LearnAct: Few-Shot Mobile GUI Agent with a Unified Demonstration Benchmark

Guangyi Liu[1*] **Pengxiang Zhao**[1*] **Liang Liu**[2‡] **Zhiming Chen**[2] **Yuxiang Chai**[3]
Siheng Chen[4] **Yaozhen Liang**[1] **WenHao Wang**[1] **Zhengxi Lu**[1] **Shuai Ren**[2]
Hao Wang[2] **Shibo He**[1] **Yong Liu**[1] **Wenchao Meng**[1†]
[1]Zhejiang University [2]vivo AI Lab [3]CUHK MMLab [4]Shanghai Jiao Tong University
wmengzju@zju.edu.cn
[*]Equal contribution. [‡]Project Lead. [†]Corresponding author.

Abstract

Mobile GUI agents show promise in automating tasks but face significant generalization challenges in long-tail scenarios. While learning from few-shot demonstrations is an emerging solution, its progress is hindered by two critical gaps: the lack of a comprehensive benchmark for systematic evaluation on mobile devices, and the absence of a systematic framework designed to learn from demonstrations in this domain. To address these gaps, we introduce **LearnGUI**, the first comprehensive benchmark designed for studying demonstration-based learning in mobile agents, comprising 2,252 offline and 101 online tasks. We further develop **LearnAct**, a modular agent framework engineered to systematically extract, retrieve, and leverage knowledge from visual demonstrations. Extensive evaluations across six backbone models validate our approach: LearnAct achieves dramatic improvements for general-purpose models (e.g., Gemini-2.5-Pro: 38.5%→58.9%) and specialized models alike (e.g., UI-TARS-7B-SFT's online success rate: 18.1%→32.8%), demonstrating consistent gains across model architectures. Our work provides a robust benchmark and a systematic framework, paving the way for more adaptable and practical mobile agents. Our code and data are publicly available at https://lgy0404.github.io/LearnAct/.

1 Introduction

Mobile Graphical User Interface (GUI) agents, powered by Large Language Models (LLMs), have emerged as a powerful paradigm for automating device interactions (Wen et al., 2023, 2024; Zhang et al., 2024a; Lu et al., 2024b). By perceiving screen states and generating actions, these agents promise to revolutionize how humans use mobile devices. However, their widespread adoption is hindered by a fundamental generalization challenge (Li et al., 2024; Zhang and Zhang, 2023; Hong et al., 2024). The immense diversity of applications and user-specific tasks creates a long-tail of scenarios that is infeasible to cover with conventional pre-training or fine-tuning paradigms, leading to poor performance in unseen situations as illustrated in Figure 1. To address this long-tail challenge, learning from a small number of user-provided demonstrations has emerged as a promis-

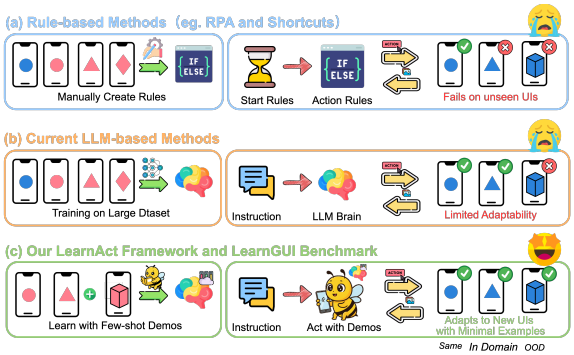

Figure 1: **The LearnAct Framework and LearnGUI Benchmark focus on addressing the long-tail challenges in mobile GUI agent performance through demonstration-based learning.** From rule-based automation to LLM-powered agents, mobile GUI automation has evolved significantly, yet still struggles with long-tail scenarios due to interface diversity. Our LearnAct framework introduces demonstration-based learning to effectively handle these challenges, outperforming existing methods in both offline and online evaluations.

Findings of the Association for Computational Linguistics: ACL 2026, pages 29820–29843

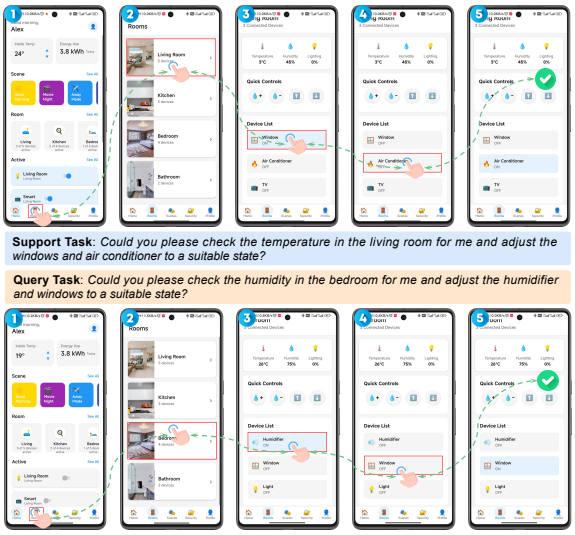

Figure 2: **Illustration of demonstration-based learning in LearnGUI.** Given a query task (e.g., checking bedroom humidity), the agent learns from similar support tasks (e.g., checking living room temperature) to complete the new task.

ing direction.[1] Pioneering works have begun to explore this paradigm for GUI agents (Zhang et al., 2023; Zheng et al., 2023; Kagaya et al., 2024; Lutz et al., 2024). However, despite these initial explorations, two critical gaps impeding progress toward robust and practical mobile agents.

First, a comprehensive benchmark for the systematic and reproducible evaluation of demonstration-based learning in mobile agents is conspicuously absent. Second, there is no systematic framework specifically designed for this task on mobile devices. Prior methods are often not generalizable to the mobile domain. Frameworks such as Synapse (Zheng et al., 2023), RAP (Kagaya et al., 2024), and WILBUR (Lutz et al., 2024) target web environments with accessible structured metadata. In the mobile context, AppAgent (Zhang et al., 2023) represents a notable exploration, but its approach of logging UI element functions from UI Trees relies on fragile metadata often unavailable on mobile devices (Lu et al., 2024b; Wang et al., 2024a; Liu et al., 2024) and lacks a systematic approach to knowledge extraction and utilization.

To bridge these fundamental gaps, we present two key contributions. We introduce **LearnGUI, the first comprehensive benchmark specifically designed to evaluate how mobile GUI agents learn from few-shot demonstrations** (Figure 2). Furthermore, we develop **LearnAct, a modular agent framework engineered to systematically extract, retrieve, and leverage knowledge from demonstrations purely through visual input**. Our experimental results decisively validate our framework's effectiveness. In summary, our contributions are as follows:

- We develop LearnGUI, the first benchmark designed for studying demonstration-based learning in mobile GUI agents, comprising 2,252 offline and 101 online tasks with high-quality human demonstrations.
- We design and implement LearnAct, a modular agent framework with specialized components that systematically extracts, retrieves, and leverages knowledge from human demonstrations.
- Extensive evaluations across six backbone models demonstrate substantial gains: LearnAct improves Gemini-2.5-Pro from 38.5% to 58.9% in offline tests and boosts UI-TARS-7B-SFT's online success rate from 18.1% to 32.8% (+81.2% relative), with consistent improvements across both general-purpose and specialized models, validating the broad applicability of demonstration-based learning.

2 LearnGUI Benchmark

Systematic research into how mobile GUI agents can learn from a few demonstrations requires a specialized benchmark. To address this need, we introduce LearnGUI, a benchmark designed to fairly evaluate and analyze an agent's ability to utilize a given set of k-shot demonstrations. Its core principle is to provide a controlled environment where demonstration quantity and multi-dimensional similarity (instruction, UI, action) are systematically varied, enabling reproducible research into demonstration-based learning. The LearnGUI benchmark consists of two components designed for distinct evaluation purposes: **LearnGUI-Offline** for systematic analysis of **step-by-step task execution** across varying similarity conditions, and **LearnGUI-Online** for assessing **end-to-end task completion** in real-world interactive scenarios.

[1]For a comprehensive review of related work on mobile GUI agents and benchmarks, see Appendix A.

2.1 Data Collection

LearnGUI-Offline is built upon the AMEX dataset (Chai et al., 2024), restructuring its independent tasks into k-shot combinations with multi-dimensional similarity metrics. LearnGUI-Online augments the AndroidWorld environment (Rawles et al., 2024a) by providing high-quality human demonstrations for its tasks. Both components share a unified action space (CLICK, TYPE, SWIPE, PRESS_BACK, PRESS_HOME, PRESS_ENTER, and TASK_COMPLETE). The detailed construction process for both components is provided in Appendix B.1.

2.2 Dataset Statistics

Table 5 presents the comprehensive statistics of the LearnGUI benchmark in comparison with existing datasets. With 2,353 instructions across 73 applications and an average of 13.2 steps per task, LearnGUI offers rich data for studying demonstration-based learning in mobile GUI agents. The dataset provides various k-shot combinations (k=1,2,3) for each task, along with multi-dimensional similarity metrics across instruction, UI, and action dimensions. This design enables systematic analysis of how different types and quantities of demonstrations affect learning outcomes. The similarity distribution reflects the natural variation in mobile tasks within applications, with a meaningful spread across similarity levels that allows for detailed investigation of knowledge transfer under different conditions. A detailed visualization of these similarity distributions is provided in Appendix B.2.

2.3 Dataset Splits

We divided LearnGUI-Offline into training and testing splits. The training set contains 2,001 tasks from 44 applications, while the test set includes 251 tasks from 9 entirely different applications, following the strict application-partitioned methodology of AMEX (Chai et al., 2024). Based on empirically determined thresholds, we categorize tasks into four similarity quadrants, as shown in Figure 3. These thresholds were determined by defining the top 30% of all computed similarity scores in the dataset as *high similarity* (SH), ensuring our categorization reflects the natural distribution of task similarity in real-world applications. This allows for systematic analysis of how different similarity types affect learning. Appendix B.3 provides detailed statistics of these splits and a full explanation of the quadrants.

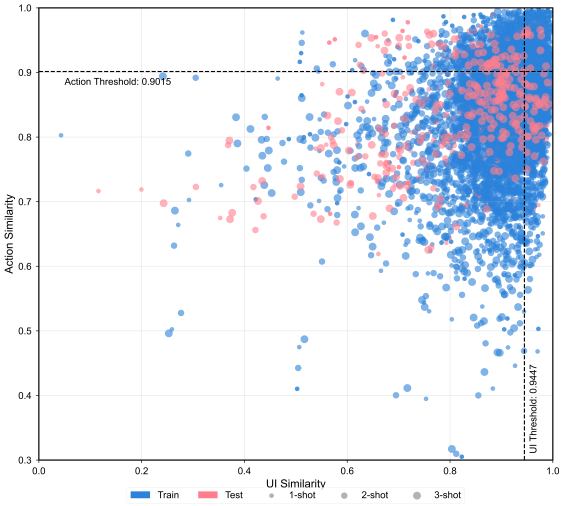

Figure 3: **Joint distribution of UI similarity and action similarity in LearnGUI-Offline.** The quadrant divisions represent our categorization of tasks into four profiles: $UI_{SH}Act_{SH}$, $UI_{SH}Act_{SL}$, $UI_{SL}Act_{SH}$, and $UI_{SL}Act_{SL}$, enabling analysis of how different similarity combinations affect learning transfer.

Collectively, the comprehensive structure of LearnGUI, which combines carefully designed offline splits for controlled analysis with an online environment for real-world validation, provides a powerful and unified resource for studying how mobile GUI agents learn from demonstrations under varying conditions.

3 Method: LearnAct

Building on the insights from our LearnGUI benchmark, we introduce LearnAct, a modular agent framework that enhances mobile GUI agents through demonstration-based learning, making them more adaptable in unseen scenarios (Figure 4). The framework comprises three synergistic components: (1) DemoParser (Section 3.1), which extracts structured knowledge from raw demonstrations; (2) KnowSeeker (Section 3.2), which retrieves the most relevant knowledge for a given task; and (3) ActExecutor (Section 3.3), which leverages this knowledge to guide task execution.

3.1 DemoParser

The DemoParser transforms raw human demonstrations into structured demonstration knowledge. It takes as input a raw action sequence (consisting of coordinates-based clicks, swipes, and text inputs) along with corresponding screenshots and task instructions. It then utilizes a Gemini-1.5-Pro to gen-

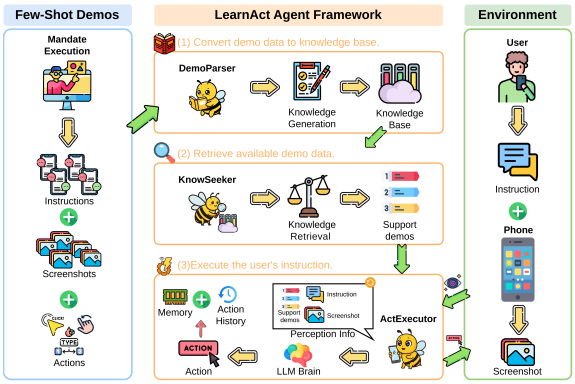

Figure 4: **Illustration of the overall framework of LearnAct.** Architecture diagram showing the three main components (DemoParser, KnowSeeker, ActExecutor) and their interconnections within the LearnAct system, including data flow from human demonstrations to execution.

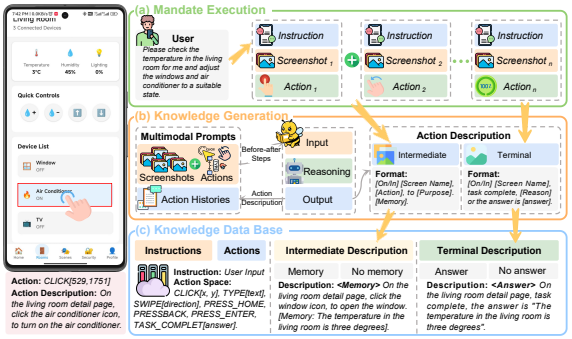

Figure 5: **Pipeline of DemoParser.** Input instructions and corresponding actions and screenshots; output low-level action descriptions and create knowledge database. This process transforms high-level user instructions into precise operation sequences while building a reusable domain knowledge base to improve mobile interface interaction automation efficiency.

erate semantically descriptive action descriptions that capture the essence of each demonstration step (e.g., “On Search Page, click the search box, to enter keywords”). Building on these descriptions, it constructs a structured knowledge base that records both the high-level action semantics and the contexts in which they occur, as shown in Figure 5.

Formally, DemoParser implements a knowledge generation function $G : \mathcal{I} \times \mathcal{S} \times \mathcal{A} \to \mathcal{K}$, where $\mathcal{I}$ represents the space of instructions, $\mathcal{S}$ is the space of screenshot sequences, $\mathcal{A}$ is the space of action sequences, and $\mathcal{K}$ is the knowledge space. For each demonstration trajectory $(i, s, a) \in \mathcal{I} \times \mathcal{S} \times \mathcal{A}$, DemoParser generates a knowledge entry $k = (i, a, d) \in \mathcal{K}$, where i is the original instruction, a is the raw action sequence, and d is the paired sequence of semantic action descriptions. Screenshots are consumed during offline parsing to generate d, while ActExecutor receives the retrieved instruction, raw actions, and semantic descriptions. This format converts raw coordinate-based actions (e.g., CLICK[123,456]) into meaningful operation descriptions (e.g., "click search box").

The knowledge generation process is decomposed into a sequence of description generation steps for each action in the demonstration trajectory. Let d_j represent the description for action a_j, which is generated using a context-aware description function $\delta : \mathcal{I} \times \mathcal{A}_j \times \mathcal{V}_j \times \mathcal{H}_{j-1} \to \mathcal{D}$, where $\mathcal{V}_j$ is the visual representation of action a_j execution and $\mathcal{H}_{j-1} = \{d_1, d_2, \ldots, d_{j-1}\}$ is the history of previous action descriptions. For each demonstration, DemoParser augments the original instruction and action sequence with semantically descriptive action descriptions, providing crucial context about the purpose and significance of each action.

For intermediate actions, DemoParser analyzes a visual representation of the action execution showing before-action and after-action screenshots with the action visualized (e.g., click locations highlighted). The framework combines this visual input with the task instruction, action history, and current action to generate a description that follows a standardized format: "[On/In] [Screen Name], [Action Details], to [Purpose]". For example: "On Home Screen, tap ’Settings’ icon, to access device configuration." For terminal actions, DemoParser processes the final screenshot, task instruction, and complete action history to generate a conclusion in the format: "[On/In] [Screen], complete task, [Reason/Answer]"

A distinctive feature of DemoParser is its memory mechanism, which captures critical information observed during task execution. This is particularly valuable for complex tasks requiring information retention across multiple steps. The model identifies and annotates task-relevant information that will likely be needed in subsequent steps and has not been previously recorded. These memory annotations are included in the action descriptions when appropriate: "[On/in] [Screen], [Action], to [Purpose]. [Memory: important information]". The complete algorithm and detailed prompt templates are provided in Appendix C.1.

3.2 KnowSeeker

KnowSeeker is the retrieval component that bridges the knowledge base and the executor by identify-

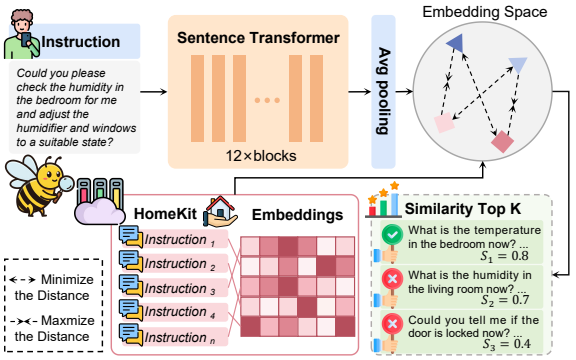

Figure 6: **Pipeline of KnowSeeker.** TheKnowSeeker converts demo trajectories from the knowledge base into a vector database. When executing user tasks, KnowSeeker retrieves the top-k relevant demos from the vector database for subsequent use. This approach enables efficient retrieval of similar demonstrations to assist with new task execution.

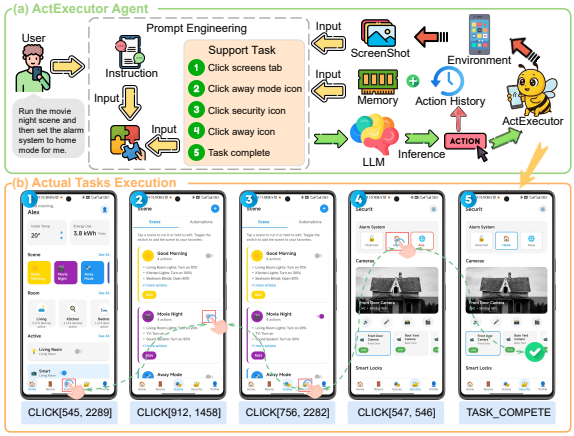

Figure 7: **Pipeline of ActExecutor.** The ActExecutor executes the low-level action descriptions generated by the Action Planner Agent. It uses theKnowSeeker to retrieve relevant demonstrations from the knowledge base and execute the actions in the demonstrations. This approach enables efficient execution of low-level actions to assist with new task execution.

ing demonstrations most relevant to the current task (Figure 6). It implements a retrieval function that selects a subset of knowledge entries based on the semantic similarity between the current task instruction and the instructions stored in the knowledge base. At its core, the retrieval mechanism relies on transforming instructions into dense vector representations (embeddings). The similarity between the current instruction i and a demonstration's instruction i_j is then quantified using cosine similarity on their respective embeddings, e_i and e_j:

$$sim(i, i_j) = \frac{e_i \cdot e_j}{\|e_i\| \cdot \|e_j\|} \quad (1)$$

To ensure runtime efficiency, embeddings for all demonstration instructions are pre-computed, enabling fast vector searches during task execution. This approach ensures that knowledge retrieval scales effectively as the knowledge base grows. The detailed formal definition of the retrieval function and implementation specifics are provided in Appendix C.2.

3.3 ActExecutor

ActExecutor is the framework's execution component, responsible for translating retrieved demonstration knowledge into effective actions in the environment (Figure 7). Its decision-making is governed by a policy π implemented through a large vision-language model (VLM). This policy processes a comprehensive prompt P that integrates all available information sources to generate the final action. The policy function can be expressed as:

$$\pi(o_t, \mathcal{D}) = f_{LLM}(P(i, o_t, h_{t-1}, \mathcal{D})) \quad (2)$$

where the prompt P is dynamically constructed to include the user instruction i, the current observation (screenshot) o_t, the action history h_{t-1}, and crucially, the retrieved demonstration knowledge $\mathcal{D}$. This structure allows the agent to reason based on context and relevant examples.

The execution process follows a standard perception-decision-action loop. In the Perception Phase, the agent observes the current screen state. In the Decision Phase, it constructs the comprehensive prompt and uses the VLM to select an action. Finally, in the Action Phase, the chosen action is executed in the environment. This loop continues until the task is completed or a maximum step limit is reached. The full execution algorithm and detailed prompt templates are provided in Appendix C.3.

This approach enables ActExecutor to leverage demonstrations as contextual exemplars, guiding its actions when faced with novel UI states or tasks. By closing the loop between knowledge extraction (DemoParser), relevant retrieval (KnowSeeker), and knowledge-guided execution (ActExecutor), the LearnAct framework provides a complete, end-to-end solution for effective demonstration-based learning in mobile GUI agents.

Models	Method	Supports	Average	Gmail	Booking	Music	SHEIN	NBC	CM	ToDo	Signal	Yelp
SPHINX-GUI	AMEX	0-shot	67.2	45.9	64.5	74.4	71.8	70.3	67.4	79.3	64.9	66.3
Gemini-1.5-Pro	Baseline	0-shot	19.3	20.1	16.4	24.5	10.2	35.6	14.1	17.4	27.9	15.2
	LearnAct	1-shot	51.7 [+32.4]	55.5	47.1	**60.0**	35.7	**56.4**	54.7	60.6	63.1	54.6
		2-shot	55.6 [+36.3]	57.5	53.2	55.3	39.6	56.1	58.2	68.1	69.7	60.0
		3-shot	**57.7** [+38.4]	**58.4**	**56.6**	54.6	**43.9**	53.9	**69.4**	**69.2**	**70.5**	57.6
Gemini-2.5-Pro	Baseline	0-shot	38.5	45.9	24.5	57.3	35.7	41.6	41.7	65.0	61.7	47.4
	LearnAct	1-shot	57.5 [+19.0]	61.7	51.8	63.3	46.7	63.8	55.3	73.8	70.5	52.1
		2-shot	58.5 [+20.0]	61.7	51.3	64.0	52.6	64.4	56.5	**79.1**	78.7	49.7
		3-shot	**58.9** [+20.4]	**63.3**	**52.4**	**67.3**	**51.4**	**63.3**	**58.8**	75.9	**73.8**	**50.3**
Gemini-2.5-Flash	Baseline	0-shot	27.5	25.9	19.9	44.7	23.7	37.3	30.5	31.9	40.1	39.9
	LearnAct	1-shot	46.0 [+18.5]	49.1	34.4	65.1	43.9	55.3	46.5	64.2	57.4	48.5
		2-shot	48.4 [+20.9]	50.9	35.7	66.2	47.9	57.6	51.2	67.0	65.6	52.1
		3-shot	**50.3** [+22.8]	**52.4**	**38.5**	**67.6**	**50.0**	**59.8**	**52.4**	**65.6**	**68.0**	**54.6**
UI-TARS-7B-SFT	Baseline	0-shot	77.5	68.1	81.0	81.1	72.9	80.9	70.6	66.0	92.6	82.4
	LearnAct	1-shot	**82.8** [+5.3]	79.9	**82.9**	**86.6**	75.7	86.3	79.4	84.0	**89.3**	83.0
		2-shot	81.9 [+4.4]	**80.1**	80.7	86.2	**76.1**	87.2	80.0	83.7	84.4	**84.2**
		3-shot	82.1 [+4.6]	79.9	80.9	86.2	75.7	**86.9**	**81.2**	**85.8**	84.4	**84.2**
Qwen2-VL-7B-Instruct	Baseline	0-shot	71.8	60.8	73.9	76.0	65.5	75.5	62.9	78.7	82.8	69.1
	LearnAct	1-shot	77.3 [+5.5]	**75.0**	77.5	77.8	69.8	83.5	72.9	78.0	83.6	78.8
		2-shot	78.5 [+6.7]	**75.0**	78.0	77.8	**73.3**	86.0	73.5	81.9	**87.7**	77.6
		3-shot	**79.4** [+7.6]	**75.0**	**78.8**	**78.6**	72.6	**87.8**	**77.1**	**82.6**	**87.7**	**80.6**

Table 1: **Overall performance on LearnGUI-Offline dataset (action match accuracy %).** Results show absolute values and relative improvements [in brackets] compared to baselines. Performance is evaluated across different models and number of support examples (1/2/3-shot) on the full test set, including per-application breakdown.

4 Experiments

We conducted comprehensive evaluations of the LearnAct framework through both offline and online experiments. The offline experiments were performed on the LearnGUI-Offline dataset to evaluate step-by-step task execution capabilities, while the online experiments utilized the LearnGUI-Online platform to assess end-to-end task completion in real-world interactive scenarios.

4.1 Experiment Setup

Our experiments evaluate the LearnAct framework across both offline and online settings. The framework's component models are fixed: DemoParser uses Gemini-1.5-Pro (Team et al., 2024) for knowledge extraction, and KnowSeeker uses a sentence transformer model for retrieval. Our analysis centers on the ActExecutor, for which we evaluate six distinct backbone models: Gemini-1.5-Pro, Gemini-2.5-Pro (Comanici et al., 2025), Gemini-2.5-Flash (Comanici et al., 2025), UI-TARS-7B-SFT (Qin et al., 2025), Qwen2-VL-7B-Instruct (Wang et al., 2024d), and Qwen3-VL-8B-Instruct (Bai et al., 2025). For offline evaluations, UI-TARS-7B-SFT and Qwen2-VL-7B-Instruct were fine-tuned on the LearnGUI-Offline training set, whereas the three Gemini models (1.5-Pro, 2.5-Pro, and 2.5-Flash) were used without fine-tuning to assess their zero-shot generalization and in-context learning capabilities. For online evaluations, we additionally test Qwen3-VL-8B-Instruct without fine-tuning (using its original pre-trained weights) to assess how newer-generation foundation models perform with demonstration-based learning in real-world interactive scenarios. In online tests, all LearnAct-enhanced agents utilized a single (1-shot) retrieved demonstration. We benchmark our framework against strong baselines, including the zero-shot performance of each backbone model, and report step accuracy for offline tasks and success rate (SR) for online tasks. Full implementation details are provided in Appendix D.

4.2 Main Results and Analysis

Our experiments reveal several key findings on the effectiveness of demonstration-based learning for mobile GUI agents.

Finding 1: Demonstration-based learning significantly boosts performance across model generations, with consistent benefits for both general-purpose and specialized models. As shown in Table 1, LearnAct yields substantial improvements across all models. Among general-

Models	Sup.	$UI_{SH}Act_{SH}$		$UI_{SH}Act_{SL}$		$UI_{SL}Act_{SH}$		$UI_{SL}Act_{SL}$	
		type	match	type	match	type	match	type	match
Gemini-1.5-Pro	1	79.5 [+12.8]	50.2 [+**35.6**]	78.1 [+12.3]	47.8 [+33.2]	77.5 [+9.2]	52.3 [+30.5]	77.9 [+**14.1**]	44.2 [+29.3]
	2	77.7 [+**13.0**]	53.9 [+**37.3**]	73.2 [+10.8]	49.9 [+34.7]	80.0 [+9.0]	56.5 [+34.8]	77.2 [+12.9]	48.9 [+34.4]
	3	72.3 [+**15.8**]	53.5 [+**39.6**]	72.8 [+12.9]	49.5 [+34.6]	78.7 [+10.4]	60.0 [+38.4]	79.2 [+12.8]	51.6 [+36.3]
Gemini-2.5-Pro	1	86.2 [+6.2]	54.0 [+23.2]	84.6 [+4.4]	52.1 [+17.6]	84.2 [+4.4]	57.3 [+25.1]	83.4 [+4.4]	49.2 [+15.6]
	2	83.9 [+7.3]	58.6 [+22.8]	83.8 [+3.1]	53.8 [+13.8]	84.8 [+4.9]	56.9 [+24.6]	84.0 [+4.4]	50.8 [+16.8]
	3	83.2 [+9.9]	62.4 [+24.6]	84.2 [+4.3]	55.2 [+17.5]	85.4 [+6.7]	59.3 [+26.0]	84.7 [+4.6]	51.5 [+17.7]
Gemini-2.5-Flash	1	86.2 [+2.5]	41.1 [+15.6]	85.5 [+1.4]	42.2 [+12.5]	84.0 [+2.2]	43.4 [+11.3]	83.1 [+0.6]	38.5 [+7.0]
	2	82.4 [+5.6]	47.2 [+13.7]	84.1 [+4.5]	50.9 [+11.3]	83.9 [+2.9]	43.8 [+9.4]	84.2 [+3.7]	40.0 [+8.6]
	3	82.2 [+8.5]	53.5 [+13.5]	82.7 [+6.7]	53.7 [+11.5]	83.7 [+5.0]	45.9 [+11.6]	85.6 [+5.5]	42.5 [+10.8]
Qwen2-VL-7B-Instruct	1	86.0 [+**5.3**]	72.2 [+**6.3**]	85.4 [+4.9]	69.6 [+5.5]	86.0 [+2.0]	76.2 [+5.4]	82.9 [+1.3]	69.4 [+4.3]
	2	85.0 [+**5.7**]	75.6 [+**9.3**]	84.0 [+4.1]	71.2 [+5.7]	86.9 [+2.5]	76.8 [+6.3]	84.0 [+2.5]	70.5 [+5.5]
	3	80.2 [+**5.0**]	70.3 [+7.9]	82.9 [+4.7]	70.2 [+5.7]	85.6 [+1.9]	77.5 [+**8.4**]	85.6 [+3.4]	72.8 [+6.6]
UI-TARS-7B-SFT	1	88.1 [+1.9]	77.8 [+**6.6**]	87.2 [+**2.1**]	75.3 [+6.4]	87.7 [+0.3]	80.1 [+5.9]	85.0 [-0.2]	75.0 [+2.8]
	2	85.5 [+**2.1**]	76.7 [+**8.3**]	85.7 [+1.6]	75.9 [+4.9]	87.3 [-0.4]	79.1 [+5.9]	84.9 [-0.8]	74.1 [+2.1]
	3	87.1 [+**7.9**]	78.2 [+**13.9**]	85.5 [+2.6]	75.4 [+4.9]	86.0 [-0.9]	78.9 [+6.8]	85.5 [-0.9]	75.2 [+2.7]

Table 2: **Performance breakdown of LearnGUI-Offline on different UI and action combinations.** Performance metrics (type and match accuracy) across four similarity quadrants showing absolute values and relative improvements [in brackets] compared to baselines. Results are grouped by model and number of support examples (1/2/3-shot).

purpose models, we observe a clear progression: Gemini-1.5-Pro achieves dramatic gains from 19.3% to 57.7% (+198.9%), while the newer Gemini-2.5-Pro demonstrates stronger baseline performance (38.5%) and reaches 58.9% with three demonstrations (+20.4 percentage points absolute gain). Notably, Gemini-2.5-Flash, despite being a more efficient model variant, achieves competitive performance (50.3% with 3-shot) from a lower baseline (27.5%), showing an +82.9% relative improvement. This progression validates that demonstration-based learning remains effective across model generations and architectures.

Even for specialized models like UI-TARS-7B-SFT, which start from a strong 77.5% baseline, a single demonstration provides a significant boost to 82.8%. The pattern reveals that while general-purpose models benefit more dramatically from demonstrations (especially when starting from lower baselines), all model types consistently improve with demonstration-based learning. Interestingly, the benefit of multiple demonstrations ($k > 1$) varies by model capability: general-purpose models like Gemini-1.5-Pro show continued scaling with more examples (19.3%→51.7%→55.6%→57.7%), whereas specialized models and more capable general-purpose models (Gemini-2.5-Pro) plateau earlier, suggesting they can effectively extract task patterns from fewer demonstrations.

Finding 2: Demonstrations provide synergistic visual and procedural knowledge, with model capability modulating the relative importance of UI versus action similarity. Analysis of performance across the similarity quadrants (Table 2) reveals a nuanced interplay that varies by model sophistication. High UI similarity (UI_{SH}) provides crucial *visual grounding*, creating a strong anchor that helps agents map demonstrated actions to correct UI elements. This effect is powerfully illustrated by Gemini-1.5-Pro, which achieves its peak 3-shot match accuracy gain of +39.6% in the $UI_{SH}Act_{SH}$ quadrant.

Interestingly, more capable models show different sensitivity patterns. Gemini-2.5-Pro exhibits strong performance across all quadrants, with its highest absolute accuracy (62.4%) in $UI_{SH}Act_{SH}$, but maintains robust performance even in challenging $UI_{SL}Act_{SL}$ scenarios (51.5% with 3-shot). In contrast, Gemini-2.5-Flash shows more pronounced dependence on similarity: it achieves 53.5% in $UI_{SH}Act_{SH}$ but only 42.5% in $UI_{SL}Act_{SL}$, suggesting that resource-efficient models benefit more critically from high-similarity demonstrations.

Action similarity plays a vital complementary role by providing *procedural scaffolding* that defines the *how* of a task. Its importance becomes particularly salient when visual grounding is weak (UI_{SL}). For specialized models like UI-TARS-7B-SFT, high action similarity boosts match accuracy by +6.8% ($UI_{SL}Act_{SH}$), far outpacing the +2.7%

Models	Params	SR (%)
Proprietary Models		
GPT-4o(Hurst et al., 2024)	-	34.5
Gemini-Pro-1.5(Team et al., 2024)	-	22.8
Claude CUA(Anthropic, 2024)	-	27.9
SeedVL-1.5(Guo et al., 2025a)	-	62.1
Open-Source Models		
Aguvis(Xu et al., 2024b)	72B	26.1
UI-TARS-72B-DPO(Qin et al., 2025)	72B	46.6
GUI-Critic-R1-7B(Wanyan et al., 2025)	7B	27.6
Qwen2-VL-7B-Instruct + 0-shot	7B	9.9
Qwen2-VL-7B-Instruct + LearnAct	7B	21.1 [+11.2]
Qwen3-VL-8B-Instruct + 0-shot	8B	47.6
Qwen3-VL-8B-Instruct + LearnAct	8B	**56.9** [+9.3]
UI-TARS-7B-SFT + 0-shot	7B	18.1
UI-TARS-7B-SFT + LearnAct	7B	32.8 [+14.7]

Table 3: **Performance comparison of different models on the LearnGUI-Online benchmark.** Task success rate (SR) is measured. GPT-4o and Gemini-Pro-1.5 use both image and accessibility tree (AXTree) as input, while all other models use only image input. Improvements from LearnAct are shown in brackets.

gain when both similarities are low. This dual-axis analysis validates the design of LearnGUI and reveals a key insight: while all models benefit from demonstrations, their learning strategies differ—more capable models can generalize across dissimilar contexts, whereas less capable models require closer alignment between demonstration and target task.

Finding 3: Offline gains translate to significant online task success, bridging the performance gap between smaller specialized models and SOTA LLMs. The benefits observed in static offline evaluations successfully transfer to dynamic, real-world environments (Table 3). With LearnAct, the 7B parameter UI-TARS-7B-SFT model's success rate jumps from 18.1% to 32.8%, approaching the 34.5% of the much larger GPT-4o. Notably, Qwen3-VL-8B-Instruct achieves a strong 47.6% baseline without fine-tuning, surpassing all compared systems, and reaches 56.9% with LearnAct, demonstrating that demonstration-based learning effectively complements even advanced foundation models. This validates that our paradigm offers a practical pathway to enhance both specialized and general-purpose models for real-world deployment. Further details, including performance breakdown charts and case studies, are available in Appendix E.1.

Ablation Setting		Average
DemoParser	KnowSeeker	
Baseline		19.3
	✓	40.6
✓		41.6
✓	✓	**51.7**

Table 4: **Ablation study of LearnAct components.** Performance comparison on average accuracy. A detailed per-application breakdown is in Appendix E (Table 8).

Finding 4: Both knowledge extraction and retrieval are critical and complementary. To validate our framework design, we conducted an ablation study (Table 4). The results are unequivocal: removing either component causes a significant performance drop compared to the full framework (51.7% accuracy). Without DemoParser's semantic enrichment, performance falls to 40.6%; without KnowSeeker's relevance-based retrieval, where one demonstration is sampled uniformly at random from the same application's knowledge base, it drops to 41.6%. This confirms that merely providing demonstrations is insufficient; the agent must be able to both parse them into structured, meaningful knowledge and retrieve the most relevant examples for the task at hand. Additional random-k baselines and per-application breakdowns are provided in Appendix E.2.

5 Discussion and Future Work

Our findings champion a paradigm shift for mobile GUI agents: away from solely pursuing larger models toward demonstration-based learning. The results underscore that *how* an agent learns, particularly how it leverages demonstrations for visual grounding and procedural reasoning, can be as important as its raw scale. While LearnAct and LearnGUI establish a strong foundation, several promising directions emerge. One avenue is exploring demonstration quality and diversity, including learning from imperfect or noisy demonstrations to enhance robustness. Another path involves evolving from static to dynamic knowledge bases that enable self-learning from successful executions and synthesize generalizable sub-routines through richer abstraction techniques.

6 Conclusion

We address generalization challenges in mobile GUI agents by introducing LearnGUI, the first benchmark for demonstration-based learning (2,252 offline and 101 online tasks), and LearnAct, a modular framework that extracts, retrieves, and utilizes knowledge from visual demonstrations. Experiments show substantial gains: Gemini-2.5-Pro improves from 38.5% to 58.9% offline, while UI-TARS-7B-SFT's online success rate increases from 18.1% to 32.8%. Our work provides a robust benchmark and a systematic framework, paving the way for more adaptable and practical mobile agents.

Limitations

While our work establishes a strong foundation for demonstration-based learning in mobile GUI agents, several limitations warrant acknowledgment. First, our benchmark focuses on Android mobile devices, and generalization to other platforms (e.g., iOS) requires further investigation. Second, LearnGUI constructs demonstration-query pairs within the same application; cross-app transfer is an important direction for future benchmark extensions. Third, our approach assumes demonstrations are provided a priori; adaptive demonstration selection based on agent uncertainty could further improve performance.

Acknowledgments

We thank the anonymous reviewers and the area chair for their constructive feedback. This work was supported by the National Natural Science Foundation of China (Grant No. 92367205 and No. U24A20258) and the Natural Science Foundation of Zhejiang Province under Grant LR26F030002.

References

Anthropic. 2024. Developing a computer use model.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and 1 others. 2021. Uibert: Learning generic multimodal representations for ui understanding. *arXiv preprint arXiv:2107.13731*.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, and 45 others. 2025. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. 2021. Mobile app tasks with iterative feedback (motif): Addressing task feasibility in interactive visual environments. *arXiv preprint arXiv:2104.08560*.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. 2024. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*.

Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, and 1 others. 2024. Guicourse: From general vision language models to versatile gui agents. *arXiv preprint arXiv:2406.11317*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.

Pengzhou Cheng, Zheng Wu, Zongru Wu, Tianjie Ju, Aston Zhang, Zhuosheng Zhang, and Gongshen Liu. 2025. Os-kairos: Adaptive interaction for mllm-powered gui agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6701–6725.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Dong Guo, Faming Wu, Feida Zhu, Fuxing Leng, Guang Shi, Haobin Chen, Haoqi Fan, Jian Wang, Jianyu Jiang, Jiawei Wang, and 1 others. 2025a. Seed1.5-vl technical report. *arXiv preprint arXiv:2505.07062*.

Yuan Guo, Tingjia Miao, Zheng Wu, Pengzhou Cheng, Ming Zhou, and Zhuosheng Zhang. 2025b. Atomic-to-compositional generalization for mobile agents with a new benchmark and scheduling system. *arXiv preprint arXiv:2506.08972*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. 2024. Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents. *arXiv preprint arXiv:2402.03610*.

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

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*.

Yinheng Li, Hailey Hultquist, Justin Wagle, and Kazuhito Koishida. 2025. Instruction agent: Enhancing agent with expert demonstration. *arXiv preprint arXiv:2509.07098*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, and 1 others. 2025. Llm-powered gui agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*.

Guangyi Liu, Pengxiang Zhao, Yaozhen Liang, Qinyi Luo, Shunye Tang, Yuxiang Chai, Weifeng Lin, Han Xiao, WenHao Wang, Siheng Chen, and 1 others. 2026. Memgui-bench: Benchmarking memory of mobile gui agents in dynamic environments. *arXiv preprint arXiv:2602.06075*.

Zhe Liu, Cheng Li, Chunyang Chen, Junjie Wang, Boyu Wu, Yawen Wang, Jun Hu, and Qing Wang. 2024. Vision-driven automated mobile gui testing via multimodal large language model. *arXiv preprint arXiv:2407.03037*.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024a. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024b. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

Michael Lutz, Arth Bohra, Manvel Saroyan, Artem Harutyunyan, and Giovanni Campagna. 2024. Wilbur: Adaptive in-context learning for robust and accurate web agents. *arXiv preprint arXiv:2404.05902*.

Pawel Pawlowski, Krystian Zawistowski, Wojciech Lapacz, Marcin Skorupa, Adam Wiacek, Sebastien Postansque, and Jakub Hoscilowicz. 2024. Tinyclick: Single-turn agent for empowering gui automation. *arXiv preprint arXiv:2410.11871*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024a. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2024b. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36.

Yunpeng Song, Yiheng Bian, Yongtao Tang, and Zhongmin Cai. 2023. Navigating interfaces with ai for enhanced user interaction. *arXiv preprint arXiv:2312.11190*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan. 2022. Ugif: Ui grounded instruction following. *arXiv preprint arXiv:2211.07615*.

Gaurav Verma, Rachneet Kaur, Nishan Srishankar, Zhen Zeng, Tucker Balch, and Manuela Veloso. 2025. Adaptagent: Adapting multimodal web agents with few-shot learning from human demonstrations. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 20635–20651.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *arXiv preprint arXiv:2406.01014*.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024b. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*.

Luyuan Wang, Yongyu Deng, Yiwei Zha, Guodong Mao, Qinmin Wang, Tianchen Min, Wei Chen, and Shoufa Chen. 2024c. Mobileagentbench: An efficient and user-friendly benchmark for mobile llm agents. *arXiv preprint arXiv:2406.08184*.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024d. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Yuyang Wanyan, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Jiabo Ye, Yutong Kou, Ming Yan, Fei Huang, Xiaoshan Yang, and 1 others. 2025. Look before you leap: A gui-critic-r1 model for pre-operative error diagnosis in gui automation. *arXiv preprint arXiv:2506.04614*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 543–557.

Hao Wen, Hongming Wang, Jiaxuan Liu, and Yuanchun Li. 2023. Droidbot-gpt: Gpt-powered ui automation for android. *arXiv preprint arXiv:2304.07061*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Han Xiao, Guozhi Wang, Yuxiang Chai, Zimu Lu, Weifeng Lin, Hao He, Lue Fan, Liuyang Bian, Rui Hu, Liang Liu, and 1 others. 2025. Ui-genie: A self-improving approach for iteratively boosting mllm-based mobile gui agents. *arXiv preprint arXiv:2505.21496*.

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

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

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024b. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*.

Jiayi Zhang, Chuang Zhao, Yihan Zhao, Zhaoyang Yu, Ming He, and Jianping Fan. 2024a. Mobileexperts: A dynamic tool-enabled agent team in mobile devices. *arXiv preprint arXiv:2407.03913*.

Jiwen Zhang, Jihao Wu, Yihua Teng, Minghui Liao, Nuo Xu, Xiao Xiao, Zhongyu Wei, and Duyu Tang. 2024b. Android in the zoo: Chain-of-action-thought for gui agents. *arXiv preprint arXiv:2403.02713*.

Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024c. Llamatouch: A faithful and scalable testbed for mobile ui automation task evaluation. *arXiv preprint arXiv:2404.16054*.

Zhuosheng Zhang and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436*.

Pengxiang Zhao, Guangyi Liu, Yaozhen Liang, Weiqing He, Zhengxi Lu, Yuehao Huang, Yaxuan Guo, Kexin Zhang, Hao Wang, Liang Liu, and 1 others. 2025. Mas-bench: A unified benchmark for shortcut-augmented hybrid mobile gui agents. *arXiv preprint arXiv:2509.06477*.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2023. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *arXiv preprint arXiv:2306.07863*.

Dataset	# Inst.	# Apps	# Step	HL	LL	GT	FS
Static Datasets							
PixelHelp	187	4	4.2	✓	✗	✓	✗
MoTIF	276	125	4.5	✓	✓	✓	✗
UIBert	16,660	-	1	✗	✓	✓	✗
UGIF	523	12	6.3	✓	✓	✓	✗
AITW	30,378	357	6.5	✓	✗	✓	✗
AITZ	2,504	70	7.5	✓	✓	✓	✗
AndroidControl	15,283	833	4.8	✓	✓	✓	✗
AMEX	2,946	110	12.8	✓	✗	✓	✗
AppAgent	50	10	-	✓	✗	✗	✗
Benchmarks with Dynamic Environments							
LlamaTouch	496	57	7.01	✓	✗	✓	✗
AndroidWorld	116	20	-	✓	✗	✗	✗
AndroidLab	138	9	8.5	✓	✗	✗	✗
LearnGUI	**2,353**	**73**	**13.2**	✓	✓	✓	✓

Table 5: **Comparison of different datasets and environments for benchmarking Mobile GUI agents.** Column definitions: # Inst. (number of instructions), # Apps (number of applications), # Step (average steps per task), HL (has high-level instructions), LL (has low-level instructions), GT (provides ground truth trajectories), FS (supports few-shot learning).

A Related Work

Mobile GUI Agent Benchmarks. High-quality datasets are crucial for developing and evaluating mobile GUI agents. Existing resources can be broadly categorized as static datasets or benchmarks with dynamic environments. Static datasets provide task descriptions with corresponding UI states and actions (Li et al., 2020; Burns et al., 2021; Bai et al., 2021; Venkatesh et al., 2022; Rawles et al., 2024b; Zhang et al., 2024b; Li et al., 2024; Chai et al., 2024; Wang et al., 2024c), but are typically collections of independent tasks not structured for few-shot learning evaluation. Benchmarks with dynamic environments like LlamaTouch (Zhang et al., 2024c), AndroidWorld (Rawles et al., 2024a), AndroidLab (Xu et al., 2024a), UI-Nexus (Guo et al., 2025b), MAS-Bench (Zhao et al., 2025), and MemGUI-Bench (Liu et al., 2026) offer interactive or compositional testing, but natively lack the demonstration-query pairings required for systematic few-shot analysis. While AppAgent (Zhang et al., 2023) explored demonstration-based learning by providing 50 instruction texts, it lacks the ground-truth annotations and structured demonstration pairings necessary for systematic analysis. As a result, a critical gap exists for a benchmark specifically designed to systematically and fairly evaluate few-shot demonstration learning for mobile GUI agents (see FS column in Table 5).

Mobile GUI Agents. Prevailing strategies for mobile GUI agents (Liu et al., 2025) rely on either prompting large language models (Wei et al., 2022; Yao et al., 2024; Wen et al., 2023; Song et al., 2023; Wang et al., 2024b,a) or fine-tuning them on extensive datasets (Cheng et al., 2024; Chen et al., 2024; Lu et al., 2024a; Pawlowski et al., 2024; Hong et al., 2024; Lin et al., 2024; Xu et al., 2024b). Recent systems also improve reliability through adaptive intervention, scheduling, self-improvement, or memory mechanisms, such as OS-Kairos (Cheng et al., 2025), Agent-Nexus (Guo et al., 2025b), UI-Genie (Xiao et al., 2025), and UI-Mem (Xiao et al., 2026). As noted in the introduction, these approaches struggle with generalization in diverse, real-world scenarios. While demonstration-based learning has been explored to mitigate this (Zhang et al., 2023; Zheng et al., 2023; Kagaya et al., 2024; Lutz et al., 2024; Verma et al., 2025; Li et al., 2025), our work is distinguished by its focus on a systematic, vision-only framework co-designed with a comprehensive mobile-first benchmark. In contrast to web-focused AdaptAgent (Verma et al., 2025) or desktop-focused Instruction Agent (Li et al., 2025), LearnAct builds a mobile knowledge base and retrieves transferable demonstrations across related mobile tasks. Our approach avoids reliance on UI Trees, which are often a fragile and inaccessible component in many mobile contexts (Lu et al., 2024b; Wang et al., 2024a; Liu et al., 2024), thus ensuring broader applicability.

B Benchmark Details

This section provides detailed descriptions of the LearnGUI benchmark, corresponding to Section 3 in the main paper.

B.1 Construction Details

B.1.1 LearnGUI-Offline Construction

We built LearnGUI-Offline by restructuring and enhancing the AMEX dataset (Chai et al., 2024), which contains 2,946 independent mobile tasks. To transform this resource for few-shot learning evaluation, we made several key modifications:

Action Space Standardization. We structured the action space to enhance its suitability for real-world scenarios. We removed 219 `TASK_IMPOSSIBLE` actions from the original AMEX dataset due to inconsistent labeling. To better support information retrieval tasks, we enhanced `TASK_COMPLETE`

to TASK_COMPLETE[answer]. This involved manually reviewing all original AMEX tasks, identifying those that required an answer but were not annotated as such, and adding the answers by hand. Ultimately, we annotated and retained 185 tasks in this new format, aligning our action space with modern benchmarks like AndroidWorld (Rawles et al., 2024a). The unified action space used across both benchmark components is detailed in Table 6.

Action	Definition
CLICK[x, y]	Click at coordinates (x, y).
TYPE[text]	Type the specified text.
SWIPE[direction]	Swipe in the specified direction.
PRESS_HOME	Go to the home screen.
PRESS_BACK	Go back to the previous app screen.
PRESS_ENTER	Press the enter button.
TASK_COMPLETE	Mark the task as complete. Provide answer if required.

Table 6: The action space of LearnGUI.

Quality Control. For LearnGUI-Offline, we inherited the human demonstrations from AMEX and additionally performed manual checks during action-space standardization: the 219 inconsistent TASK_IMPOSSIBLE actions were removed, and the 185 answer-style tasks converted to TASK_COMPLETE[answer] were reviewed by the authors. These checks ensure that the few-shot combinations are built from valid trajectories and terminal labels.

K-shot Task Combinations. We constructed systematic k-shot task combinations through a multi-step process. We began by recovering application context for each task through instruction and screenshot analysis, as the original dataset lacked explicit app labels. Next, we computed instruction similarity between tasks within the same application using the all-MiniLM-L6-v2 model. Finally, we created k-shot combinations (k=1,2,3) for each query task by selecting the k most similar tasks within the same application as support demonstrations, ensuring that the average similarity exceeded a minimum threshold of 0.6. This process yielded 2,252 tasks with valid k-shot combinations.

Similarity Measurement. To enable multi-dimensional similarity analysis, we computed metrics across three key dimensions. For Instruction Similarity, we utilized the scores calculated during the K-shot Task Combinations process. For UI Similarity, we merged the UI trees from all steps of each task and calculated similarity using TF-IDF vectorization and cosine similarity, capturing the visual and structural similarity of interfaces. For Action Similarity, following the DemoParser approach detailed in Section 3.1, we generated descriptive representations of each action and computed embedding-based cosine similarity between task pairs. To ensure our action similarity metric accurately captures the specific semantic understanding we want agents to master, we deliberately used this DemoParser-aligned process. This approach guarantees that the benchmark directly measures an agent's ability to leverage the exact type of semantic knowledge our framework provides, establishing clear methodological alignment between the evaluation criteria and our framework's objectives.

Separation from Agent Execution. The similarity metrics above are used only for dataset construction and post-hoc analysis. They are never provided to the agent, never used as supervision, and never affect the online execution score. KnowSeeker retrieves demonstrations using dense embeddings of raw task instructions only, while offline correctness is measured by action matching and online correctness is measured by AndroidWorld's programmatic task-success evaluator.

B.1.2 LearnGUI-Online Construction

For evaluating mobile GUI agents in real-time interactive scenarios, we developed LearnGUI-Online based on the AndroidWorld environment (Rawles et al., 2024a). While AndroidWorld provides 116 dynamically constructed task templates, it lacks human demonstration trajectories essential for few-shot learning evaluation.

We identified 101 tasks suitable for human completion, excluding 15 tasks that proved challenging for human users. We then collected high-quality human demonstrations for these tasks. For tasks with dynamic elements, we generated specific instances and recorded corresponding demonstrations. Demonstration quality was verified by retaining trajectories that satisfied AndroidWorld's built-in reward function and by manually filtering infeasible templates before collection. The resulting LearnGUI-Online dataset provides a realistic testbed for evaluating few-shot learning capabilities in mobile GUI agents under authentic conditions.

B.2 Additional Dataset Statistics

Figure 8 illustrates the distribution of similarity scores across different dimensions in the LearnGUI-Offline dataset, enabling systematic analysis of how

different types of similarity between demonstration and query tasks affect learning efficacy.

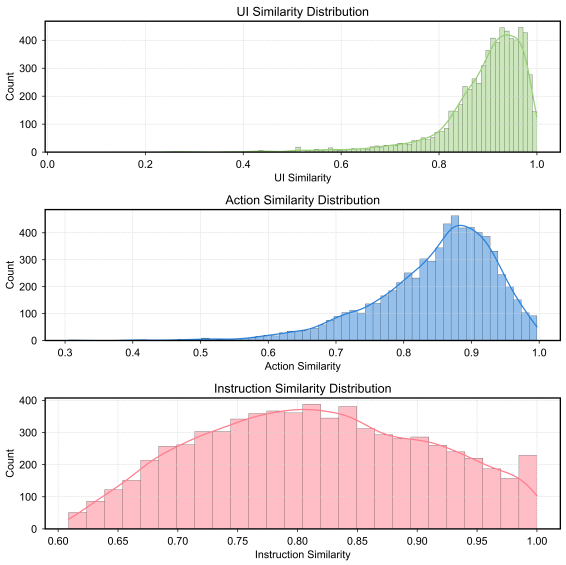

Figure 8: **Distribution of instruction, UI, and action similarity scores in LearnGUI-Offline.** The histograms show the distribution of similarity scores across three dimensions: instruction similarity (top), UI similarity (middle), and action similarity (bottom). These distributions enable systematic analysis of how different types of similarity between demonstration and query tasks affect learning efficacy.

B.3 Dataset Splits and Similarity Quadrants

Table 7 presents detailed statistics of the training and testing splits. In LearnGUI-Offline, tasks are classified into four quadrants based on high (SH) and low (SL) similarity for both UI and actions. The choice of defining the top 30% of scores as *high similarity* (SH) is deliberate. This threshold is chosen to ensure the SH category contains a sufficient number of tasks for meaningful statistical analysis while reflecting the natural distribution of task relatedness in real-world applications. This method establishes a clear, non-trivial distinction between high and low similarity tasks, yielding empirical thresholds of 0.9447 for UI and 0.9015 for action similarity. The four quadrants are defined as follows:

- $\textbf{UI}_{\textbf{SH}}\textbf{Act}_{\textbf{SH}}$: High UI similarity and high action similarity. For example, in a smart home app, two tasks that both involve adjusting the brightness of different lights in the living room would navigate through similar UI screens.
- $\textbf{UI}_{\textbf{SH}}\textbf{Act}_{\textbf{SL}}$: High UI similarity but low action similarity. For instance, in a smart home app, turning on all lights with a single button press versus adjusting each light's color temperature.
- $\textbf{UI}_{\textbf{SL}}\textbf{Act}_{\textbf{SH}}$: Low UI similarity but high action similarity. For example, setting a schedule for lights versus setting a schedule for the thermostat—different UI screens but similar action patterns.
- $\textbf{UI}_{\textbf{SL}}\textbf{Act}_{\textbf{SL}}$: Low UI similarity and low action similarity. For instance, checking security camera footage versus creating a scene that coordinates multiple devices.

C Framework Details

This section provides detailed descriptions of the components of our LearnAct framework, corresponding to the methods presented in Section 4 of the paper.

C.1 DemoParser

DemoParser is the knowledge extraction engine of LearnAct. Its primary function is to transform raw, low-level human demonstration trajectories (sequences of screen interactions) into a structured knowledge base of high-level, semantically rich action descriptions. This process converts coordinate-based actions (e.g., 'CLICK[123,456]') into meaningful operations (e.g., "click search box"), making the implicit knowledge in demonstrations explicit and machine-readable for the agent. Formally, DemoParser implements a knowledge generation function $G : \mathcal{I} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{K}$, where an instruction $\mathcal{I}$, screenshot sequence $\mathcal{S}$, and action sequence $\mathcal{A}$ are mapped to a structured knowledge entry $\mathcal{K}$.

C.1.1 Prompts

We provide all of our prompt templates used in DemoParser for generating semantically descriptive action descriptions from demonstration data. The process distinguishes between intermediate actions within a trajectory and the final, terminal action. For intermediate steps, a detailed prompt (Figure 9) guides the model to analyze before-and-after screenshots and describe the action's purpose, including a memory mechanism to retain critical information for future steps. For the terminal action, a separate set of prompts (Figure 10) is used to generate a concluding description, handling both standard task completions and those requiring a specific answer. These carefully designed prompts

Split	K-shot	Tasks	Apps	Actions	Ins_{Sim}	UI_{Sim}	Act_{Sim}	$\text{UI}_{\text{SH}}\text{Act}_{\text{SH}}$	$\text{UI}_{\text{SH}}\text{Act}_{\text{SL}}$	$\text{UI}_{\text{SL}}\text{Act}_{\text{SH}}$	$\text{UI}_{\text{SL}}\text{Act}_{\text{SL}}$
Training Split											
Offline-Train	1-shot	2,001	44	26,184	0.845	0.901	0.858	364	400	403	834
Offline-Train	2-shot	2,001	44	26,184	0.818	0.898	0.845	216	360	358	1,067
Offline-Train	3-shot	2,001	44	26,184	0.798	0.895	0.836	152	346	310	1,193
Testing Splits											
Offline-Test	1-shot	251	9	3,469	0.798	0.868	0.867	37	49	56	109
Offline-Test	2-shot	251	9	3,469	0.767	0.855	0.853	15	42	55	139
Offline-Test	3-shot	251	9	3,469	0.745	0.847	0.847	10	36	49	156
Online-Test	1-shot	101	20	1,423	-	-	-	-	-	-	-

Table 7: **Statistics of LearnGUI dataset splits.** Each split is analyzed across multiple dimensions: Tasks (number of tasks), Apps (number of applications covered), Actions (total action steps), similarity metrics ($\text{Ins/UI/Act}_{\text{Sim}}$), and distribution across four similarity profiles categorized by high (SH) and low (SL) UI and action similarity.

guide the vision-language model to produce structured knowledge that captures the essence of human demonstrations, as shown in Figures 9 and 10.

C.1.2 Algorithm

The complete knowledge generation process of DemoParser is detailed in Algorithm 1. The algorithm iterates through each demonstration in the dataset, generating a step-by-step semantic description for each action and compiling the results into a structured knowledge base.

C.2 KnowSeeker

KnowSeeker is the retrieval component of LearnAct, designed to efficiently identify the most relevant demonstrations from the knowledge base for a given task. It implements a retrieval function $R : \mathcal{I} \times \mathcal{K} \rightarrow \mathcal{K}^{(s)}$, where $\mathcal{I}$ is the instruction space, $\mathcal{K}$ is the full knowledge base, and $\mathcal{K}^{(s)} \subset \mathcal{K}$ is the retrieved subset of relevant knowledge entries. This filtering is crucial for providing the agent with focused, high-value examples.

The retrieval mechanism is implemented as a two-phase process centered on semantic similarity:

Offline Embedding Index Construction To ensure efficient retrieval, we pre-process the knowledge base K to build a searchable index, $\mathcal{K}_{idx}$. This index consists of pairs, each containing the embedding of a demonstration's instruction and a reference to the complete knowledge entry. This structure allows for fast similarity searches while retaining a direct link to the original data. We use the `all-MiniLM-L6-v2` sentence transformer model to generate the embeddings. The index construction is formally described as:

$$\mathcal{K}_{idx} = \{(f_{embed}(i), k) \mid k = (i, \_, \_) \in K\} \quad (3)$$

Algorithm 1 DemoParser Knowledge Generation Process

Require: Demonstration dataset $D = \{(i_k, s_k, a_k)_{k=1}^{N}\}$ where i_k is instruction, s_k is screenshot sequence, a_k is action sequence
Ensure: Knowledge base K with semantically descriptive action descriptions
$K \leftarrow \emptyset$
for each demonstration (i, s, a) in D **do**
  $d \leftarrow \emptyset$
  for $j = 1$ to $|a|$ **do**
    if $j < |a|$ **then**
      Create visualization of action a_j with before-after screenshots from s_j and s_{j+1}
      $h \leftarrow$ Previous action descriptions $\{d_1, d_2, \ldots, d_{j-1}\}$
      $d_j \leftarrow \text{GenerateDescription}(i, a_j, visualization, h)$ using prompt format detailed in Appendix C.1.1
      d_j follows format: "[On/In] [Screen], [Action], to [Purpose]" with optional memory
    else
      $h \leftarrow$ Complete action history $\{d_1, d_2, \ldots, d_{|a|-1}\}$
      $d_{|a|} \leftarrow \text{GenerateFinalDescription}(i, s_{|a|}, h, a_{|a|})$ using prompt detailed in Appendix C.1.1
      $d_{|a|}$ follows format: "[On/In] [Screen], complete task, [Reason/Answer]"
    end if
    Add d_j to description sequence d
  end for
  Add (i, a, d) to knowledge base K
end for

return K

Prompt 1: Intermediate Action Description

System Prompt:
You are a mobile UI interaction analyst. Follow these rules: 1. Analyze the split-screen image (Before-action left, After-action right) 2. For click actions, a high-contrast red marker (white-bordered circle) shows the precise click location, with a green square surrounding it and a 'C' label at the top-right corner of the square indicating the click. 3. Output JSON with ONLY ONE 'action_description' field in this exact format: "[On/In] [Screen Name], [Action Details], to [Purpose]"
Action Types: - click [element] (e.g. 'Search button') - swipe [up/down/left/right] - type [text] in [field] - press [back/home/enter]
Validation Rules: 1. Screen names should be 2-6 words 2. Keep purpose descriptions under 8 words 3. Never mention coordinates/IDs
MEMORY RECORDING RULES: If the current screen contains information relevant to the user's instruction that needs to be remembered for future steps, include a Memory part in your action description. The format should be: "[On/In] [Screen Name], [Action Details], to [Purpose]. [Memory: important information for future steps]"
Memory should ONLY be added when: 1. The information is relevant to completing the user's instruction 2. The information will likely be needed in future steps 3. This specific information has NOT been recorded in previous action history entries
Memory examples: 1. For a travel planning task: On Travel Blog, click 'Bali Beach Guide', to read article. [Memory: Guide mentions Kuta Beach has surfing lessons for $25/hour] 2. For a shopping task: In Product Details, click 'Add to Cart', to select item. [Memory: iPhone 13 Pro costs $999 with 128GB storage] 3. For a note-taking task: On Weather App, swipe down forecast, to view weekend. [Memory: Saturday will be rainy with 80% precipitation]
Avoid using Memory for: 1. Obvious UI changes that don't contain task-relevant information 2. Information already captured in previous action steps 3. Generic observations not specific to the user's task objective

Figure 9: **Prompt template for intermediate action descriptions.** The template guides DemoParser to generate standardized descriptions for intermediate actions, including detailed rules for memory annotations that capture important information observed during task execution.

where k is a knowledge entry from the knowledge base K, i is the instruction from that entry, and f_{embed} is the embedding function.

Online Similarity Computation and Retrieval During task execution, given a new user instruction i_{new}, its embedding $e_{new} = f_{embed}(i_{new})$ is computed. Following Equation (1) from the main text, cosine similarity is calculated between e_{new} and each instruction embedding e stored in the index $\mathcal{K}_{idx}$. The top-k knowledge entries k from the index pairs (e, k) that yield the highest similarity scores are selected as the most relevant demonstrations and passed to the ActExecutor.

C.3 ActExecutor

ActExecutor is the decision-making and execution component of the LearnAct framework. It is responsible for synthesizing all available information—the user's instruction, the current screen observation, the action history, and the retrieved demonstration knowledge from KnowSeeker—to generate the next action. Its core is a policy π implemented by a large vision-language model (VLM), which takes a comprehensive prompt as input to make context-aware decisions. By conditioning on relevant examples, the ActExecutor effectively performs in-context learning at each step, adapting its behavior to the specific nuances of the task at hand without needing to be retrained. The policy function can be expressed as:

$$\pi(o_t, \mathcal{D}) = f_{VLM}(P(i, o_t, h_{t-1}, \mathcal{D})) \quad (4)$$

where the prompt P is dynamically constructed from the instruction i, current observation o_t, action history h_{t-1}, and retrieved demonstrations $\mathcal{D}$.

C.3.1 Prompts

We provide the prompt templates used by ActExecutor to make decisions based on current observa-

Prompt 2: Terminal Action Description - Standard Completion

System Prompt for standard completion:
Determine the final task status. Output rules: 1. Use ONLY ONE 'action_description' field 2. Format: "[On/In] [Screen], complete task, [Reason]"
Validation Rules: - Reason should be less than 10 words - Screen name must match previous context
Examples: 1. Basic completion: On Payment Screen, complete task, successfully submit order 2. Failure case: In Search Results, cannot complete task, no nearby Vivo mobile phone stores found

Prompt 3: Terminal Action Description - With Answer

System Prompt for completion with answer:
Determine the final task status with the given answer. Output rules: 1. Use ONLY ONE 'action_description' field 2. Format: "[On/In] [Screen], complete task, the answer is [answer]"
Validation Rules: - Screen name must match previous context - Use the exact answer provided in the TASK_COMPLETE action
Examples: 1. Answer is a price: On Checkout Screen, complete task, the answer is "$299.9". 2. Answer is a list: On Payment Options Screen, complete task, the answer is "google pay, check out with affirm, add credit/debit card".

Figure 10: **Prompt templates for terminal action descriptions.** The templates provide specific formats for both standard task completion and information retrieval tasks, ensuring consistent output structure across different task types.

tions, action history, and demonstration knowledge. The prompt (Figure 11) is structured to provide the VLM with a complete operational context. It includes a role definition, the current screenshot with its dimensions, the user's high-level instruction, the history of operations already performed, and, crucially, the retrieved demonstration(s) as examples. The prompt concludes with strict response requirements, defining the valid action space and output format to ensure that the VLM's generation is constrained to executable commands. These prompts guide the vision-language model to select appropriate actions for task execution, as shown in Figure 11.

C.3.2 Algorithm

The full execution process of ActExecutor is detailed in Algorithm 2. The algorithm follows a standard perception-decision-action loop. In each iteration, it first perceives the current screen state (GetObservation), then constructs the comprehensive prompt integrating all contextual information (ConstructPrompt), uses the VLM to decide on the next action (f_{VLM}), and finally executes the action in the environment (ExecuteAction). This loop continues until the task is completed or a maximum step limit is reached, ensuring a structured and methodical approach to task execution.

Algorithm 2 ActExecutor Task Execution Process

Require: User instruction i, Knowledge base K, Maximum steps T
Ensure: Task execution trajectory
$t \leftarrow 0$
$h \leftarrow \emptyset$
$\mathcal{D} \leftarrow \text{KnowSeeker}(i, K)$
while $t < T$ and not IsTaskComplete **do**
  $o_t \leftarrow \text{GetObservation}()$
  $P_t \leftarrow \text{ConstructPrompt}(i, o_t, h, \mathcal{D})$
  $a_t \leftarrow f_{LLM}(P_t)$
  $d_t \leftarrow \text{GenerateDescription}(i, a_t, o_t, h)$
  $h \leftarrow h \cup \{(a_t, d_t)\}$
  $\text{ExecuteAction}(a_t)$
  $t \leftarrow t + 1$
end while

return $\{(a_0, d_0), (a_1, d_1), \ldots, (a_{t-1}, d_{t-1})\}$

D Experiment Setup Details

This section provides the detailed setup for the experiments described in Section 5 of the main paper.

Prompt 4: Task Execution Prompt

Role Definition:
You are a smartphone assistant to help users complete tasks by interacting with apps. I will give you a screenshot of the current phone screen.
Example Tasks: [Only when demonstrations are available]
Example 1: [Demonstration instruction] Steps taken in this example: Step-1: [Action] [Action Description] Step-2: [Action] [Action Description] ...
Background: This image is a phone screenshot. Its width is [width] pixels and its height is [height] pixels. The user's instruction is: [instruction]
History operations: [Only when action history is available]
Before reaching this page, some operations have been completed. You need to refer to the completed operations to decide the next operation. These operations are as follow: Step-1: [Action] [Action Description] Step-2: [Action] [Action Description] ...
Response requirements: Now you need to combine all of the above to decide just one action on the current page. You must choose one of the actions below:
`"SWIPE[UP]"`: Swipe the screen up. `"SWIPE[DOWN]"`: Swipe the screen down. `"SWIPE[LEFT]"`: Swipe the screen left. `"SWIPE[RIGHT]"`: Swipe the screen right. `"CLICK[x,y]"`: Click the screen at the coordinates (x, y). x is the pixel from left to right and y is the pixel from top to bottom `"TYPE[text]"`: Type the given text in the current input field. `"PRESS_BACK"`: Press the back button. `"PRESS_HOME"`: Press the home button. `"PRESS_ENTER"`: Press the enter button. `"TASK_COMPLETE[answer]"`: Mark the task as complete. If the instruction requires answering a question, provide the answer inside the brackets. If no answer is needed, use empty brackets `"TASK_COMPLETE[]"`.
Response Example: Your output should be a string and nothing else, containing only the action type you choose from the list above. For example: `"SWIPE[UP]"` `"CLICK[156,2067]"` `"TYPE[Rome]"` `"PRESS_BACK"` `"PRESS_HOME"` `"PRESS_ENTER"` `"TASK_COMPLETE[1h30m]"` `"TASK_COMPLETE[]"`

Figure 11: **Task execution prompt template.** This comprehensive prompt directs ActExecutor to generate actions based on current observations, action history, and retrieved demonstrations, with explicit formatting requirements to ensure consistent action outputs.

Foundation Models and Training We conducted experiments with six foundation models: Gemini-1.5-Pro, Gemini-2.5-Pro, Gemini-2.5-Flash, UI-TARS-7B-SFT, Qwen2-VL-7B-Instruct, and Qwen3-VL-8B-Instruct. For all models, we set the temperature to zero to obtain deterministic responses. For Qwen2-VL-7B-Instruct and UI-TARS-7B-SFT, we employed parameter-efficient fine-tuning using LoRA with rank 64, alpha 128, and dropout probability 0.1. We targeted all modules while freezing the vision encoder to ensure computational efficiency. Training used a learning rate of 1e-5 with cosine scheduling, batch size of 1, gradient accumulation over 8 steps, a warmup ratio of 0.001, and was conducted for 1 epoch. All fine-tuning experiments were conducted on 8 NVIDIA L40S GPUs. For offline experiments, the three Gemini models (1.5-Pro, 2.5-Pro, and 2.5-Flash) were evaluated directly on the LearnGUI-Offline test set without additional training, allowing us to assess their zero-shot generalization and in-context learning capabilities. UI-TARS-7B-SFT and Qwen2-VL-7B-Instruct were fine-tuned on the LearnGUI-Offline training set before evaluation. The inclusion of multiple Gemini model variants enables us to examine how demonstration-based learning scales across different model generations (1.5 vs 2.5) and efficiency tiers (Pro vs Flash). For online experiments, we deployed all models except Gemini-1.5-Pro (which showed limited task completion capabilities in preliminary tests despite accuracy improvements) to the LearnGUI-Online environment, with Qwen3-VL-8B-Instruct additionally evaluated exclusively in online settings without offline fine-tuning to assess how newer-generation foundation models perform with demonstration-based learning using their original pre-trained weights. All LearnAct-enhanced mod-

els used 1-shot demonstration retrieval. Our online evaluation is built directly upon the AndroidWorld benchmark (Rawles et al., 2024a), utilizing identical environment setup (Android emulator via ADB), evaluation metrics, and protocols. The only difference in our experimental setup is providing agents with the capability to retrieve similar demonstration trajectories, ensuring robust experimental conditions and direct comparability with other AndroidWorld-based studies.

Baselines To rigorously evaluate our approach, we compared LearnAct against several baselines. These include: (1) SPHINX-GUI Agent, the original agent developed for the AMEX dataset, providing a reference point for task execution on similar data; (2) Zero-shot inference versions of all models (Gemini-1.5-Pro, Gemini-2.5-Pro, Gemini-2.5-Flash, UI-TARS-7B-SFT, Qwen2-VL-7B-Instruct, and Qwen3-VL-8B-Instruct) within the LearnAct framework but without demonstration knowledge, maintaining identical execution environments for fair comparison; and (3) For online evaluation, we additionally compared against GPT-4o, Gemini-Pro-1.5, Claude Computer-Use, and Aguvis to benchmark against current advanced systems. The zero-shot baselines for the three Gemini models and Qwen3-VL-8B-Instruct establish how well these foundation models perform on mobile GUI tasks without any task-specific demonstrations, providing a clear assessment of the value added by demonstration-based learning.

Evaluation Metrics For offline evaluation, we adopted mainstream evaluation protocols widely used in recent mobile GUI agent research, such as UI-TARS (Qin et al., 2025) and OS-ATLAS (Wu et al., 2024). Specifically, we measured step accuracy, which consists of two components: action type accuracy and action match accuracy. Action type accuracy measures the percentage of steps where the predicted action type (CLICK, TYPE, SWIPE, etc.) matches the ground truth. Action match accuracy measures the percentage of steps where both the action type and its parameters are correct, following standard evaluation criteria. For CLICK actions, coordinates are considered correct if they fall within 14% of screen width from the ground truth. For TYPE actions, the content is correct if the F1 score between prediction and ground truth exceeds 0.5. For SWIPE actions, the direction must precisely match the ground truth. For other actions (e.g., PRESS_BACK), an exact match is required. For TASK_COMPLETE actions, we only verify the action type and ignore the answer field. For online evaluation, we measured the task success rate (SR), which represents the percentage of tasks completed successfully in the real-time interactive environment.

E Additional Experimental Results

This section provides additional experimental results and analyses that supplement the findings presented in Section 5 of the paper.

E.1 Detailed Online Evaluation Results

To provide further support for Finding 3, this section presents more granular results from the online experiments, including quantitative performance comparisons and qualitative case studies.

Performance Comparison Visualizations Figures 12 and 13 provide detailed comparisons of model performance with and without LearnAct enhancement in online evaluation scenarios. These charts break down the success rate across various task dimensions, illustrating the consistent benefits of the LearnAct framework.

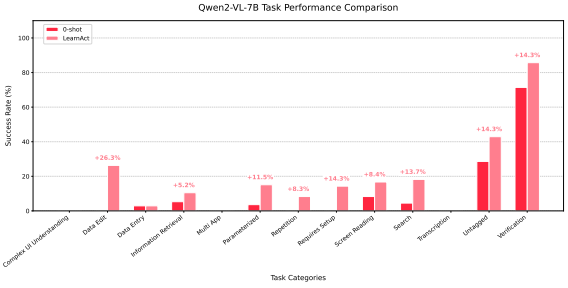

Figure 12: **Detailed performance comparison of Qwen2-VL-7B-Instruct with and without LearnAct on LearnGUI-Online.** The figure shows the task success rates of Qwen2-VL-7B-Instruct baseline versus Qwen2-VL-7B-Instruct enhanced with LearnAct across different task dimensions in the LearnGUI-Online benchmark.

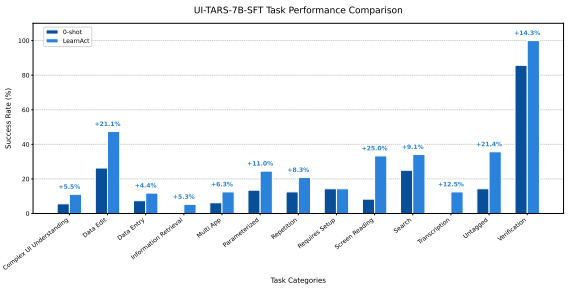

Figure 13: **Detailed performance comparison of UI-TARS-7B-SFT with and without LearnAct on LearnGUI-Online.** The figure presents a comprehensive breakdown of task success rates for UI-TARS-7B-SFT baseline versus UI-TARS-7B-SFT enhanced with LearnAct across multiple task dimensions in the LearnGUI-Online benchmark.

Qualitative Case Studies We present three detailed case studies from our online experiments to provide concrete examples of how LearnAct leverages demonstration knowledge to solve tasks in unseen mobile applications. These case studies highlight different scenarios where demonstration knowledge proves particularly beneficial for task execution, including complex navigation (Figure 15), information retrieval (Figure 14), and data manipulation (Figure 16).

E.2 Ablation Study Details

To supplement Finding 4, Table 8 provides a detailed, per-application breakdown of our ablation study. The data reinforces that the full framework, combining both DemoParser and KnowSeeker, consistently achieves the best performance across all tested applications. The table also reveals a nuanced interplay between the components. While removing either component universally degrades performance, their individual contributions vary. For instance, in applications like Gmail and Music, KnowSeeker's retrieval yields slightly better results than DemoParser's enrichment. This suggests that for some tasks, identifying the correct demonstration is more critical than parsing its steps. Conversely, in apps like NBC and Signal, DemoParser's contribution is more pronounced, highlighting the importance of semantic understanding. This underscores that both components are complementary, addressing different facets of the demonstration-based learning challenge.

E.3 Additional Baselines and Efficiency Analysis

Random-k Retrieval. Table 9 extends the "without KnowSeeker" ablation to all evaluated offline backbones and shot counts. Random-k keeps DemoParser's semantic representation unchanged but selects k demonstrations uniformly at random from the same application's knowledge base. KnowSeeker consistently improves over random retrieval, showing that the gains are not explained by simply exposing the model to more demonstrations.

Direct Few-Shot Prompting. Table 10 compares LearnAct with simpler few-shot prompting strategies using Gemini-1.5-Pro. Direct multimodal ICL injects screenshots and raw actions as examples; w/o DemoParser injects raw actions only with the same retrieval as LearnAct; w/o KnowSeeker uses random retrieval with DemoParser descriptions. The full framework performs best, supporting the need for both semantic parsing and relevance-based retrieval.

Demo-Specific Fine-Tuning. We also compared LearnAct with lightweight demo-specific LoRA adaptation on the same demonstrations. As shown in Table 11, LoRA gives small offline gains but requires retraining and underperforms LearnAct in the online setting, where randomized task parameters make overfitting to demonstration trajectories less useful.

Runtime Overhead. The dominant DemoParser cost is offline and one-time. At runtime, KnowSeeker performs one vector search over precomputed instruction embeddings without extra API calls. Table 12 shows the measured per-step overhead on LearnGUI-Online: the retrieved demonstration adds 594 tokens and about 0.2 seconds per step while improving success rate by 14.7 percentage points for UI-TARS-7B-SFT.

DemoParser Backbone Choices. Although the main experiments use Gemini-1.5-Pro for DemoParser, Table 13 shows that the module can be replaced by smaller or open-source VLMs. Qwen3-VL-8B-Instruct achieves the strongest online gain in this ablation, reducing dependence on closed-source parsing models.

Failure Modes. The online category breakdowns in Figures 12 and 13 show that LearnAct improves most on data editing, screen reading, search, and verification tasks. Remaining failures concentrate in complex UI understanding, transcription, and multi-app tasks, where success may require precise OCR, long-horizon state tracking, or interactions across app boundaries that cannot be fully inferred from a single retrieved demonstration. These failure modes motivate future work on dynamic retrieval, stronger state tracking, and cross-app demonstration transfer.

Ablation Setting		Average	Gmail	Booking	Music	SHEIN	NBC	CM	ToDo	Signal	Yelp
DemoParser	KnowSeeker										
Baseline		19.3	20.1	16.4	24.5	10.2	35.6	14.1	17.4	27.9	15.2
	✓	40.6	47.7	31.3	55.4	29.1	47.0	43.0	58.2	48.8	50.7
✓		41.6	46.9	34.1	52.7	27.9	51.9	45.3	51.4	61.1	51.8
✓	✓	**51.7**	**55.5**	**47.1**	**60.0**	**35.7**	**56.4**	**54.7**	**60.6**	**63.1**	**54.6**

Table 8: **Detailed ablation study of LearnAct components.** This table provides a detailed per-application breakdown of the ablation study results, complementing the summarized view in the main paper. Performance comparison across four configurations: baseline (no components), DemoParser only, KnowSeeker only, and both components combined.

Model	Method	1-shot	2-shot	3-shot
Gemini-1.5-Pro	Random-k	41.6	43.1	44.2
	LearnAct	**51.7**	**55.6**	**57.7**
Gemini-2.5-Pro	Random-k	49.8	51.0	51.6
	LearnAct	**57.5**	**58.5**	**58.9**
Gemini-2.5-Flash	Random-k	38.2	39.5	40.3
	LearnAct	**46.0**	**48.4**	**50.3**
UI-TARS-7B-SFT	Random-k	80.1	80.4	80.6
	LearnAct	**82.8**	**81.9**	**82.1**
Qwen2-VL-7B-Instruct	Random-k	74.1	74.8	75.2
	LearnAct	**77.3**	**78.5**	**79.4**

Table 9: **KnowSeeker vs. random-k retrieval on LearnGUI-Offline.** Values are action match accuracy (%).

Strategy	Demo Format	1-shot	2-shot	3-shot
Zero-shot	N/A	19.3	19.3	19.3
Direct multimodal ICL	Screenshot + action	26.4	27.8	29.1
w/o DemoParser	Action	40.6	–	–
w/o KnowSeeker	Action + description	41.6	43.1	44.2
LearnAct	Action + description	**51.7**	**55.6**	**57.7**

Table 10: **Few-shot strategy comparison on LearnGUI-Offline.** Values are Gemini-1.5-Pro action match accuracy (%).

Model	Setting	Offline	Online	Cost
UI-TARS-7B-SFT	LearnAct	82.8	**32.8**	0 GPU-hours
	Demo-specific LoRA	**84.3**	23.7	∼4 GPU-hours
Qwen2-VL-7B-Instruct	LearnAct	77.3	**21.1**	0 GPU-hours
	Demo-specific LoRA	**79.1**	15.2	∼4 GPU-hours

Table 11: **Demo-specific LoRA fine-tuning vs. LearnAct.** Offline reports 1-shot action match accuracy; online reports success rate.

Configuration	Tokens/Step	Latency/Step	SR
Zero-shot	6,167	∼2.5s	18.1
LearnAct 1-shot	6,761	∼2.7s	32.8
Overhead	+594 (+9.6%)	+∼0.2s (+8%)	+14.7

Table 12: **Per-step token and latency overhead.** Results are measured with UI-TARS-7B-SFT on one NVIDIA L40S 48G GPU.

DemoParser Backbone	Type	LearnAct SR	Gain
Qwen3-VL-8B-Instruct	Open-source 8B	**34.2**	**+16.1**
Gemini-1.5-Pro	Closed-source large	32.8	+14.7
GPT-4o-mini	Closed-source small	30.5	+12.4
Qwen2-VL-7B-Instruct	Open-source 7B	29.3	+11.2

Table 13: **DemoParser backbone ablation on LearnGUI-Online.** ActExecutor is UI-TARS-7B-SFT; the zero-shot baseline is 18.1% SR.

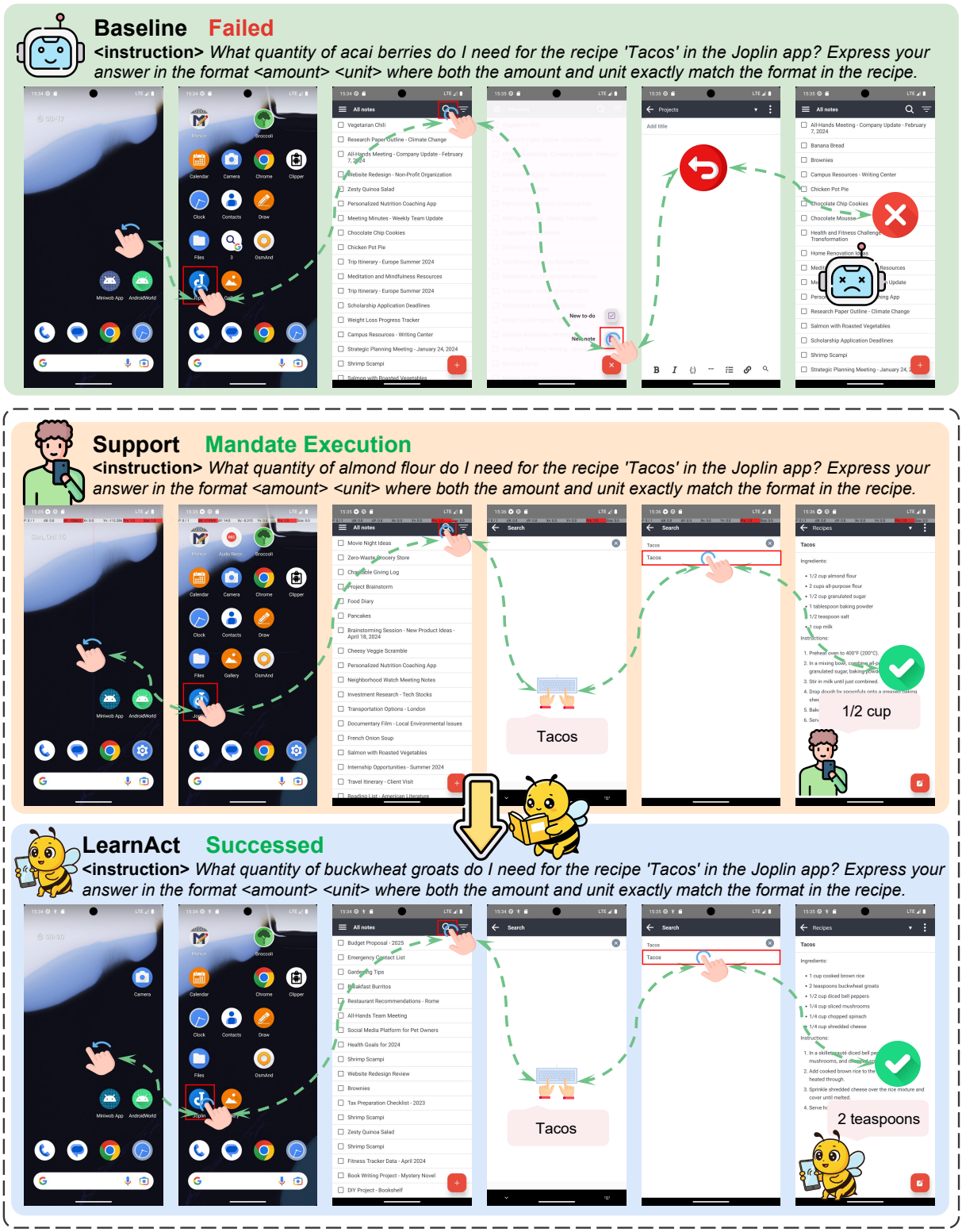

Figure 14: **UI-TARS-7B-SFT with LearnAct vs. Baseline in NotesRecipeIngredientCount Task.** Task template: "What quantity of {ingredient} do I need for the recipe '{title}' in the Joplin app? Express your answer in the format <amount> <unit> without using abbreviations." This task is classified as "easy" with an optimal completion in 2 steps, requiring search and parameter comprehension capabilities.

Figure 15: **UI-TARS-7B-SFT with LearnAct vs. Baseline in SimpleCalendarAddOneEvent Task.** Task template: "In Simple Calendar Pro, create a calendar event on {year}-{month}-{day} at {hour}h with the title '{event_title}' and the description '{event_description}'. The event should last for {duration_mins} mins." This "hard" difficulty task requires 17 optimal steps, involving data entry and complex UI understanding.

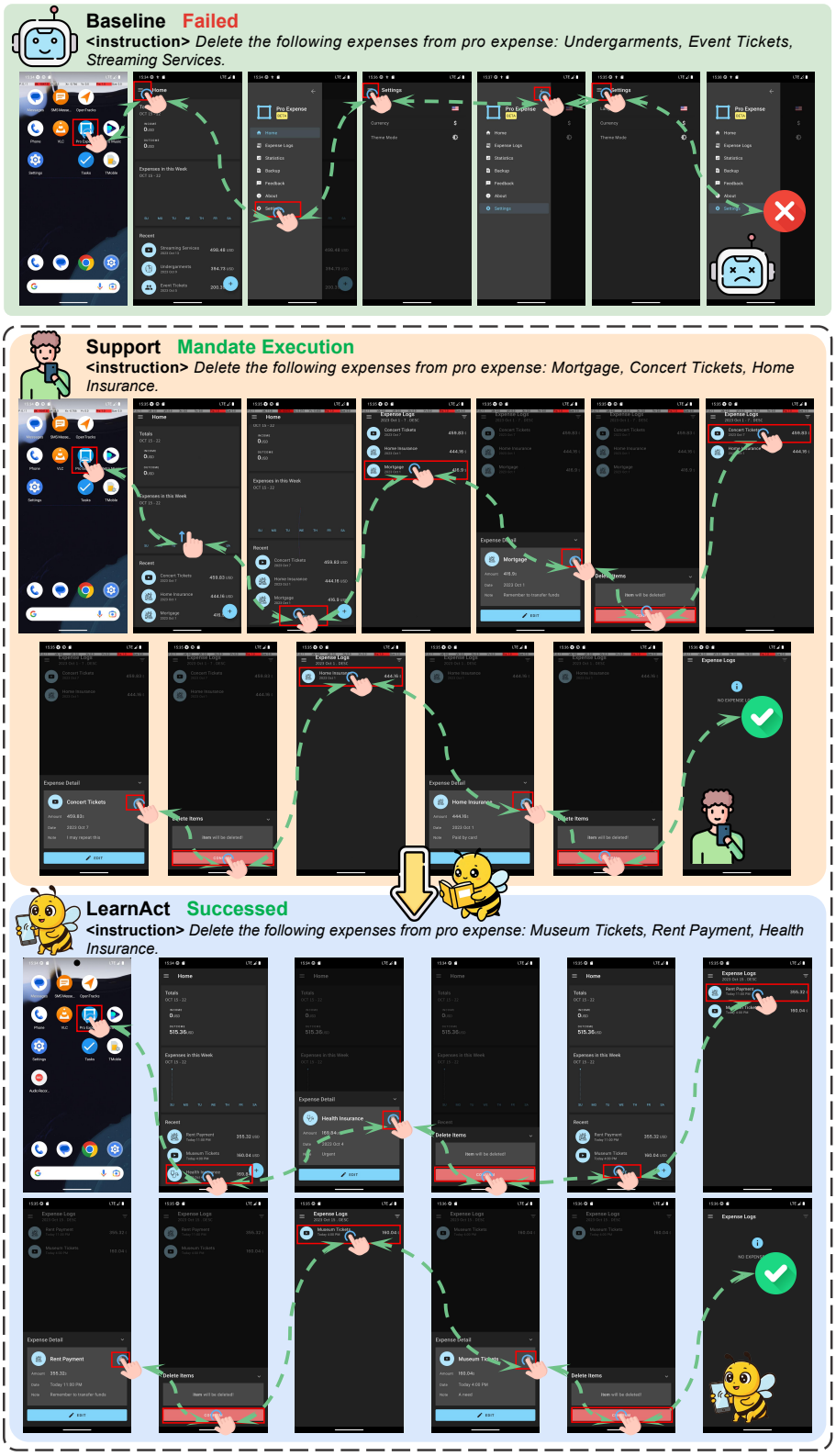

Figure 16: **Qwen2-VL-7B-Instruct with LearnAct vs. Baseline in ExpenseDeleteMultiple Task.** Task template: "Delete the following expenses from arduia pro expense: {expenses}." This task is rated “easy” with 10 optimal steps, focusing on data editing capabilities with parameterized inputs.