Match (EM) and Type Match (TM) on AndroidControl-high (AC-high), and EM with Step Efficiency (SE) on the simple and complex splits of SMAN-Bench. *Finetune* indicates whether the grounding model is fine-tuned for GUI grounding. Higher EM/TM and lower SE are better.

Grounding	Method	Backbone Model	Finetune	AC-high		SMAN-Simple		SMAN-Complex	
				EM↑	TM↑	EM↑	SE↓	EM↑	SE↓
Direct Coordinate	Qwen2.5	Qwen2.5-VL-7B	–	59.44	74.95	21.19	6.24	17.34	4.79
	OS-Atlas	Qwen2-VL-7B	✓	63.64	78.69	18.62	6.22	15.99	4.71
	UI-TARS-1.5	Qwen2.5-VL-7B	✓	63.74	81.78	17.06	6.25	14.30	4.77
	Qwen3.8	Qwen3.8-27B	–	67.20	**83.36**	35.14	5.90	26.57	4.69
Dense SoM	AppAgent	Qwen2.5-VL-7B	–	51.21	71.31	39.75	5.69	22.18	4.78
	AppAgent	Qwen3.8-27B	–	65.70	79.81	51.93	5.25	35.79	4.64
Retrieval-Constrained	ATLASACT	Qwen2.5-VL-7B	✓	**70.28**	82.24	50.70	5.28	35.87	4.66
	ATLASACT	Qwen3.8-27B	–	68.13	82.34	**60.09**	**4.80**	**43.65**	**4.50**

each baseline where applicable, while aligning model versions for controlled comparison. Further details are provided in Appendix G.

Implementation. We evaluate 7B and 27B configurations. The 7B configuration uses UI-TARS-1.5-7B as the Actor and Qwen2.5-VL-7B-Instruct as the Grounder, both adapted using LoRA. The 27B configuration uses Qwen3.8-27B for both roles without additional fine-tuning. Unless otherwise specified, we set $K = 5$, $\tau = 0.99$, and $\gamma = 0.6$, all fixed prior to evaluation. Sensitivity to the Shortcut coverage threshold and the Tip budget is reported in Appendix C. Further model, training, and inference details are provided in Appendix A.

5.2 MAIN RESULTS

5.2.1 GROUNDING PERFORMANCE

We evaluate ATLASACT under two configurations: a 7B system with LoRA adaptation and a 27B system without additional VLM fine-tuning. Tables 1 and 2 report results across the AndroidControl instruction settings and the single- and multi-path settings of SMAN-Bench.

Results with 7B models. With LoRA adaptation, ATLASACT-7B achieves the highest EM among the evaluated 7B methods across all reported settings. On AC-high, it reaches 70.28 EM, compared with 63.64 for OS-Atlas and 63.74 for UI-TARS-1.5. The advantage also holds on SMAN-Bench under both single- and multi-path evaluation. In the multi-path setting, ATLASACT-7B achieves 50.70 and 35.87 EM on SMAN-Simple and SMAN-Complex, exceeding UI-TARS-1.5-7B by 33.64 and 21.57 percentage points, respectively. These results demonstrate strong performance relative to similarly sized GUI-specialized models when GUI-specific adaptation is available.

Results with 27B models. At 27B, ATLASACT, direct-coordinate prediction, and dense SoM use the same Qwen3.8-27B backbone without additional VLM fine-tuning. ATLASACT achieves higher EM than both baselines across all reported settings. Relative to dense SoM, the gains on SMAN-Simple and SMAN-Complex are 10.30 and 6.43 percentage points under single-path evaluation, and 8.16 and 7.86 points under multi-path evaluation. Appendix E contrasts full-page and compact SoM on the same screens. The improvements also hold on AC-low and AC-high, with gains of 3.64 and 2.43 points, respectively. In addition, ATLASACT-27B obtains the lowest SE among the evaluated methods on both multi-path SMAN subsets, at 4.80 and 4.50. These matched-backbone results support the effectiveness of retrieval-constrained grounding without requiring additional VLM fine-tuning.

The gains are more pronounced on SMAN-Bench than on AndroidControl. In the examples we examined, the Chinese-app interfaces in SMAN-Bench appear more visually crowded than those in AndroidControl. This may partly explain the larger gains, as task-conditioned candidate filtering reduces competing marks and focuses the grounding VLM on relevant UI elements.

Table 3: **End-to-end task execution on AndroidWorld and AndroidDaily.** All agents use within-task interaction history. *Finetune* indicates whether the agent model is fine-tuned for GUI interaction, while *Memory* indicates whether cross-task interaction memory is enabled. SR denotes task success rate and Avg. Steps denotes the average number of decision steps.

Agent	Backbone Model	Finetune	Mem	AndroidWorld		AndroidDaily	
				SR↑	Avg. Steps↓	SR↑	Avg. Steps↓
Qwen2.5	Qwen2.5-VL-7B	–	–	18.53	–	22.99	–
OS-Atlas	Qwen2-VL-7B	✓	–	14.22	–	34.48	–
UI-TARS-1.5	Qwen2.5-VL-7B	✓	–	31.90	–	27.59	–
Qwen3.8	Qwen3.8-27B	–	–	54.31	–	79.31	–
AppAgent	Qwen2.5-VL-7B	–	–	24.14	–	45.98	–
AppAgent	Qwen3.8-27B	–	–	46.12	–	80.46	–
SE-GA	Qwen2.5-VL-7B	✓	✓	39.22	–	40.23	–
Agent S2	Qwen2.5-VL-7B	–	✓	25.86	–	49.43	–
Agent S2	Qwen3.8-27B	–	✓	53.02	–	90.80	–
Mobile-Agent-E	Qwen2.5-VL-7B	–	✓	31.90	–	50.57	–
Mobile-Agent-E	Qwen3.8-27B	–	✓	60.78	–	91.95	–
ATLASACT-7B	Qwen2.5-VL-7B	✓	–	37.07	10.07	58.62	11.03
ATLASACT-7B + Mem	Qwen2.5-VL-7B	✓	✓	**42.24**	**7.98**	**63.22**	**7.27**
ATLASACT-27B	Qwen3.8-27B	–	–	59.91	9.34	88.51	7.70
ATLASACT-27B + Mem	Qwen3.8-27B	–	✓	**67.67**	**7.55**	**95.40**	**7.12**

5.2.2 END-TO-END TASK EXECUTION

We next evaluate ATLASACT on end-to-end task execution in AndroidWorld and AndroidDaily. The memory and no-memory variants use the same grounding policy, allowing us to isolate the contribution of cross-task interaction memory while keeping grounding unchanged.

Task-level performance. As shown in Table 3, ATLASACT achieves competitive task-level performance even without cross-task memory. At 7B, it obtains 37.07% SR on AndroidWorld and 58.62% on AndroidDaily, compared with 31.90% and 27.59%, respectively, for UI-TARS-1.5-7B. At 27B, the no-memory configuration reaches 59.91% and 88.51% SR, compared with 54.31% and 79.31% for direct-action Qwen3.8-27B. These results show that the gains from the proposed grounding framework extend from action-level evaluation to end-to-end task execution.

Effect of interaction memory. Adding interaction memory consistently improves SR across both model scales and benchmarks. At 7B, SR increases by 5.17 percentage points on AndroidWorld and 4.60 points on AndroidDaily. At 27B, the corresponding gains are 7.76 and 6.89 points.

ATLASACT also achieves higher SR than the evaluated memory-enabled baselines. At 7B, the full configuration reaches 42.24% and 63.22% SR on AndroidWorld and AndroidDaily, respectively. At 27B, ATLASACT reaches 67.67% and 95.40%, exceeding Mobile-Agent-E, the strongest evaluated memory-enabled baseline at this model scale, by 6.89 and 3.45 percentage points, respectively. The full 27B configuration attains the highest SR among the evaluated methods on both benchmarks.

Within ATLASACT, interaction memory also reduces the average number of decision steps. At 7B, Avg. Steps decreases from 10.07 to 7.98 on AndroidWorld and from 11.03 to 7.27 on AndroidDaily. At 27B, the corresponding values decrease from 9.34 to 7.55 and from 7.70 to 7.12, respectively. These reductions indicate that reusable interaction memory can reduce repeated Actor decisions while improving task success.

Confidence-based VLM invocation. The skip gate directly executes high-confidence Top-1 candidates, reducing the need for Grounder VLM calls. Across the threshold sweep, lowering the invocation threshold reduces Grounder-call frequency, while its effect on EM varies across benchmarks.

Table 4: **Interaction memory ablations on AndroidWorld.** All variants use the ATLASACT-27B grounding configuration. A checkmark indicates that the component is enabled. SR is evaluated over the full evaluation set. Avg. Steps denotes the average number of decision steps on the fixed subset of tasks successfully completed by the no-memory configuration.

Variant	Tips	Shortcuts	Checks	SR (%)↑	Avg. Steps↓
No memory	–	–	–	59.91	9.88
Tips only	✓	–	–	66.81	9.06
Shortcuts w/o checks	–	✓	–	62.50	8.93
Shortcuts with checks	–	✓	✓	63.36	8.46
Tips + Shortcuts w/o checks	✓	✓	–	65.09	8.54
Full memory	✓	✓	✓	**67.67**	**8.12**

This yields a configurable trade-off between grounding accuracy and VLM invocation frequency. Full results are provided in Appendix B.

5.3 ABLATION STUDIES

We analyze retrieval-constrained grounding through matched-backbone comparisons and interaction memory through six configurations of ATLASACT-27B on AndroidWorld, with the grounding configuration held fixed (Table 4).

Under the same Qwen3.8-27B backbone and without additional VLM fine-tuning, ATLASACT achieves higher EM than both direct-coordinate prediction and dense SoM under the single- and multi-path SMAN-Bench settings (Tables 1 and 2). These matched-backbone comparisons help isolate the effect of retrieval-constrained grounding from changes in the underlying VLM.

Tips and Shortcuts play complementary roles in interaction memory. Using Tips alone increases SR from 59.91% to 66.81%, showing that retrieved experience provides useful decision guidance for subsequent tasks. Building on this, adding structurally checked Shortcuts reduces Avg. Steps from 9.06 to 8.12, a 10.4% reduction, while further increasing SR to 67.67%. This suggests that Tips primarily provide task-level decision guidance, whereas Shortcuts offer a more efficient mechanism for reusing previously learned action sequences.

However, the benefit of Shortcuts depends on reliable reuse. With Tips enabled, adding Shortcuts without structural checks lowers SR from 66.81% to 65.09%. Enabling checks raises SR to 67.67% and reduces Avg. Steps from 8.54 to 8.12. The same trend holds without Tips: structural checks increase SR from 62.50% to 63.36% and reduce Avg. Steps from 8.93 to 8.46. These results suggest that structural validation suppresses unreliable Shortcut reuse, allowing reusable action sequences to provide more consistent end-to-end benefits.

6 CONCLUSION AND FUTURE WORK

We presented ATLASACT, a mobile GUI agent framework that uses accessibility structure as a shared executable interface for action grounding and interaction memory reuse. Retrieval-constrained grounding addresses the gap between semantic target understanding and precise localization by separating target specification from executable target resolution, allowing the Actor to focus on semantic reasoning. Structure-validated memory reduces repeated model decisions through reusable Shortcuts, while structural checks mitigate failures caused by incompatible interface states. Across four mobile GUI benchmarks, ATLASACT improves both grounding and end-to-end task execution, while memory ablations show that structural validation makes Shortcut reuse more reliable. Overall, the results demonstrate the benefit of combining the semantic reasoning capabilities of multimodal models with explicit GUI structure for precise and reusable interaction.

Future work will explore more flexible skill-level reuse of interaction experience to improve GUI execution efficiency, together with more targeted retrieval and integration of relevant experience for different task requirements.

AI USE STATEMENT

Generative AI tools were used solely to assist with language editing and to improve the clarity and readability of the manuscript.

REFERENCES

Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. Agent S2: A compositional generalist-specialist framework for computer use agents. In *Conference on Language Modeling*, 2025.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Ming-Hsuan Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *CoRR*, abs/2502.13923, 2025.

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 (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

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

Chao Hao, Shuai Wang, and Kaiwen Zhou. Uncertainty-aware GUI agent: Adaptive perception through component recommendation and human-in-the-loop refinement. *CoRR*, abs/2508.04025, 2025.

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. In *The Tenth International Conference on Learning Representations*, 2022.

Shilong Jin, Lanjun Wang, and Zhuosheng Zhang. SE-GA: Memory-augmented self-evolution for GUI agents. In *Forty-third International Conference on Machine Learning*, 2026.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*, pp. 3146–3154, 2017.

Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibo Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Qwen3-VL-Embedding and Qwen3-VL-Reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *CoRR*, abs/2601.04720, 2026a.

Runze Li, Yuwen Zhai, Bo Xu, Liwu Xu, Nian Shi, Wei Zhang, Ran Lin, and Liang Wang. Echotrail-gui: Building actionable memory for gui agents via critic-guided self-exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Findings*, pp. 9347–9356, 2026b.

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 *Advances in Neural Information Processing Systems 37*, 2024.

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

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

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, 2026. URL `https://qwen.ai/blog?id=qwen3.8`.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

Yifan Sui, Xin Huang, Hongbing Li, Fang Xu, Jiahe Lv, Haolong Yan, Yeqing Shen, Litao Liu, Zhimin Fan, Ziyang Meng, Jia Wang, Junbo Qi, Kaijun Tan, Zheng Ge, Xiangyu Zhang, Daxin Jiang, and Osamu Yoshie. Androiddaily: A verifiable benchmark for mobile gui agents on real-world closed-source applications. *CoRR*, abs/2605.27761, 2026.

Libo Sun, Jiwen Zhang, Siyuan Wang, and Zhongyu Wei. MAGNET: Towards adaptive GUI agents with memory-driven knowledge evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28181–28206, 2026.

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. *CoRR*, abs/2401.16158, 2024.

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. In *Workshop on Scaling Environments for Agents*, 2025.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. OS-ATLAS: A foundation action model for generalist GUI agents. *CoRR*, abs/2410.23218, 2024.

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. *CoRR*, abs/2602.05832, 2026.

Weikai Xu, Zhizheng Jiang, Yuxuan Liu, Pengzhi Gao, Wei Liu, Jian Luan, Yunxin Liu, Yuanchun Li, Bin Wang, and Bo An. SMAN-bench: A cross-system benchmark for mobile agents under single- and multi-path, ambiguous, and noisy tasks. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 69772–69805, 2025.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. *CoRR*, abs/2310.11441, 2023.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile UI understanding with multimodal llms. In *Computer Vision - ECCV 2024 - 18th European Conference*, volume 15122 of *Lecture Notes in Computer Science*, pp. 240–255, 2024.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 70:1–70:20, 2025.

Jiayuan Zhang, Kaiquan Chen, Zhihao Lu, Enshen Zhou, Qian Yu, and Jing Zhang. Prune4web: DOM tree pruning programming for web agent. In *Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence*, pp. 34710–34718, 2026.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*, volume 235, pp. 61349–61385, 2024.

A IMPLEMENTATION AND TRAINING DETAILS

All hyperparameters used in the main experiments, including $K = 5$, $\tau = 0.99$, $\gamma = 0.6$, and the same-app Tip cap of 5, were fixed before the final benchmark evaluations. Appendices B and C report post-hoc sensitivity analyses.

A.1 MODELS AND ADAPTATION

The 7B system fine-tunes both the Actor and the Grounder with LoRA Hu et al. (2022). The 27B system uses Qwen3.8-27B for both roles with no additional fine-tuning. Qwen3-VL-Embedding Li et al. (2026a) is used only for retrieval and is not fine-tuned.

Both adapters use the same LoRA configuration and optimization setup. LoRA is applied only to the language-model projections. The vision tower is frozen. The image budget is the training-script default, `max_pixels` $= 200704$ ($256 \times 28 \times 28$). The run logs do not record an override. The Actor base is UI-TARS-1.5-7B Qin et al. (2025) and the Grounder base is Qwen2.5-VL-7B-Instruct Bai et al. (2025). The 20,000 Actor examples and the 41,714 Grounder click and long-press examples are taken from the official AndroidControl training split Li et al. (2024). They are episode-disjoint from the 200 validation and test trajectories used for AC-low and AC-high. The learning rate uses a cosine schedule.

Table 5: LoRA adaptation for the 7B Actor and Grounder.

	Actor	Grounder
Base model	UI-TARS-1.5-7B	Qwen2.5-VL-7B-Instruct
Examples	20,000 official training-split steps	41,714 official training-split click and long-press steps
Epochs	1	1
Global batch size	32	16
LoRA rank / α / dropout	16 / 32 / 0.05	16 / 32 / 0.05
Target modules	`q,k,v,o,gate,up,down_proj`	same
Trainable parameters	47,589,376 / 8,339,756,032 (0.57%)	same
Learning rate	10^{-4}, cosine, warmup 0.03	same
Precision	bf16	bf16

Actor. The Actor is trained on the current screenshot plus a text context of the task goal, the current step, and earlier steps. History screenshots are not included. The supervision target is a two-line response, not a coordinate or a node index:

Actor training target

```
Thought: Tap on the search option at the bottom of the screen
Action: click(target_object='Search')
```

`click` and `long_press` must name a `target_object`. Those strings are generated and then filtered, not written by hand. In this AndroidControl data, `type`, `scroll`, `press_back`, `press_home`, and `wait` do not carry a target. These are the dataset action names. At inference the Actor uses the action space in Appendix A.3, and the parser accepts the aliases: `long_press` is `long_click`, `input_text` is `type`, and `navigate_back` is `press_back`. `wait` occurs in the training counts but is not in that inference action space. In prose, the gesture is called a long-press. This adaptation teaches the Actor to name a target. It is not the grounding-model fine-tune reported in the main tables.

Grounder. The Grounder is trained to read a screenshot on which five colored boxes have been drawn and to answer with a single index. One box is the ground-truth node. The other four are embedding hard negatives. Their order is shuffled, so the label is not always 1. Steps with fewer

than five nodes, or without a ground-truth box or retrieval negatives, are dropped. The training prompt is:

Grounder training prompt

```
Instruction: "Click on the three bar menu button"

The screenshot shows a mobile UI. Five numbered regions (1–5) are
  marked with colored boxes. Which numbered box contains the UI
  element the user wants to interact with?
Reply with only the number (1, 2, 3, 4, or 5).
```

The label is that number. The model is not trained to emit coordinates. At inference the Grounder uses the same form: it answers with the bare index of a numbered box, 1–K, and the code maps that index back to the node. The training prompt above is the supervision used for the LoRA checkpoint. It is the fine-tune counted in the grounding-model column of the main tables.

A.2 SKIP ROUTER

The skip router is a binary LightGBM classifier Ke et al. (2017) that estimates whether the top-retrieved executable node is already the ground-truth target. It is used only on pointer steps (defined in Appendix B). Its input is a 14-dimensional summary of the current retrieval-score ranking. The classifier outputs

$$p_{\text{rank1}} = P(\text{gt_rank} = 1 \mid \phi_t). \tag{8}$$

If $p_{\text{rank1}} \geq 0.99$, ATLASACT executes the Top-1 node and does not call the Grounder. Otherwise it passes a fixed Top-5 candidate set to the Grounder. The threshold 0.99 is an inference choice, not a quantity learned by the classifier. It was fixed before the final benchmark evaluations. Appendix B is a post-hoc sweep and was not used to select it. At inference the router sees only ϕ_t. The ground-truth rank is used solely to construct offline training labels.

Features. All features are computed after sorting executable nodes by cosine similarity. They do not include the screenshot, the instruction, the target-object string, or the identity of the model that produced the target object. $s_1 \geq \cdots \geq s_5$ are the five largest scores, with missing ranks filled by zero when a screen has fewer than five nodes. n is the number of executable nodes on the screen.

Table 6: Skip-router features. All are functions of the retrieval scores.

Feature	Definition
$s_1, \ldots, s_5$	Top-5 retrieval scores
$\text{gap}_{12}, \ldots, \text{gap}_{45}$	Adjacent gaps $s_i - s_{i+1}$
top5_entropy	Softmax entropy of the available Top-5 scores
top5_std	Standard deviation of the available Top-5 scores
$s_1/(s_2 + \epsilon)$	Ratio of the first and second scores, $\epsilon = 10^{-6}$
$s_1 - \text{mean}(s_{1:5})$	Margin of the top score over the Top-5 mean
$\log(1 + n)$	Log number of executable nodes

Data and label. Training examples are single-step click screens from AndroidControl Li et al. (2024), identified by {episode}_{step}. This set is a separate 500-screen corpus. It is not the official AndroidControl-low or AndroidControl-high test split used in the main evaluation.

For each screen, six language models each produce one target-object description: DeepSeek-V4-Flash, Doubao-Seed-1.6-Flash, GPT-4o-mini, Qwen3-8B Qwen Team (2025), Qwen3.5-Flash, and Qwen3.7-Plus. Each description is embedded with Qwen3-VL-Embedding Li et al. (2026a), the same embedding model used at inference, and scored against the screen's node embeddings. This yields $500 \times 6 = 3000$ retrieval rankings. The model identity is stored with the row but is not a classifier input.

The ground-truth click is matched to the five accessibility nodes whose boxes are nearest to that point (`nearest_5`). gt_rank is the best rank of any of these nodes in the retrieval order. The training label is $y = 1$ if and only if $\mathrm{gt_rank} = 1$. Of the 3000 rankings, 2327 are positive.

Rows are split by screen, so the six rankings of one screen stay in the same fold. With a 20% held-out fraction and seed 42, this gives 2400 training rankings (400 screens) and 600 held-out rankings (100 screens). The held-out set contains 428 positives and 172 negatives.

Training. The classifier is `LGBMClassifier` with binary objective, 31 leaves, at least 20 examples per leaf, learning rate 0.05, 300 trees, and balanced class weight. On the 600 held-out rankings, ROC-AUC is 0.867, so the 14 score features separate rankings whose Top-1 node matches the ground truth from those that do not. This is a ranking metric on the training distribution. It is not Exact Match, and it is not computed at the inference threshold 0.99. How that threshold trades Grounder calls against grounding accuracy is reported in Appendix B.

A.3 PROMPTS

The strings below are the prompts sent by the submitted ATLASACT code. Braces mark values filled at runtime; they are not literal characters in the template. The Actor, the set-of-marks Grounder, and the raw-screenshot coordinate grounder all use the system message "You are a helpful assistant." The instructions themselves are in the user message. All prompts in this section are the AndroidWorld versions.

The set-of-marks Grounder below asks for the same bare index as the training prompt above. At inference the index is 1–K for the boxes actually drawn, and the code maps that index back to the node.

Actor with memory. This is the default Actor user message (`enable_memory=True`). With memory disabled, the shortcut note, the Tips block, and the Shortcuts block are omitted; the rest of the message is the same. The fixed prefix begins:

Actor prefix

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

## Output Format

Thought: ...
Action: ...

## Action Space
click(target_object='')
long_click(target_object='')
scroll(direction='up or down or left or right')
scroll(start_box='(x1,y1)', end_box='(x2,y2)')
drag(start_box='(x1,y1)', end_box='(x2,y2)')
type(content='')
keyboard_enter()
press_back()
press_home()
open_app(app_name='')
finished(answer='')

## Note
```

```
- Use English in the Thought part.
- Summarize this step (action + target) in one sentence in Thought.
- click / long_click: if tapping a visible named control, MUST set
  target_object to the EXACT on-screen text (or a short visual
  description if there is no text). Do NOT use start_box,
  coordinates, or node_id for click retrieval.
- Blank / overlay / dimmer / no named control:
  click(target_object='') or long_click(target_object='') --- leave
  target_object empty. Describe the location in Thought. Do NOT
  invent labels such as "empty area" or "wallpaper".
- type / scroll / drag / keyboard_enter / press_back / press_home /
  open_app / finished: do NOT set target_object.
- scroll full-screen: set direction to the FINGER movement (NOT
  content direction). Want content BELOW → direction='up'. Want
  content ABOVE → direction='down'. Want content to the RIGHT →
  direction='left'. Want content to the LEFT → direction='right'.
- Scoped scroll (one list/column, or when the keyboard is visible):
  use start_box='(x1,y1)' and end_box='(x2,y2)' with parentheses
  and UITARS integers 0-1000. Never full-screen direction scroll
  into the keyboard.
- drag: long-press then drag (sliders, brightness, volume). MUST set
  start_box='(x1,y1)', end_box='(x2,y2)' with parentheses. Integers
  in 0-1000 only --- do not omit (), do not mix 0-1 floats with
  0-1000, do not use pixel values. Do NOT use scroll for slider
  thumbs.
- type: MUST set content to the exact string to type.
- open_app: MUST set app_name to an Android World name, not a
  package and not a nickname. On launcher/home: open_app only ---
  do not scroll to reveal apps.
- finished: when the goal is already achieved, output finished()
  immediately. When a result must be submitted, put only the final
  value in answer=; otherwise leave answer empty.
- Coordinates in start_box / end_box are UITARS 0-1000 (not 0-1, not
  pixels).

TARGET_OBJECT (click / long_click only --- used for TopK UI
  retrieval):
- If the control shows visible text, copy that EXACT on-screen
  wording. Do not paraphrase it.
- Only if the control has NO visible text, use a short visual
  description (e.g. "back arrow", "shutter button").
- Keep it minimal: usually 1-4 words. Do NOT add location,
  dialog/page context, or role words.
- Put context (which dialog, why tap) in Thought --- NOT in
  target_object.
```

The same prefix then compares each earlier step's instruction and action with the current screenshot, tells the Actor to output `finished()` as soon as the goal is already met, and to change plan when the screen does not advance. A typing task should focus the field before `type()`. The user message then gives the goal, the step as {`step`} / {`max_steps`}, up to eight earlier-step lines (`Target` when a click names a control, `Coordinate` after a coordinate tap), and, when history exists, repeats the last step. The screenshot line is the raw, unannotated frame. If any history exists, the message adds a one-line completion check; if at least two steps exist, it also adds a course-check

line. Every call ends with "Plan this step: if the goal is already done, output finished(); otherwise Thought then one Action."

With memory enabled, that prefix is followed by a shortcut note. It adds `invoke_shortcut(shortcut_name='')` and allows it only when the current screen meets the shortcut's precondition, the name matches the list exactly, and slot values are left empty for a later fill. The entry control must already be visible. Tips are reference only and do not override a matching shortcut. Retrieved tips and admitted shortcuts are inserted after the goal; each shortcut is one bullet with its parameters and precondition.

Set-of-Marks Grounder. The Grounder prompt follows Set-of-Marks Yang et al. (2023). Coordinates in the action space use the UI-TARS integer range Qin et al. (2025). This is the Grounder for a named `click` or `long_click` when the interactive-node pool is non-empty and the skip router does not fire. Up to five boxes are numbered 1–K in retrieval order, so 1 is the top retrieved box in that set. The step instruction is the Actor's step instruction for this step. If that is empty, the code uses "Click on" or "Long-press" plus the target object, or "the target" when the target object is empty too. The model replies with only the number. The code lists the integers from 1 to K, with "or" before the last number when $K \geq 2$; a single box is just "1". The code maps that number back to the node.

Set-of-Marks Grounder

```
Instruction: "{step_instruction}"

The screenshot shows a mobile UI. {K} numbered regions (1--{K}) are
   marked with colored boxes. Which numbered box contains the UI
   element the user wants to interact with?
Reply with only the number ({numbers}).
```

Raw-screenshot coordinate grounder. This is the fallback when `target_object` is empty, or when a target is set but the interactive-node pool is still empty after one recapture. It is run by the Grounder. The screenshot is raw. The action space contains only the action the Actor already chose: `click` or `long_click`. When `target_object` is empty, the target line is "No named control. Tap the location described by the step instruction." Otherwise it is "Retrieval target:" followed by the target object.

Coordinate grounder

```
You are a GUI agent. Tap a location on the RAW unannotated
   screenshot. The action type is already decided.

## Output Format

Thought: ...
Action: ...

## Action Space
{action}(start_box='<|box_start|>(x,y)<|box_end|>')

## Note
- Use English in the Thought part.
- The screenshot is RAW and unannotated (no numbered boxes). Do NOT
   look for #labels.
- Coordinates are UITARS integers in 0-1000. (0,0)=top-left,
   (1000,1000)=bottom-right. NOT normalized 0-1, NOT pixels.
```

```
- Output exactly this Action, with start_box.
- Do NOT output target_object, node_id, or finished().

## User Instruction
Task Goal: {goal}
Step instruction: {step_instruction}
{target_line}

## Current screenshot --- raw, unannotated (no numbered boxes)
Output {action}(start_box='<|box_start|>(x,y)<|box_end|>') with
   UITARS 0-1000 coordinates.
```

If the step instruction is empty, the code writes "(tap the intended control)".

Memory side calls. These calls are not part of the step loop. After `invoke_shortcut`, a separate model fills declared slots as one JSON object of strings. After scoring, the same memory model rewrites the tip list for the current bucket and proposes at most three shortcut lifecycle operations (`create`, `merge`, `replace`, `repair`, `deprecate`, or `noop`). The full strings are in the submitted code.

A.4 SHORTCUT ENTRY AND REPLAY CHECKS

Tips and Shortcuts follow the memory types introduced by Mobile-Agent-E Wang et al. (2025). The checks below are the A11y conditions this paper adds on top of that reuse.

The main text admits a Shortcut when the current application matches and the structural coverage of its entry signature is at least γ. This section states which accessibility fields enter Φ, which set is the denominator of that ratio, and what each replay step checks before and after it executes.

What Φ stores. The entry signature is taken from the accessibility tree of the screen where the Shortcut starts. The application name pkg is the package that appears on the most nodes in that tree, counting every node that names a package. Φ itself is a set of fingerprints of interactive nodes only. A node is interactive when it is clickable, focusable, or scrollable, or when its class name contains `edittext`. Nodes of `com.android.systemui` whose bottom edge lies at or above 200 pixels are treated as the status bar and are omitted, so the clock does not enter every signature.

Each remaining node contributes at most one string. If it has a resource id, the fingerprint is `rid:` followed by that id. Otherwise, if it has visible text, the fingerprint is `text:` followed by that text; if it has only a content description, the fingerprint is `desc:` followed by that description. A node with neither a resource id nor a label contributes nothing.

Before the signature is saved, task-varying literals are removed: the typed text and the app name of each step in the Shortcut. Only `text:` and `desc:` entries are removed this way; a resource id is kept. The remaining strings are ordered with resource ids first and truncated to eight. The current-screen fingerprint Φ_t is built by the same rules, but it is not truncated and task-varying literals are not removed from it.

What the coverage compares. Let Φ be the stored list and Φ_t the fingerprint set of the screen being tested. Coverage is the fraction of stored entries that occur verbatim in the current set,

$$\mathrm{cov}(\Phi, \Phi_t) = \frac{|\Phi \cap \Phi_t|}{|\Phi|}. \tag{9}$$

The denominator is the number of fingerprints saved with the Shortcut, not the number of fingerprints on the current screen, and not the size of the union. The package test runs first, and only when both sides name a package: if they differ, the Shortcut is rejected and the ratio is not used. If either package is missing, that test does not reject. After it, an empty stored list passes. Otherwise the Shortcut is admitted when $\mathrm{cov}(\Phi, \Phi_t) \geq \gamma$. The main experiments use $\gamma = 0.6$, fixed before

evaluation, so a coverage strictly below 0.6 fails. Admission only makes the Shortcut available to the Actor; it does not invoke it.

Per-step checks during replay. Once a Shortcut is invoked, each atomic step is checked against the accessibility tree, then executed, then checked again. A failed pre-check aborts replay before the action runs. For `click` and `long_click` the pre-check is a re-resolution: the step must name a resource id or a target object, and that target must match exactly one node in the current tree. If it does not, replay aborts and the Shortcut is disabled for the rest of the episode. A strong post-check that fails does the same. A weak post-check that fails is recorded as `suspect` and replay continues. The post-check is polled for up to three captures, 0.8 s apart. A step that satisfies the check on the first capture returns `ok`; one that satisfies it after polling returns `slow`.

For other action types the pre-check is narrower. `input_text` requires an enabled node that is focused, focusable, or clickable, and whose class contains `edittext`, `autocompletetextview`, or `multiautocompletetextview`. `scroll` and `swipe` require some scrollable node, unless the step has a direction and no resource id, in which case the check passes. Every other action type passes the pre-check. If the accessibility file is missing, `input_text`, `scroll`, and `swipe` also pass.

The post-check depends on the action type. For `input_text`, the typed string must occur in the text of an `EditText`; that failure is strong. An empty string passes. A password field is only checked for the continued presence of an `EditText`, and that failure is weak, because the typed text is not readable. For `open_app` and `navigate_home`, the current package must equal the package recorded with the step; that failure is strong. If no package was recorded, the check passes and is weak. For `click` and `long_click`, when a checked value was recorded from the screen after the original click, the same node must be found again by resource id or label and its checked value must equal that recording. A mismatch, or a missing node, is a strong failure. `wait` has no assertion. When none of these apply, the check asks only whether the screen signature changed: the two packages differ, or the Jaccard distance between the fingerprint sets is greater than 0.2. This fallback is weak, so an unchanged screen is `suspect` and replay continues.

B Accuracy–Invocation Trade-off of the Skip Gate

This appendix characterizes the trade-off between Grounder invocation frequency and grounding accuracy as τ varies. It is a post-hoc analysis and was not used to select the main configuration. On each pointer step, the fast path executes the Top-1 retrieved node when $p_{\text{rank1}} \geq \tau$; otherwise, the Grounder selects among the Top-K candidates. We fix $K = 5$ and vary τ over 22 values from 0 to 0.99. The main experiments use $\tau = 0.99$, which was fixed before evaluation.

A pointer step is an action whose target is resolved by retrieval-constrained grounding. On AndroidControl these are click and long-press. On SMAN-Bench they are click and scroll; long-press does not occur. Each SMAN action id is bound to a node, and a scroll also carries a direction, so the scroll target is resolved by the same retrieval and router as a click. On SMAN-Bench, a scroll also carries a `target_object`: the Actor names the region or list to scroll, and that string is the retrieval query (Appendix A.3). The Actor also outputs the direction, whether or not the Grounder is called. A skipped scroll is correct only when both the retrieved node and the direction match the gold action. On AndroidControl, AndroidWorld, and AndroidDaily, scroll is executed from the direction alone and does not enter the router. Grounder invocation is the fraction of pointer steps that require Grounder inference. Exact Match (EM) is computed over all evaluable steps, the same denominator as Tables 1 and 2. At $\tau = 0.99$, the reported EM therefore matches the corresponding main-table results. Both SMAN panels, simple and complex, use the single-path setting, not the multi-path columns of Table 2.

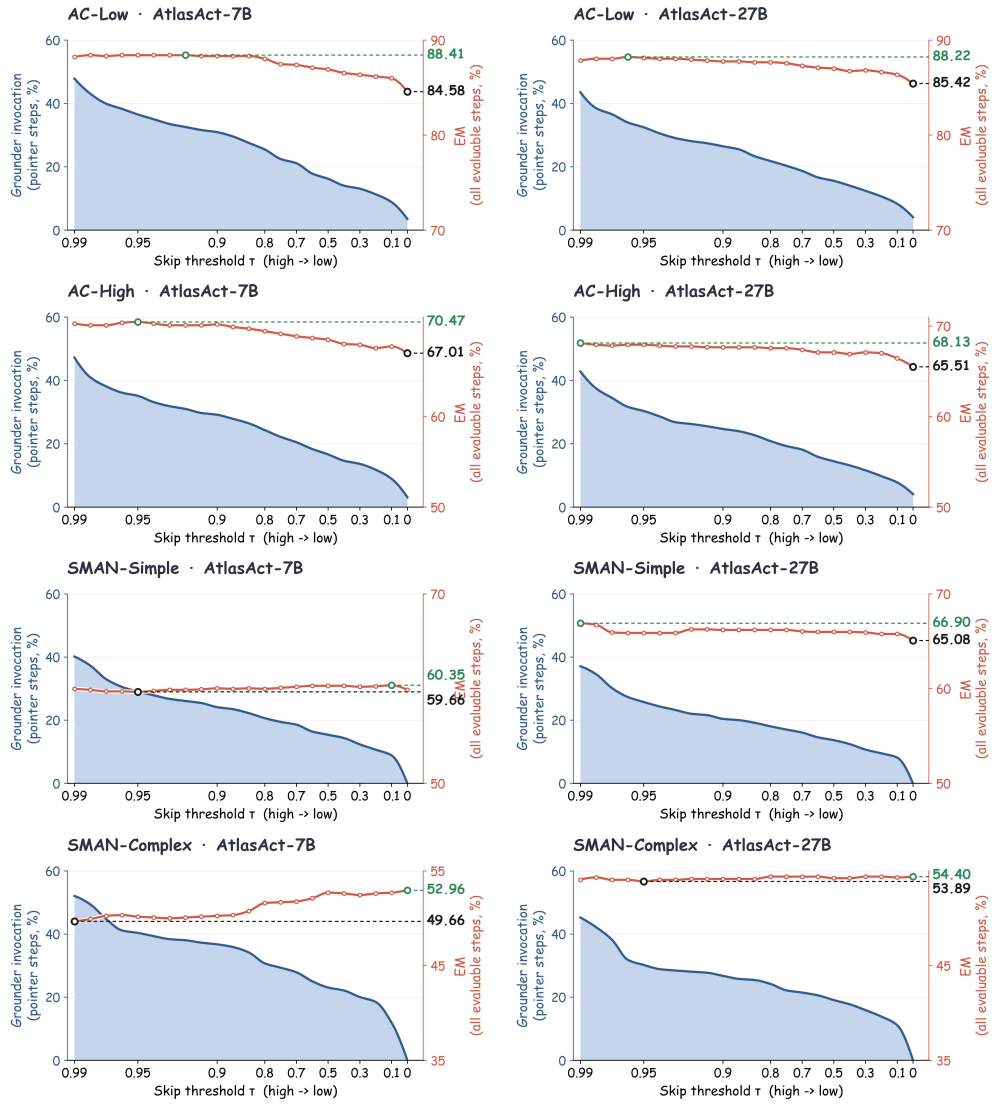

Figure 3: Skip-gate sweep at $K = 5$ for all eight settings, in four rows of two panels. Top to bottom, the rows are AC-low, AC-high, SMAN-simple, and SMAN-complex. In each row, 7B is on the left and 27B is on the right. The SMAN panels use the single-path setting. In every panel the horizontal axis is τ, decreasing from 0.99 to 0. The left axis is the Grounder invocation rate over pointer steps. The right axis is Exact Match over all evaluable steps, the same denominator as Tables 1 and 2. Markers indicate the minimum and maximum EM in that panel.

At $\tau = 0.99$, the Grounder is invoked on 37.1%–52.1% of pointer steps. At $\tau = 0$, invocation falls to 3.1%–4.1% on AndroidControl and to 0% on both SMAN splits. The residual AndroidControl calls occur when the A11y tree contains no interactive node, leaving no Top-1 candidate. In these cases, the system falls back to raw-screenshot coordinate grounding.

From $\tau = 0$ to $\tau = 0.99$, the change in EM stays within 3.6 points in every setting. EM rises by 2.4–3.6 points on AC-low and AC-high, and by 0.1 and 1.8 points on SMAN-Simple for 7B and 27B, while on SMAN-Complex it falls by 3.3 points at 7B and by 0.3 points at 27B. At $\tau = 0$, nearly every pointer step already executes the retrieved Top-1 node, so most of the EM at $\tau = 0.99$ is present before any additional Grounder call. Retrieval alone is therefore a strong low-cost grounding mode: a large fraction of pointer actions can use Top-1 directly. The router implements conditional

computation. It decides when an extra Grounder call is warranted; it is not itself the source of the grounding gain. The operating point $\tau = 0.99$ is a conservative choice fixed before evaluation. It is not the accuracy-optimal threshold on every split.

Type Match (TM) remains constant across all 22 thresholds because the gate changes only target selection, not the Actor-predicted action type. Skip accuracy on the directly executed subset is higher at $\tau = 0.99$ than at $\tau = 0$ in every setting, reflecting the more conservative fast path at higher thresholds. However, this does not translate monotonically into overall EM because Grounder-selected targets are not uniformly more accurate than retrieved Top-1 targets.

C Sensitivity to Shortcut Coverage and Tip Budget

AndroidWorld and AndroidDaily in the main paper use a fixed Shortcut coverage threshold $\gamma = 0.6$ and a same-app Tip cap of 5. Both values were fixed before the final benchmark evaluations. This section is a post-hoc sensitivity analysis and was not used to select them. Here, γ controls the structural-coverage threshold for admitting a Shortcut, while n denotes the maximum number of same-app Tips. Up to three global Tips may be added in addition to this cap, subject to the same similarity gate, so the Actor receives at most $n + 3$ Tips and often fewer. A cap of 0 disables same-app Tips but still allows global Tips, whereas ∞ removes both numerical caps. Figure 4 reports AndroidWorld success rates over this grid.

Overall, performance is more sensitive to the same-app Tip budget than to the Shortcut coverage threshold. A moderate Tip budget is sufficient, while removing the caps provides no consistent additional benefit. In contrast, performance remains relatively stable across a broad range of γ, indicating that the default setting lies on a broad performance plateau rather than a sharp optimum.

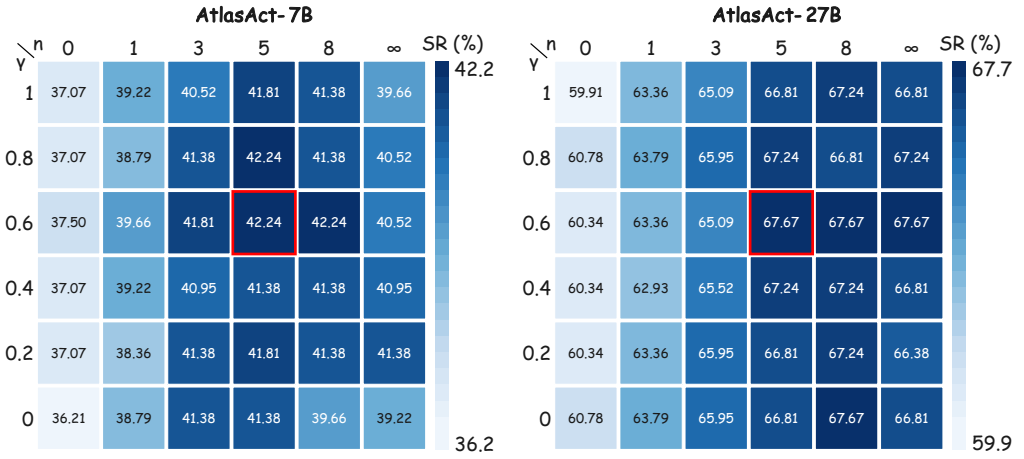

Figure 4: AndroidWorld success rate (%) across Shortcut coverage threshold γ and same-app Tip cap n. Up to three global Tips may be added in addition to the same-app cap, and ∞ removes both numerical caps. Each panel uses its own color scale. The red box marks the fixed configuration used in the main experiments ($\gamma = 0.6$, $n = 5$).

D A worked trajectory

Figure 5 follows one successful AndroidWorld episode Rawles et al. (2025), deleting the newest note in Markor. The Actor reads the raw screenshot. A pointer step takes the fast path when retrieval is confident, and the slow path when it is not.

Step 0 is `open_app`. It is not a pointer action, so the Grounder is not called. Step 1 is a long-press on a file list that contains several similar notes. Compact SoM marks the Top-5 candidates, and the Grounder selects one box. Steps 2 and 3 are the trash icon and the confirm button. Each target is

unambiguous, so both steps skip the Grounder and execute Top-1. Step 4 sees that the note is gone and the Actor outputs `finished()`.

The single box and click mark on Steps 2 and 3 are drawn for this figure only. A real skip does not render a Top-1 SoM image. No Grounder is called, and the retrieved Top-1 node is executed directly.

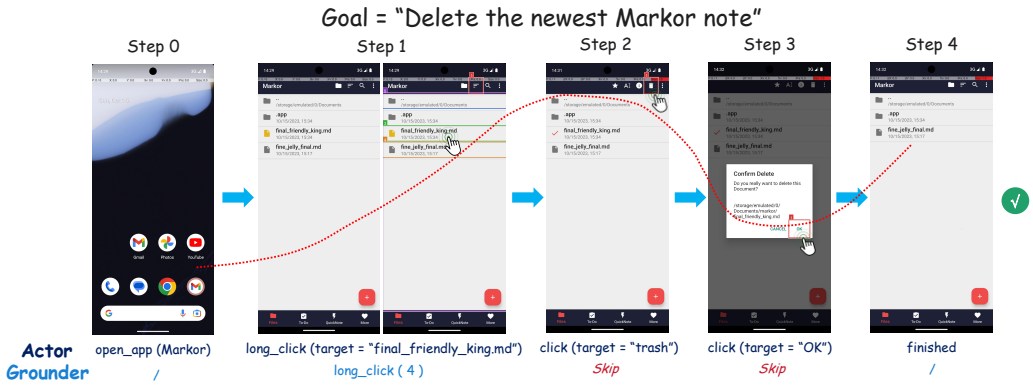

Figure 5: One episode of ATLASACT deleting the newest Markor note. Blue arrows are Actor actions. The red dotted arrow marks the Grounder call on the ambiguous long-press. Lightning marks are skipped pointer steps.

E FULL-PAGE SOM VERSUS COMPACT SOM

This section is a visual comparison, not a new accuracy table. Tables 1 and 2 already measure Exact Match when the Grounder chooses among every marked element (AppAgent-style dense SoM Zhang et al. (2025); Yang et al. (2023)) or among a retrieved subset (ATLASACT). Figure 6 shows why that difference is visible on the screen.

The figure uses the same four screens, taken from real AndroidControl and SMAN-Bench test cases. Panel (a) marks every interactive node, with click tags and scroll tags in the AppAgent style. On these screens that is tens of boxes. Panel (b) marks only the five nodes whose embeddings are closest to the step's target object. The query is that target string, embedded with the same Qwen3-VL-Embedding model used for node retrieval. No other candidates are drawn.

Panel (a) gives the Grounder a large set of look-alike controls, so the task-relevant box is one mark among many. On screens with small controls or dense text, those marks also cover content the model

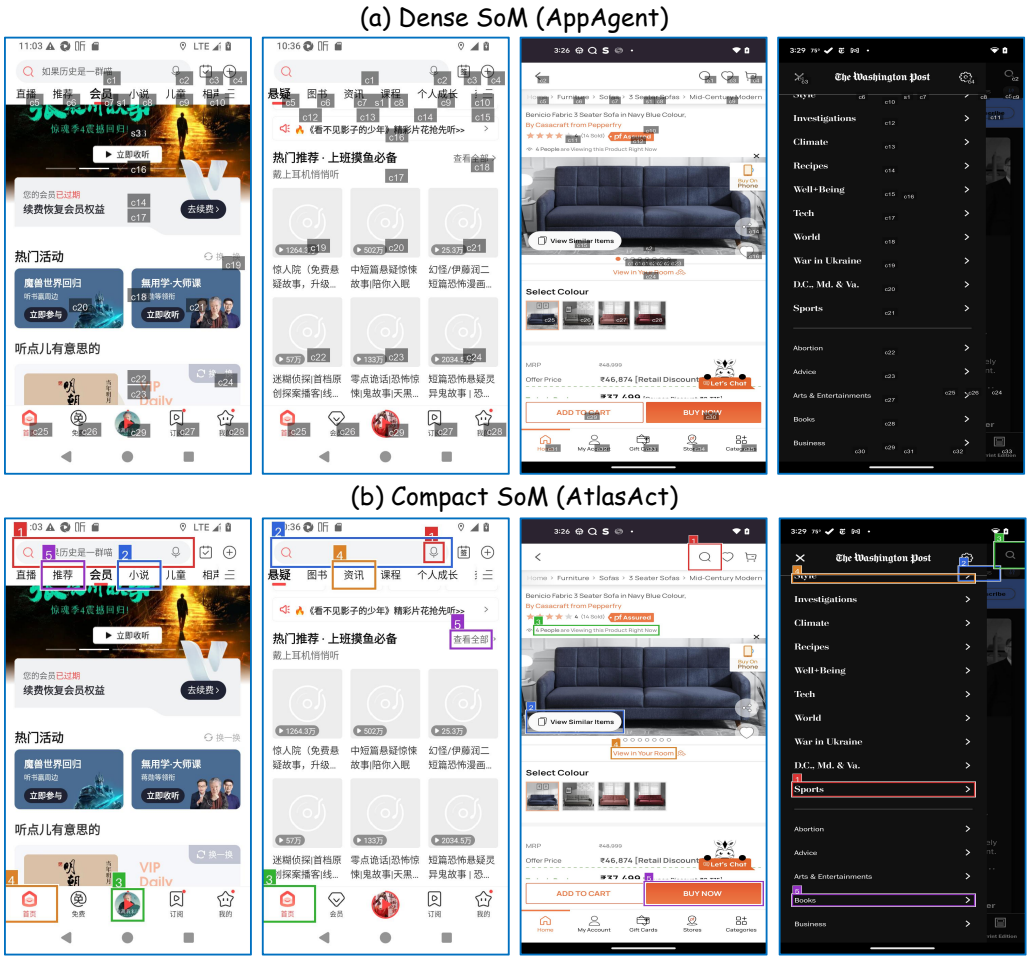

Figure 6: Full-page SoM versus compact SoM on the same four screens. (a) Dense SoM marks every interactive node. (b) Compact SoM marks only the retrieved Top-5 nodes, numbered 1–5.

would otherwise read. Panel (b) keeps the same screenshot and the same task, but leaves most of that text unmarked; the boxes that remain are the ones retrieval ranked for the target. The claim this figure supports is the one already stated with the grounding tables: the gain is not only that selection replaces coordinate regression, but that the visual decision is made over a small, task-conditioned set.

F TEXT IN THE NODE EMBEDDING

This section compares two node embeddings used for Top-K retrieval. Both use the same Qwen3-VL-Embedding model Li et al. (2026a). The query is the step's `target_object`. Text+patch embeds the node's accessibility text together with its screenshot crop. Patch-only embeds the crop alone. Every ATLASACT entry in the main tables uses text+patch. Patch-only is the comparison drawn here, not a reported system. The curves are retrieval hit rates, not the Exact Match in Tables 1 and 2. The Actor and the Grounder are not part of this measurement.

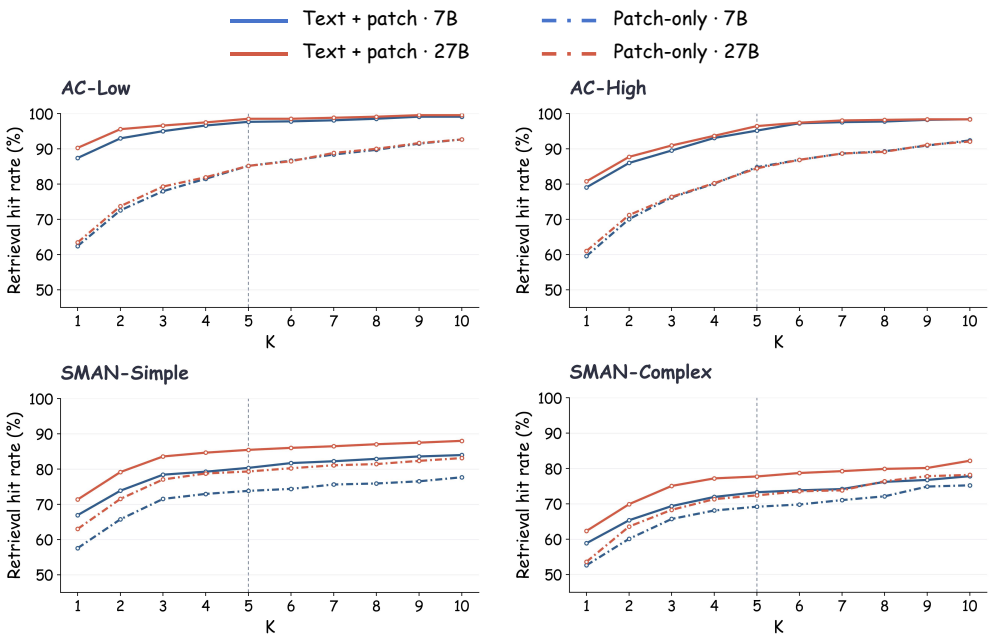

Figure 7: Retrieval hit rate versus K. Solid lines are text+patch. Dash-dot lines are patch-only. Blue is 7B and coral is 27B. The vertical line marks $K = 5$, the candidate set used in the main experiments. The top row is AndroidControl. A hit is scored with the same pointer rule as Android-Control Exact Match. The bottom row is SMAN-Bench single-path. A hit is the ground-truth node appearing at rank K or better. The two rows do not share a denominator.

On AndroidControl, text+patch is already at 95%–98% by $K = 5$. Patch-only is about 85% there, and its Top-10, about 92%–93%, still sits below the text+patch Top-5. On SMAN-single the same ordering holds, with a smaller gap: at $K = 5$, simple is 80.3%/85.5% against 73.8%/79.3%, and complex is 73.3%/77.7% against 69.2%/72.4%, for 7B and 27B. The two runs do not share the same `target_object` string, so each pair is one query set against its own node index.

The same contrast appears in Exact Match. Patch-only is the agent with the image-only node embedding. Text+patch is the ATLASACT row of Tables 1 and 2. The parenthetical value is text+patch minus patch-only. The single-step table has no Step Efficiency. The multi-step table reports it for SMAN only. Lower SE is better, so a negative ΔSE is an improvement. AC-high has no SE.

Table 7: Single-step Exact Match with and without text in the node embedding. Text+patch repeats Table 1.

	7B		27B	
Dataset	EM P-only	EM T+P	EM P-only	EM T+P
AC-low	74.21	**88.22 (+14.01)**	74.02	**87.85 (+13.83)**
SMAN-single-simple	51.88	**59.98 (+8.10)**	61.64	**66.90 (+5.26)**
SMAN-single-complex	44.08	**49.66 (+5.58)**	46.95	**54.06 (+7.11)**

Table 8: Multi-step Exact Match with and without text in the node embedding. Text+patch repeats Table 2. SE is lower-is-better. AC-high has no SE.

	7B				27B			
Dataset	EM P-only	EM T+P	SE P-only	SE T+P	EM P-only	EM T+P	SE P-only	SE T+P
AC-high	57.98	**70.28 (+12.30)**	–	–	55.14	**68.13 (+12.99)**	–	–
SMAN-multi-simple	41.68	**50.70 (+9.02)**	5.65	**5.28 (–0.37)**	55.31	**60.09 (+4.78)**	5.04	**4.80 (–0.24)**
SMAN-multi-complex	24.11	**35.87 (+11.76)**	4.72	**4.66 (–0.06)**	35.36	**43.65 (+8.29)**	4.60	**4.50 (–0.10)**

G BENCHMARK DETAILS

The main tables use four benchmarks under two evaluation regimes. AndroidControl Li et al. (2024) and SMAN-Bench Xu et al. (2026) are offline: each prediction is compared with a logged action. AndroidWorld Rawles et al. (2025) and AndroidDaily Sui et al. (2026) are online: a trial is scored by the benchmark's own check on a live device. Metric definitions remain in the main text. This section fixes the splits, the counts, and the matching rules that those definitions leave open.

Table 9: Evaluation splits used in the main tables.

Regime	Split	Tasks	Steps	What the model sees	Main-table metrics
Offline step	AC-low	200	1070	Step instruction and screenshot	Exact Match, Type Match
Offline step	AC-high	200	1070	Episode goal and screenshot	Exact Match, Type Match
Offline step	SMAN single-simple	611	1864	Step instruction and screenshot	Exact Match, Type Match
Offline step	SMAN single-complex	227	1182	Step instruction and screenshot	Exact Match, Type Match
Offline trajectory	SMAN multi-simple	611	1864	Step instruction, earlier step text, and screenshot	Exact Match, Step Efficiency
Offline trajectory	SMAN multi-complex	227	1182	Same as multi-simple; back is allowed	Exact Match, Step Efficiency
Online task	AndroidWorld	116	–	Task goal and live screenshot	Success rate
Online task	AndroidDaily	87	–	Task goal and live screenshot	Success rate

G.1 OFFLINE STEP MATCHING

AndroidControl. The 200 trajectories are taken from the official AndroidControl validation and test sets Li et al. (2024). Low and high are the same trajectories. They share the screenshot, the accessibility tree, the gold action, and the 1070-step denominator. Only the instruction changes. Under low, each step shows that step's instruction. Under high, each step shows the episode goal and does not show the current step instruction.

A trajectory has one high-level goal and an ordered list of steps. Each step has the current screenshot, that step's accessibility tree, a step instruction, and a gold action. The gold action has a type. Click and long-press also have a touch point, scroll has a direction, text entry has a string, and `open_app` has an application name.

The Exact Match denominator drops `open_app` steps and the finish or terminate step appended at the end of a trajectory. Low and high share the remaining 1070 steps. Step Efficiency is not computed on AndroidControl.

Exact Match first compares the action type. A type mismatch scores 0. After the types agree:

- Wait, navigate back, and navigate home match on type alone.
- For input text, the reported tables follow the official AgentCPM rule: after lowercasing and stripping whitespace, the prediction matches when either string contains the other.
- For scroll, the reported tables follow the official AgentCPM rule. The gold direction is the direction of the screen content and the prediction is the finger direction, and the two are opposite: content down matches a finger swipe up, content up matches down, content left matches right, and content right matches left.
- For click and long-press, the reported tables use a point-in-box rule. The predicted point matches if it falls inside the smallest accessibility box that contains the gold point, or inside any of the five boxes nearest that point. This rule replaces the AgentCPM evaluator used

when scoring UI-TARS on AndroidControl. That evaluator enlarges every box by a factor of 1.2, keeps the five boxes whose centers are nearest the gold point, and also accepts a hit when the normalized Euclidean distance between the predicted point and the gold point is at most 0.04.

SMAN-Bench. SMAN-Bench Xu et al. (2026) is four settings of the same Mobile3M screens, not four unrelated datasets. Simple and complex are the Common-Simple and Common-Complex lists in the main text, abbreviated there as SMAN-Simple and SMAN-Complex. They are disjoint task lists, split by the length of the gold trajectory: 2–4 steps in `simple_normal_tasks.json` (8909 tasks) and 5–10 steps in `complex_normal_tasks.json` (3745 tasks). The task names do not overlap. Single-path and multi-path are two ways of following one reference path. The path is the `name` field, for example `QQmusic0_14_294_343`. Its length is the number of underscore-separated segments. Single-path moves to the next gold screenshot even when the prediction is wrong, so the number of steps is the length of that path and there is no extra cap. Multi-path takes the next screenshot from the page that `all_triple` records for the predicted action, so the agent can leave the path. The reference remains that one path. It is not the shortest among several references. Simple stops at 20 rounds and complex at 25. Complex may go back.

We evaluate on the ten applications in the open SMAN-Bench release whose screen graphs are complete: ximalaya, BaiduMap, androidesk, ctrip, gaodeMap, kugou, ludashi, QQmusic, vipshop, and xiaomiShop. Bilibili (bili) is also open-sourced in the same release, but its graphs are incomplete, so we exclude it. QQmusic tasks dominate the open lists, so we do not run all of them: we keep the first 50 simple tasks and the first 25 complex tasks. Those prefixes follow the JSON order, which is also sorted by task name, so they are the first 50 and first 25 entries of the official lists.

Tasks with a missing page are dropped entirely, including every step that would otherwise have been scored. The evaluation uses 611 simple tasks and 1864 steps, and 227 complex tasks and 1182 steps. Exact Match uses click, scroll, and input. Long-press does not occur. A step whose gold type is outside click, scroll, and input, including back, is outside the denominator. The Grounder-invocation rate uses the pointer steps defined in Appendix B, so its denominator is smaller than this Exact Match denominator. A click matches when the predicted action id equals the gold action id. Scroll labels in the files are finger directions, and evaluation does not convert them to content direction. A different direction is a different action id.

One sample is one JSON object. The task field contains the step-by-step description and a final short task line. Screenshots, HTML, and accessibility XML live in the corresponding graph directory. The accessibility tree builds candidate nodes and maps a prediction to an action id. It is not given to the model. The model sees the screenshot. The Actor reads the current step of the step-by-step description, not the short task line. On multi-path it also sees the text of the earlier steps.

Each task reloads from its own root page. Tasks do not share device state, and there is no separate order file. The order is the order of the JSON file. Step Efficiency is computed only on multi-path, as the mean number of predicted rounds divided by the mean gold length.

G.2 Online task evaluation

AndroidWorld and AndroidDaily are full-task evaluations on a live device. A trial is scored by the benchmark's own check. The main text already states that memory-enabled runs start from an empty bank and follow the dataset order. This section fixes the task list, the instantiation, and that order.

AndroidWorld. The evaluation uses all 116 AndroidWorld task templates Rawles et al. (2025), with one parameterized instance of each. That instance is drawn with the official task random seed 30. Tasks are executed in application order. Each trial is scored by the official device-state check. The score is 0, 0.5, or 1: 0.5 occurs when a composite task meets only one of its two requirements. Reported SR is the mean of these scores.

AndroidDaily. The released AndroidDaily set contains 235 end-to-end tasks Sui et al. (2026). The evaluation uses 87 of them. The omitted tasks are those that would place an order, pay, send a WeChat message, or otherwise leave a real side effect. The retained tasks stop at search, lookup, or playback. Tasks are executed in application order. Success rate is assigned by a grader VLM

that reads the task instruction and the execution trajectory. The grader uses the official prompt. It outputs three binary scores: whether the required operations were carried out (obl), whether the final outcome meets the goal (qual), and whether the trajectory has no harmful side effect (neg). SR is 1 only when all three are 1, and 0 otherwise.