PSPA-BENCH: A Personalized Benchmark for Smartphone GUI Agent

Anonymous ACL submission

Abstract

Smartphone GUI agents operate directly on app interfaces, offering broad capability without deep system integration. However, real-world use is highly personalized: users follow diverse workflows and preferences, demanding customized rather than generic assistance. Existing benchmarks overlook this personalization dimension due to limited user-specific data and the lack of fine-grained evaluation. To address this gap, we introduce PSPA-BENCH, a benchmark dedicated to evaluating personalization in smartphone GUI agents. Specifically, PSPA-BENCH utilizes the *task decomposition graph, TDG* to represent the structure of personalized task at unit-instruction level. Using TDG, PSPA-BENCH generates task templates that instantiate personalized instructions without large-scale user logs, and supports fine-grained evaluation of GUI agents' persoalization abilities. PSPA-BENCH includes over 12,855 realistic personalized instructions spanning 10 scenarios and 22 commonly used mobile apps, along with four fine-grained evaluation metrics. We evaluate 11 state-of-the-art GUI agents and find that existing methods perform poorly under personalized settings, with even the best agent achieving limited success. Based on these results, we derive five insights to guide future research, positioning PSPA-BENCH as a foundation for advancing personalized GUI agents.

1 Introduction

Smartphone GUI agents (Zhang et al., 2025a; Liu et al., 2025a) have emerged as a promising paradigm for facilitating human-computer interaction in mobile environments. Different from conventional automation systems (e.g., Siri [1]) that rely on system APIs, GUI agents (Zhang et al., 2025b; Wang et al., 2024a; Qin et al., 2025; Wen et al., 2024) operate directly on the graphical user interfaces of mobile applications. By emulating user actions such as tapping, swiping, and text input, these agents remain effective in ecosystems with limited or no backend integration.

However, unlike many other computing platforms, smartphone usage is inherently personalized (Li et al., 2015; Nguyen et al., 2025). As shown in Figure 1, users vary significantly in how they accomplish the same task through diverse action sequences (Tian et al., 2020). For instance, when ordering takeout, different users may prefer different search strategies and different food categories. Consequently, the demand for smartphone GUI agents is shifting away from *generic, one-size-fits-all* solutions toward *personalized, context-aware* assistants. Moreover, a GUI agent is not intended to act as a *one-time* helper; rather, it is expected to provide continuous support over *long-term use* (Lee et al., 2024b; Wang et al., 2025; Hu et al., 2025). This requires the agent to accumulate execution experience and adapt to evolving individual preferences, delivering a "more understanding of you" interaction experience progressively.

Existing benchmarks evaluate GUI agents from different perspectives. For example, Mobile-Bench (Deng et al., 2024), AndroidWorld (Rawles et al., 2025) and Spa-Bench (Chen et al., 2025a), focus on the general task-execution capabilities of GUI agents; TransBench (Lu et al., 2025) emphasize the transferability of GUI agents; AEIA-MN (Chen et al., 2025b) and Mobile Safety Bench (Lee et al., 2024a) focus on the safety of GUI agents. However, considering both the **sparsity of user data** and the **absence of suitable metrics for personalized evaluation**, the systematic assessment of GUI agents' personalization capabilities remains largely unexplored.

To address this gap, we introduce PSPA-BENCH, a benchmark for systematic study of personalization in smartphone GUI agents. PSPA-BENCH uses the Task Decomposition Graph (TDG) to explicitly represent the structure of personalized

[1] https://www.apple.com/siri/

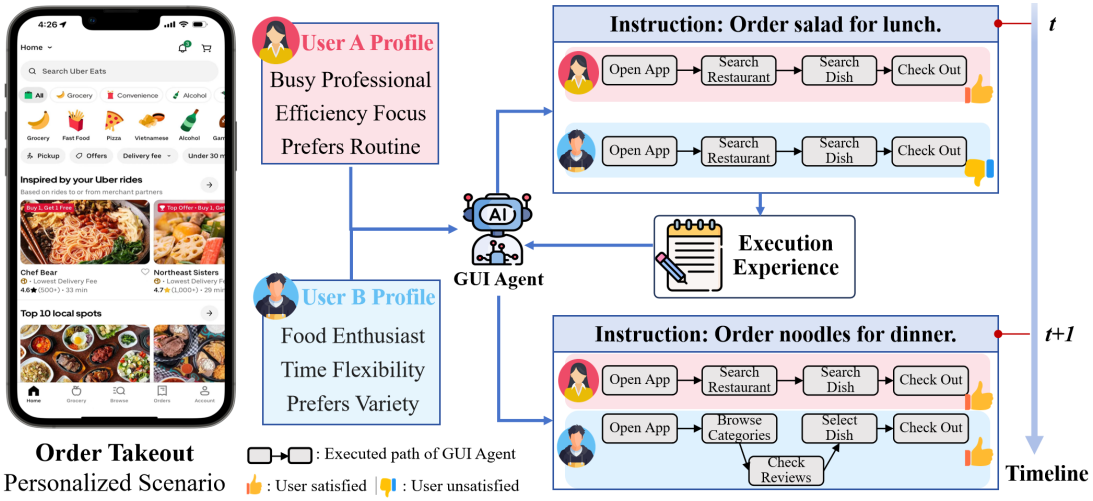


Figure 1: GUI agents are shifting from *general-purpose, one-time* task execution to *personalized, long-term* service. (i) Given the same instruction, an agent should adapt its execution path to user preferences; (ii) For the same user, the agent uses historical execution experience to provide long-term support; experience up to time t informs a refined execution at $t+1$ for similar instructions. The formulation of personalized GUI agent task is provided in Appendix B.

GUI tasks. To compensate for sparse user data, PSPA-BENCH generates personalized instructions by building TDG-derived templates without relying on large-scale user logs. For fine-grained evaluation, PSPA-BENCH measures the ratio of completed unit instructions as the fine-grained metrics by through graph-trace alignment at the unit-instruction level. PSPA-BENCH covers 10 daily-use scenarios (e.g., shopping, travel, dining) across 22 mobile apps, defines 100 user personas, and yields 12,855 personalized instructions. A detailed comparison with existing smartphone GUI benchmarks is provided in Appendix A. With the proposed PSPA-BENCH, we conduct comprehensive evaluations with 11 GUI agents, and the evaluation results from PSPA-BENCH demonstrate that there is still a long way for the development of personalized GUI agents. Furthermore, we highlight 5 key findings that provide directions for the development of personalized GUI agents: (i) effective **perception** underpins personalized instruction execution; (ii) strong **reasoning** is required to handle ambiguity and complex behaviors; (iii) personalization amplifies the trade-off between **accuracy and efficiency**; (iv) **persistent memory** is essential for long-term adaptation; and (v) **self-evolution** further enhances this adaptive capability. To summarize, our contributions are as follows:

- The problem of personalization in smartphone GUI agents is systematically formalized, highlighting the scarcity of personalized user data and the lack of evaluation metrics as core challenges.
- We propose PSPA-BENCH, a benchmark with a data generation method for sparse user data and process-oriented evaluation metrics based on task decomposition graphs. PSPA-BENCH covers 10 representative daily scenarios, 22 third-party apps, 100 user personas, and 12,855 personalized instructions
- 11 state-of-the-art GUI agents are benchmarked on PSPA-BENCH, and the results reveal that current methods perform poorly under personalized settings, and further distill 5 key directions for future research based on the evaluation results.

2 Benchmarking Setup

Despite rapid progress in GUI agents, the systematic assessment of their personalization capabilities remains largely unexplored. Current benchmarks face two main limitations. First, privacy regulations (e.g., GDPR (European Parliament and Council of the European Union, 2016)) and user concerns restrict the collection of user-specific data, hindering the creation of realistic yet privacy-compliant datasets. Second, evaluations rely on coarse metrics such as binary success signals (Chen et al., 2025a; Rawles et al., 2025; Yang et al., 2025) or task completion rates (Xing et al., 2024), which assume fixed action paths and fail to assess adaptability to diverse user preferences or learning over time.

To overcome these limitations, we propose PSPA-BENCH[2], a benchmark specifically designed for personalized GUI agents (Figure 2) . At the core of PSPA-BENCH is the *task decomposition graph* (TDG), which explicitly captures the personalized structure of GUI agent tasks. Based on the TDG, we explain how PSPA-BENCH generate personalized task instructions without relying on large-scale user logs, and evaluate agent executions with unit-instruction level fine-grained metrics. Finally, we present the comparison GUI agents employed in our benchmark.

2.1 Task Decomposition Graph

To address two limitations identified above, i.e., the sparsity of real user data and the lack of metrics tailored to personalization, we introduce the task decomposition graph (TDG). The TDG explicitly captures the personalized structure of GUI tasks. This structure (i) enables controlled generation of personalized task instances even when user traces are scarce, and (ii) supports fine-grained, preference-aware evaluation beyond coarse end-state signals.

Specifically, GUI tasks such as "Buy a piece of clothing on a shopping app" implicitly combine universal steps with user-specific requirements (e.g., preferred product category or price range). To make these requirements explicit, we represent each task as a directed acyclic task decomposition graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$, where nodes are minimal GUI actions and edges denote execution dependencies. Any source-to-sink path defines a valid task completion. To model personalization, we distinguish fixed nodes $\mathcal{U}_f$ for user-invariant steps (e.g., launching the app) and flexible nodes $\mathcal{U}_p$ for preference-dependent actions (e.g., selecting preferred categories).

Therefore, compared with vague task descriptions, the TDG provides a more explicit and structured representation of the task with fine-grained unit instructions. This graph-based representation can serve as a unified foundation for generating personalized instructions and for conducting fine-grained, evaluation of agent behavior. The constructed TDGs are provided in Appendix C.

[2]The code and data of PSPA-BENCH are available at https://anonymous.4open.science/status/PSPA-Bench

2.2 Template-Driven Personalized Instruction Generation

To address user data sparsity, we use the TDG as a generative blueprint to create diverse, realistic personalized instructions without large-scale user logs. The task structure ensures universal steps are preserved while systematically varying preference-dependent details.

Template construction. The TDG explicitly separates fixed nodes $\mathcal{U}_f$ from flexible nodes $\mathcal{U}_p$. We use the fixed nodes and dependencies $\mathcal{E}$ as a procedural backbone, and convert flexible nodes into parameterized slots. This template preserves the shared task structure while clearly indicating where personalization occurs.

Specifically, each template $\mathsf{TP}_\mathcal{G}$ encodes the fixed procedural backbone of the task (derived from $\mathcal{U}_f$) together with a slot schema $\mathcal{L} = \{l_{u_p}\}_{u_p \in \mathcal{U}_p}$ mapping each flexible node u_p to a slot l_{u_p}. Every slot $l \in \mathcal{L}$ is annotated with a semantic type $\tau(l)$ (e.g., `product_category`, `price_range`) and a domain of admissible values. The textual backbone preserves the execution ordering implied by $\mathcal{E}$, while slots mark precisely where personalization should be injected. We illustrate an example of the template for the shopping scenario as below:

Template Example for Shopping Scenario

```
Launch the shopping app, then search for
a <product_category> within <price_range>,
and add the top result to the cart.
```

Here, "<product_category>" and "<price_range>" are slots derived from flexible nodes in the TDG, while the rest reflects the fixed structure.

Slot instantiation. Once a template is selected, we instantiate it by filling each parameterized slot with values sampled from the user's preference distribution. For a user profile ρ, each slot l is assigned a value $v_l \sim D^\rho_{\tau(l)}$, yielding a concrete instruction $f(\mathsf{TP}_\mathcal{G}, v_l) \mapsto I$ that respects the dependencies in $\mathcal{G}$. In this framework, the user profile ρ directly controls the *personalization strength*. A set of similar user profiles induces low-diversity slot values and thus weaker personalization, whereas a set of diverse profiles leads to more varied instructions, corresponding to stronger personalization signals. We provide the detailed instruction instantiation process in Appendix D.

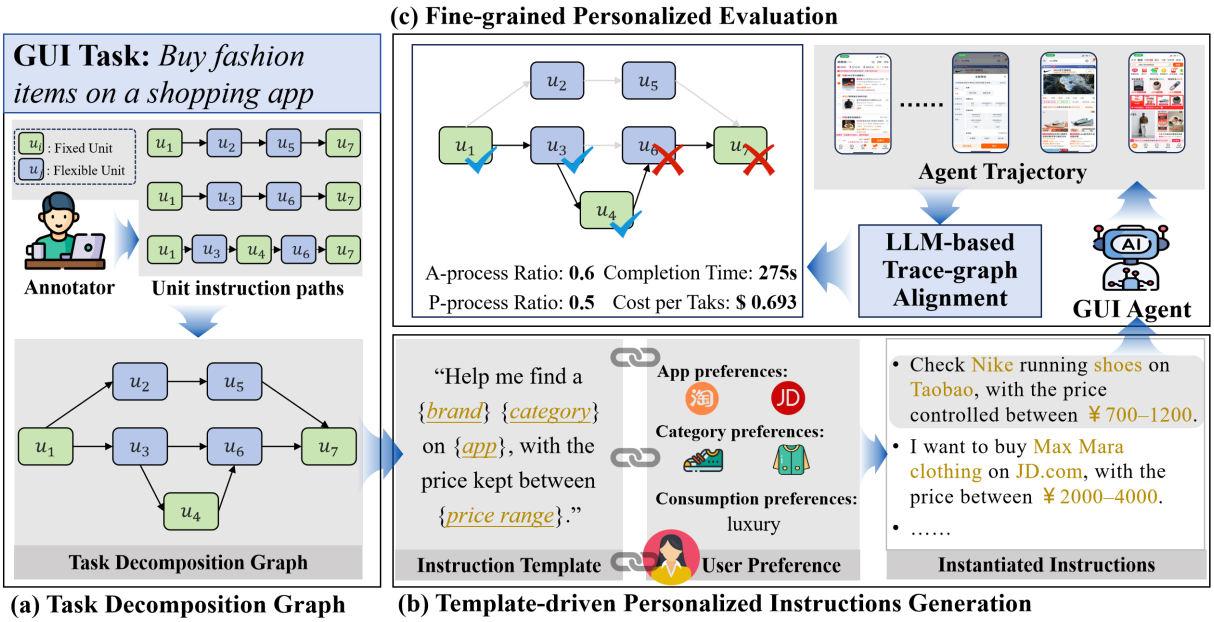


Figure 2: Benchmark framework of PSPA-BENCH. (a) **Task decomposition graph**: the task is decomposed into a directed acyclic graph of unit instructions, where fixed nodes denote universal steps and flexible nodes denote user-specific requirements. (b) **Template-driven personalized instruction generation**: the TDG is used to construct task templates, which are then instantiated with user preferences to generate personalized instructions. (c) **Fine-grained personalized evaluation**: The trace-graph alignment is used to compute APR (A-Progress Ratio) and PPR (P-Progress Ratio) for fine-grained evaluation of both immediate and long-term performance.

Task difficulty control. To control task difficulty, we vary instruction templates along two dimensions: (i) *Complexity level* (low, medium, high), based on the number of unit instructions in the corresponding TDG. (ii) *Clarity level* (high, medium, low), which governs how explicitly the instruction communicates its intent. We provide examples of templates in personalized scenarios in Appendix F

2.3 Fine-grained Personalized Evaluation

We evaluate agents along two complementary objectives in fine-grained level (Section 1): (i) an *immediate objective*: accurately and efficiently executing the current instruction, and (ii) a *long-term objective*: adapting to user-specific preferences over long-term use.

Let the trace be $\mathcal{T} = (a_1, \ldots, a_L)$, where a_i denotes the i-th action. Since a TDG may contain multiple valid source-to-sink paths, we identify the path $P \subseteq \mathcal{G}$ that best matches the trace using a two-stage alignment procedure. First, we define an LLM-based alignment method $m : \mathcal{T} \to \mathcal{U} \cup \{\varnothing\}$ that assigns each action either to the unit instruction it fulfills or to $\varnothing$ if it does not correspond to any instruction. Second, we select the optimal path P^* from all valid paths $\mathcal{P}$ in the TDG by maximizing the number of matched unit instructions:

$$P^* = \arg\max_{P \in \mathcal{P}} \sum_{u \in P} \mathbb{1}\left[\exists\, a_i \in \mathcal{T} : m(a_i) = u\right] \tag{1}$$

where $\mathbb{1}[\cdot]$ is the indicator function. In case of ties (multiple paths achieving the same maximum match count), we apply the following tie-breaking rules in order: (i) prefer paths with more completed *flexible* nodes $\mathcal{U}_p$, as they better reflect personalization; (ii) prefer shorter paths (fewer total nodes), favoring efficiency; (iii) if ties persist, select the path whose matched nodes appear earliest in the trace, reflecting the agent's primary execution intent. Our evaluation begins by aligning the agent's execution trace with the corresponding TDG. The resulting *trace–graph* alignment indicates (i) which unit instructions were successfully completed and (ii) where failures occurred, with attribution to fixed nodes $\mathcal{U}_f$ (universal steps) and flexible nodes $\mathcal{U}_p$ (preference-sensitive steps).

	Immediate	Long-term
Performance	APR PPR	ΔAPR ΔPPR
Efficiency	CT CPT	ΔCT ΔCPT

Figure 3: Evaluation metrics taxonomy of PSPA-BENCH.

Building on this, we assess these objectives along two metric dimensions (Figure 3):

- **Performance** measures execution quality at the unit-instruction level: **A-process Ratio (APR)**: completion ratio over all unit instructions on the executed path. **P-process Ratio (PPR)**: completion ratio restricted to flexible nodes $\mathcal{U}_p$, directly reflecting personalization.

- **Efficiency** quantifies the resources required for task execution: **Completion Time (CT)**: measured as the average elapsed time needed to finish a task. **Cost Per Task (CPT)**, which denotes the average monetary cost (USD) of completing a task, estimated from the number of model tokens consumed.

Each metric is paired with an incremental variant 'Δ' that measures improvements over time, thereby capturing the agent's ability to adapt. For example, ΔAPR = $\text{APR}_{T_2} - \text{APR}_{T_1}$ denotes performance gains between task T_1 and T_2, with analogous increments defined for PPR, CT, and CPT. Implementation details of evaluation are provided in Appendix E.

2.4 Comparison Methods

As shown in Table 1, PSPA-BENCH benchmarks a wide range of baselines and recent agent frameworks, summarized along five dimensions: **Perception**, **Reasoning**, **Memory**, **Evolution**, and **Collaboration**. **Perception** refers to how agents sense the GUI, either via accessibility (A11y) trees or visual inputs. **Reasoning** covers planning and reflection. **Memory** distinguishes temporary memory (within a task) from persistent memory (reused across tasks). **Evolution** considers whether behaviors or toolkits can be updated, and **Collaboration** characterizes the number and roles of agents.

For the **LLM category**, LLM-Base is a standard LLM (e.g., GPT-4o-mini). LLM+A11y augments it with A11y trees, while LLM+A11y+SoM adds set of marks (SoM) in screenshots for stronger visual grounding. LLM+A11y+Thinking further enhances reasoning with advanced models (e.g., GPT-O3-Mini). For **general agent frameworks**, we include ReAct (Yao et al., 2023), which interleaves reasoning and action; Self-Refine (Madaan et al., 2023), which iteratively improves via feedback; and Reflexion (Shinn et al., 2023), which uses reflection memory across tasks. For the **GUI agent frameworks**, PSPA-BENCH evaluates four representatives: AppAgent (Zhang et al., 2025b), which uses multimodal grounding through GUI parsing and screenshots; M3A (Rawles et al., 2025), a GUI-specialized ReAct; Mobile-Agent V2 (Wang et al., 2024a), a multi-agent system with planning, decision, and reflection; and Mobile-Agent E (Wang et al., 2025), which extends V2 with hierarchical collaboration, persistent memory, and self-evolution.

3 Empirical Results

3.1 Experiments on Immediate Objective

We evaluate each method's ability to execute a single personalized instruction effectively and efficiently. Under high-clarity settings and three task complexity levels (low, medium, high), we report unit-level metrics (APR, PPR) and efficiency metrics (CT, CPT) in Table 2. The results show clear performance differences across model classes, revealing the capabilities needed for personalized task execution.

(1) **Perception is the foundation for personalized instruction execution.** The drastic improvement from LLM-Base (APR = 0.002, PPR = 0) to +A11y (APR = 0.625, PPR = 0.586 at low complexity) highlights that access to GUI context is indispensable. Personalized tasks often rely on dynamic, user-specific content, such as preferred items, recent activity, or customized filters, that cannot be interpreted through language alone. Perception provides the necessary grounding, enabling the agent to map abstract instructions to the actual UI elements visible on the screen. Without this capability, the agent is effectively blind to the task's personalized components.

(2) **Reasoning is essential for resolving ambiguity and executing complex personalized behaviors.** Perception allows the agent to observe, but reasoning allows it to decide. Many personalized instructions contain implicit preferences, conditional steps, or multi-hop objectives (e.g., "find a deal that matches your usual budget and dietary preferences"). Reasoning-enhanced models such as +A11y+Thinking and ReAct show significant gains over perception-only agents as complexity increases. For instance, ReAct achieves APR = 0.503 and PPR = 0.468 at medium complexity, outperforming simpler baselines. These models can handle flexible execution paths, maintain coherence across steps, and adapt to user-specific constraints

Table 1: Comparison of evaluated methods in PSPA-BENCH. The table presents baseline LLMs, general agent frameworks, and GUI agent frameworks, assessed across five key dimensions: Perception, Reasoning, Memory, Evolution, and Collaboration.

Method		**Perception**		**Reasoning**		**Memory**		**Evolution**		**Collaboration**	
Type	**Name**	Text	Vision	Planning	Reflection	Temporary	Persistent	Rules	Toolkits	Single	Multiple
LLM	Base		✓			✓				✓	
	+A11y	✓				✓				✓	
	+A11y+SoM	✓	✓			✓				✓	
	+A11y+Thinking	✓		✓		✓				✓	
General Agent Framework	ReAct	✓	✓	✓		✓				✓	
	Self-Refine	✓	✓		✓	✓					
	Reflexion	✓	✓	✓	✓	✓	✓				✓
GUI Agent Framework	AppAgent	✓	✓	✓		✓				✓	
	M3A	✓	✓	✓		✓				✓	
	Mobile-AgentV2		✓	✓	✓	✓	✓				✓
	Mobile-Agent E		✓	✓	✓	✓	✓	✓	✓		✓

Table 2: Mean evaluation results (APR, PPR, CT, CPT) are reported across five scenarios under high-clarity conditions with varying task complexity (low, medium, high). The results indicate each method's effectiveness in achieving the immediate objective of personalized GUI tasks.

Method	Setting: Clarity fixed at middle, varying complexity											
	Complexity: low				Complexity: middle				Complexity: high			
	APR↑	PPR↑	CT↓	CPT↓	APR↑	PPR↑	CT↓	CPT↓	APR↑	PPR↑	CT↓	CPT↓
LLM-Base	0.002	0	236.7	0.612	0.001	0	256.6	0.501	0.002	0	293.4	0.816
+A11y	0.625	0.586	176.7	0.056	0.431	0.449	189.0	0.038	0.364	0.356	229.3	0.062
+A11y+SoM	0.643	0.596	243.6	0.671	0.463	0.461	254.3	0.525	0.373	0.372	293.3	0.930
+A11y+Thinking	0.674	0.624	197.7	0.673	0.466	0.464	233.0	0.726	0.382	0.378	274.8	0.884
ReAct	0.683	0.618	217.0	0.752	0.503	0.468	243.0	0.616	0.414	0.403	316.8	0.761
Self-Refine	0.596	0.542	249.1	0.758	0.438	0.441	731.4	0.796	0.308	0.339	845.3	0.931
Reflexion	0.665	0.611	250.7	0.657	0.465	0.466	332.5	0.694	0.438	0.406	464.5	0.776
AppAgent	0.654	0.592	218.7	0.329	0.469	0.495	227.8	0.337	0.373	0.402	292.6	0.502
M3A	0.685	0.623	317.6	0.951	0.521	0.485	340.7	1.010	0.460	0.403	450.6	1.507
Mobile-Agent V2	0.674	0.618	517.6	1.017	0.532	0.533	578.9	1.364	0.435	0.415	659.1	1.414
Mobile-Agent E	0.695	0.621	532.7	1.084	0.546	0.539	587.5	1.401	0.468	0.435	716.7	1.692

— all of which are critical in the context of personalization.

(3) **Personalization introduces a stronger trade-off between execution accuracy and efficiency.** While Mobile-Agent E achieves the highest APR (0.695) and PPR (0.621), it also exhibits the slowest execution (CT = 716.7 at high complexity), reflecting a strategy that prioritizes accuracy through exhaustive interaction. This behavior may stem from the need to explore interface options and disambiguate user preferences on the fly. In contrast, AppAgent adopts a more efficient but less exhaustive approach (CT = 292.6 at high complexity), with slightly lower accuracy. In personalized settings, where goals are less explicit and vary across users, striking the right balance between precision and efficiency becomes more challenging than in generic task settings.

3.2 Experiments on long-term Objective

We evaluate each method's ability to support long-term adaptation to user preferences. To simulate sustained interaction, agents first perform tasks of medium complexity and clarity to accumulate experience, and are then evaluated against a baseline without prior experience. We report relative changes in ΔAPR, ΔPPR, ΔCT, and ΔCPT, defined as the performance difference with and without experience accumulation. Evaluation is conducted under two settings: (i) increasing task complexity with fixed clarity, and (ii) increasing ambiguity with fixed complexity. Table 3 presents the results, demonstrating the importance of memory and experience processing for long-term personalization.

Formally, let M denote an evaluation metric (e.g., APR, PPR, CT, CPT). For a user u, we de-

Table 3: Mean values of evaluation metrics (ΔAPR, ΔPPR, ΔCT, ΔCPT) are reported across five scenarios under two settings: (i) high clarity with varying task complexity (low, medium, high), and (ii) medium complexity with varying task clarity (low, medium, high). All methods first perform tasks with medium complexity and clarity to acquire initial experience. The results reflect each method's ability to adapt to user preferences in long-term personalized GUI tasks. (Note: For readability, ΔAPR, ΔPPR and ΔCPT values are scaled by 100.)

Method	Setting (i): Clarity fixed at high, varying complexity											
	Complexity: low				Complexity: middle				Complexity: high			
	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓
LLM-Base	0	0	9.28	0.20	0	0	-17.68	0.73	0	0	-19.18	0.94
+A11y	0.66	-0.58	-12.73	-0.63	-0.39	0.05	-2.72	0.42	0.22	-0.72	-8.31	-0.27
+A11y+SoM	-0.09	0.57	12.01	-0.03	0.18	-0.91	4.30	0.66	-0.87	0.90	-18.63	0.62
+A11y+Thinking	-0.39	-0.80	7.37	-0.12	-0.76	-0.01	-18.62	0.82	-0.48	0.33	-7.53	0.04
ReAct	0.09	-0.63	-18.78	0.55	0.88	0.79	-3.92	0.84	-0.82	-0.61	-18.19	0.35
Self-Refine	-0.22	-0.46	13.15	-0.29	-0.44	0.09	-14.36	0.60	-0.85	0.97	10.89	-0.60
Reflexion	4.53	3.71	-38.35	-2.46	3.54	3.29	-40.62	-2.77	2.90	1.25	-26.76	-1.87
AppAgent	-0.38	-0.35	-9.18	0.28	0.77	-0.06	-15.22	0.43	0.52	0.12	10.84	-0.10
M3A	0.05	-0.14	18.98	-0.78	-0.94	0.27	-7.43	0.02	0.82	-0.50	3.58	0.51
Mobile-Agent V2	4.84	3.26	-36.48	-3.70	4.54	4.02	-41.45	-2.43	3.41	1.04	-26.02	-3.58
Mobile-Agent E	6.96	4.08	-62.08	-4.06	6.72	3.68	-57.02	-3.41	2.13	2.44	-35.51	-1.94

Method	Setting (ii): Complexity fixed at middle, varying clarity											
	Clarity: low				Clarity: middle				Clarity: high			
	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓	ΔAPR↑	ΔPPR↑	ΔCT↓	ΔCPT↓
LLM-Base	0	0	17.72	-0.35	0	0	-5.45	0.94	0	0	-0.11	0.40
+A11y	-0.43	-0.93	4.38	-0.01	-0.90	-0.44	16.33	-0.52	-0.71	-0.02	19.43	-0.52
+A11y+SoM	0.34	0.52	-10.49	0.46	-0.26	0.26	5.34	-0.07	-0.82	0.67	-7.17	0.63
+A11y+Thinking	-0.92	0.18	-7.10	0.97	0.02	-0.55	-5.81	0.65	0.38	-0.23	17.47	-0.72
ReAct	-0.32	-0.77	16.99	-0.75	-0.48	0.32	-12.69	0.11	0.06	-0.52	-16.28	0.79
Self-Refine	0.80	0.27	6.44	-0.30	0.45	0.79	-15.48	0.56	0.28	-0.83	-13.53	0.80
Reflexion	4.23	4.58	-37.50	-2.75	3.70	2.75	-39.54	-2.78	2.11	1.71	-31.34	-1.33
AppAgent	-0.35	0.49	5.99	-0.70	0.32	0.14	-16.25	0.26	-0.47	-0.51	18.92	-0.21
M3A	0.78	0.26	11.79	-0.65	0.15	-0.01	-12.19	0.44	-0.44	-0.95	-5.82	0.65
Mobile-Agent V2	4.27	3.15	-36.17	-2.95	3.72	3.62	-43.49	-2.1	2.26	1.61	-25.23	-1.31
Mobile-Agent E	6.54	7.15	-63.81	-3.08	5.13	4.94	-53.44	-2.28	5.55	4.08	-40.59	-2.75

note M_u^{with} as the metric value after experience accumulation and $M_u^{\text{w/o}}$ as the metric value without accumulation. The relative improvement is defined as: $\Delta M = \frac{1}{N}\sum_{u=1}^{N}(M_u^{\text{with}} - M_u^{\text{w/o}})$, where N is the number of evaluation tasks. For APR and PPR, a positive ΔM indicates better task completion, while for CT and CPT, a negative ΔM indicates improved efficiency. Table 3 reports the results, from which we draw two key findings:

(1) Persistent memory is the foundation for long-term adaptation. Models without persistent memory, such as ReAct and M3A, show little to no improvement, and in many cases even regress (e.g., negative ΔAPR and ΔPPR across most settings). In contrast, methods with persistent memory, e.g., Reflexion and Mobile-Agent V2, achieve consistent gains. This confirms that retaining and reusing past experience is critical for adapting to personalized, evolving user behaviors.

(2) Self-evolution enhances long-term adaptation. Persistent memory provides the foundation, but Mobile-Agent E goes further by enabling self-evolution: it can update execution rules and create shortcuts from accumulated experience in its persistent memory. This capability yields the strongest gains overall (e.g., ΔAPR = +6.72, ΔPPR = +3.68 at medium complexity). Compared to Mobile-Agent V2, which mainly reuses stored memory, Mobile-Agent E actively improves its strategy over time, leading to better generalization to new but related personalized tasks.

3.3 Ranking of Methods on Personalized GUI Benchmark

To further assess method performance on PSPA-BENCH, we rank all methods with respect to both the immediate and long-term objectives. As shown in Fig. 4, for the immediate objective (left), there is a clear trade-off between execution accuracy (APR, PPR) and efficiency (CT, CPT): Mobile-Agent E achieves the highest accuracy but incurs substantial execution cost, whereas AppAgent and LLM+A11y are faster but less accurate. This suggests that GUI agents must balance precision and efficiency. For the long-term objective (right), methods with persistent memory, e.g., Reflexion,

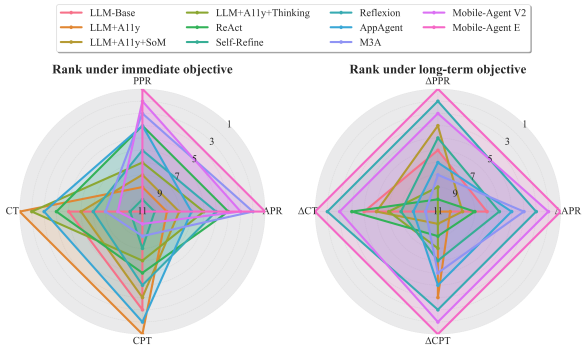


Figure 4: Radar plots showing the comparative ranks of different methods under two objectives. **Left:** Immediate objective, evaluated with APR, PPR, CT, and CPT. **Right:** Long-term objective, evaluated with ΔAPR, ΔPPR, ΔCT, and ΔCPT. Each axis indicates the rank of a method on the corresponding metric, with lower values representing better performance.

Mobile-Agent V2, and especially Mobile-Agent E, achieve consistently higher improvements across ΔAPR and ΔPPR. This confirms that retaining and reusing past experience is necessary for adapting to evolving user preferences, with stronger memory mechanisms yielding greater gains.

4 Related Works

4.1 Smartphone GUI Agent

Smartphone GUI agents (Zhang et al., 2025a; Liu et al., 2025a) automate tasks by interpreting visual and textual elements on mobile screens. Prior work (Nguyen et al., 2025) has explored several design dimensions. For perception, current agents rely on structured UI data such as A11y trees (Rawles et al., 2025; Wen et al., 2024), or use vision-based methods for richer context (Zhang et al., 2025b; Gou et al., 2025). For reasoning and planning, frameworks like ReAct (Yao et al., 2023) and Reflexion (Shinn et al., 2023) have been integrated into M3A (Rawles et al., 2025) and Mobile-Agent V2 (Wang et al., 2024a) to handle complex tasks, while GUI-Explorer (Xie et al., 2025) uses prior knowledge of app operations. Recent studies also focus on learning from execution experience: Mobile-Agent E (Wang et al., 2025) and AppAgentX (Jiang et al., 2025) summarize repetitive low-level actions into high-level shortcuts to improve efficiency. Despite recent progress, most agents still target general automation, leaving personalization underexplored. Recently, FingerTip (Chen et al., 2025a) introduces a benchmark for evaluating personalization and shows that existing agents perform poorly on such tasks. However, its coarse-grained metrics may lose important details when assessing adaptation to diverse preferences or long-term use.

4.2 Personalized LLM Agent

Personalized LLM agents (Dong et al., 2023; Tseng et al., 2024) aim to tailor assistance to individual preferences. Approaches fall into two paradigms: fine-tuning and training-free methods. Fine-tuning approaches, such as Personalized Soups (Jang et al., 2023) and (Liu et al., 2025b), adapt model parameters using explicit user feedback. While effective in capturing preferences across multiple users, these methods require large amounts of data and significant computational resources. Training-free methods instead keep the base model fixed, incorporating personalization via prompt engineering (Zhang et al., 2025c; Gao et al., 2024), retrieval-augmented generation (RAG) (Salemi et al., 2024b,a), or external memory (Wang et al., 2024b; Zhong et al., 2024). For example, CIPHER (Gao et al., 2024) uses user edits to infer preferences, while EMG-RAG (Wang et al., 2024b) organizes user-specific information with editable memory graphs. While effective in static tasks such as travel planning, applying these strategies to dynamic mobile GUI environments remains an open challenge.

5 Conclusion

We presented PSPA-BENCH, a benchmark for evaluating personalization in smartphone GUI agents. PSPA-BENCH introduces the TDG to explicitly separate universal and preference-sensitive steps, enabling both realistic personalized instruction generation and fine-grained evaluation. Covering 10 scenarios, 28 apps, and 100 user personas, PSPA-BENCH provides a large-scale, privacy-compliant testbed. Experiments with 11 representative models show that existing agents struggle on personalized tasks, revealing substantial room for improvement. At the same time, our analysis highlights that personalization requires agents to go beyond generic automation, emphasizing grounding in GUI context, structured reasoning, and mechanisms for accumulating and evolving user-specific experience. By providing a principled framework and comprehensive evaluation, PSPA-BENCH lays the foundation for advancing research on personalized GUI agents and offers a path toward more adaptive and user-centered mobile AI systems.

Limitations

While PSPA-BENCH provides a broad benchmark for evaluating personalization in smartphone GUI agents, several limitations should be noted. First, although the benchmark covers 22 apps across 10 representative scenarios, this is still a limited subset of the vast mobile application ecosystem, and the findings may not fully generalize to other domains such as healthcare, finance, or specialized professional tools. Second, the user personas and preferences in PSPA-BENCH are synthetically generated based on predefined templates rather than collected from real users, which may not fully capture the complexity, inconsistency, and evolving nature of authentic user behaviors. Third, the current benchmark is primarily designed for Android devices, limiting its direct applicability to other platforms such as iOS, where UI conventions and interaction patterns differ. Finally, the task decomposition graphs are manually constructed by domain experts, which, while ensuring quality, may not cover all possible task variations or edge cases that arise in real-world usage. These limitations also point to promising directions for future work, including extending the benchmark to broader application domains and platforms, incorporating real user data while preserving privacy, and developing automated methods for TDG construction.

References

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. 2025a. SPA-BENCH: a comprehensive benchmark for smartphone agent evaluation. In *International Conference on Learning Representations*.

Yurun Chen, Xueyu Hu, Keting Yin, Juncheng Li, and Shengyu Zhang. 2025b. AEIA-MN: evaluating the robustness of multimodal LLM-powered mobile agents against active environmental injection attacks. *arXiv preprint arXiv:2502.13053*.

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Liujianfeng Liujianfeng, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. 2024. Mobile-bench: An evaluation benchmark for LLM-based mobile agents. In *Annual Meeting of the Association for Computational Linguistics*, pages 8813–8831.

Xin Luna Dong, Seungwhan Moon, Yifan Ethan Xu, Kshitiz Malik, and Zhou Yu. 2023. Towards next-generation intelligent assistants leveraging LLM techniques. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5792–5793.

European Parliament and Council of the European Union. 2016. Regulation (eu) 2016/679 (general data protection regulation). Official Journal of the European Union.

Ge Gao, Alexey Taymanov, Eduardo Salinas, Paul Mineiro, and Dipendra Misra. 2024. Aligning LLM agents by learning latent preference from user edits. In *Advances in Neural Information Processing Systems*.

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

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shenzhi Wang, Xinchen Xu, Shuofei Qiao, Zhaokai Wang, Kun Kuang, Tieyong Zeng, Liang Wang, and 10 others. 2025. OS Agents: A survey on MLLM-based agents for computer, phone and browser use. In *Annual Meeting of the Association for Computational Linguistics*, pages 7436–7465.

Joel Jang, Seungone Kim, Bill Yuchen Lin, Yizhong Wang, Jack Hessel, Luke Zettlemoyer, Hannaneh Hajishirzi, Yejin Choi, and Prithviraj Ammanabrolu. 2023. PERSONALIZED SOUPS: Personalized large language model alignment via post-hoc parameter merging. In *Advances in Neural Information Processing Systems Workshop Adaptive Foundation Models*.

Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, and Chi Zhang. 2025. AppAgentX: Evolving GUI agents as proficient smartphone users. *arXiv preprint arXiv:2503.02268*.

J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174.

Juyong Lee, Dongyoon Hahm, June Suk Choi, W. Bradley Knox, and Kimin Lee. 2024a. MobileSafetyBench: Evaluating safety of autonomous agents in mobile device control. *arXiv preprint arXiv:2410.17520*.

Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steven Y. Ko, Sangeun Oh, and Insik Shin. 2024b. MobileGPT: Augmenting LLM with human-like app memory for mobile task automation. In *Annual International Conference on Mobile Computing and Networking*, pages 1119–1133.

Huoran Li, Xuan Lu, Xuanzhe Liu, Tao Xie, Kaigui Bian, Felix Xiaozhu Lin, Qiaozhu Mei, and Feng Feng. 2015. Characterizing smartphone usage patterns from millions of Android users. In *ACM Internet Measurement Conference*, pages 459–472.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Yaxuan Guo, Han Xiao, Weifeng Lin, Yuxiang Chai, Yue Han, Shuai Ren, Hao Wang, Xiaoyu Liang, Wenhao Wang, Tianze Wu, Linghao Li, Hao Wang, Guanjing Xiong, Yong Liu, and Hongsheng Li. 2025a. LLM-powered GUI agents in phone automation: Surveying progress and prospects. *arXiv preprint arXiv:2504.19838*.

Jiongnan Liu, Yutao Zhu, Shuting Wang, Xiaochi Wei, Erxue Min, Yu Lu, Shuaiqiang Wang, Dawei Yin, and Zhicheng Dou. 2025b. LLMs + Persona-Plug = personalized LLMs. In *Annual Meeting of the Association for Computational Linguistics*, pages 9373–9385.

Yuheng Lu, Qian Yu, Hongru Wang, Zeming Liu, Wei Su, Yanping Liu, Yuhang Guo, Maocheng Liang, Yunhong Wang, and Haifeng Wang. 2025. TransBench: Breaking barriers for transferable graphical user interface agents in dynamic digital environments. In *Findings of the Association for Computational Linguistics: ACL*, pages 12464–12478, Vienna, Austria.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, and 11 others. 2025. GUI agents: A survey. In *Findings of the Association for Computational Linguistics: ACL*, pages 22522–22538.

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, and 16 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 E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2025. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*.

Alireza Salemi, Surya Kallumadi, and Hamed Zamani. 2024a. Optimization methods for personalizing large language models through retrieval augmentation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 752–762.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. 2024b. LaMP: When large language models meet personalization. In *Annual Meeting of the Association for Computational Linguistics*, pages 7370–7392.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Yuan Tian, Ke Zhou, Mounia Lalmas, and Dan Pelleg. 2020. Identifying tasks from mobile app usage patterns. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2357–2366.

Yu-Min Tseng, Yu-Chao Huang, Teng-Yun Hsiao, Wei-Lin Chen, Chao-Wei Huang, Yu Meng, and Yun-Nung Chen. 2024. Two tales of persona in LLMs: A survey of role-playing and personalization. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 16612–16631.

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. In *Advances in Neural Information Processing Systems*.

Zheng Wang, Zhongyang Li, Zeren Jiang, Dandan Tu, and Wei Shi. 2024b. Crafting personalized agents through retrieval-augmented generation on editable memory graphs. In *Conference on Empirical Methods in Natural Language Processing*, pages 4891–4906.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-Agent-E: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*.

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 *Annual International Conference on Mobile Computing and Networking*, pages 543–557.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. In *Annual Meeting of the Association for Computational Linguistics*, pages 5650–5667.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the weakness of large language model agents within a complex Android environment. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6061–6072.

Qinglong Yang, Haoming Li, Haotian Zhao, Xiaokai Yan, Jingtao Ding, Fengli Xu, and Yong Li. 2025. FingerTip 20K: A benchmark for proactive and personalized mobile LLM agents. *arXiv preprint arXiv:2507.21071*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Chaoyun Zhang, Shilin He, Jiaxu Qian, Bowen Li, Liqun Li, Si Qin, Yu Kang, Minghua Ma, Guyue Liu, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2025a. Large language model-brained GUI agents: A survey. *Transactions on Machine Learning Research*.

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

Weizhi Zhang, Xinyang Zhang, Chenwei Zhang, Liangwei Yang, Jingbo Shang, Zhepei Wei, Henry Peng Zou, Zijie Huang, Zhengyang Wang, Yifan Gao, Xiaoman Pan, Lian Xiong, Jingguo Liu, Philip S. Yu, and Xian Li. 2025c. PersonaAgent: When large language model agents meet personalization at test time. *arXiv preprint arXiv:2506.06254*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing large language models with long-term memory. In *AAAI Conference on Artificial Intelligence*, pages 19724–19731.

Appendix

A Comparison with Other Benchmarks

Table 4 summarizes the comparison between PSPA-BENCH and existing benchmarks for smartphone GUI agents. General-purpose GUI benchmarks such as AndroidArena, MobileAgentBench, and AndroidWorld are primarily designed to evaluate agents' basic task-execution capabilities. While they offer large-scale task datasets, they lack support for personalization—there are no user profiles, preferences, or long-term evaluation protocols. SPA-Bench improves realism by leveraging physical devices and third-party apps, but it still does not consider personalization.

Personalized GUI benchmarks, such as LearnGUI and FingerTip, take steps toward user-aware evaluation. However, LearnGUI is limited to simulated environments without physical devices or third-party apps, reducing ecological validity. FingerTip collects real interactions on physical devices with user profiles, but the user preferences are fixed and static, limiting its ability to capture the dynamic and evolving nature of personalization in real-world use.

In contrast, PSPA-BENCH introduces several design choices that address these gaps:

1. **Realistic Environment and App diversity.** Unlike many benchmarks that rely on simulated environments or limited app support, PSPA-BENCH is deployed on physical smartphones and spans 22 third-party apps, covering diverse real-world scenarios such as e-commerce, food delivery, and travel. This setting ensures that agents are evaluated under realistic constraints (e.g., UI latency, pop-ups, network variability), increasing ecological validity and practical relevance.

2. **Dynamic Modeling of User Preferences.** Most existing personalized benchmarks(e.g., FingerTip) either fix user preferences or omit them entirely. In contrast, PSPA-BENCH models both short-term and long-term user preferences to generate realistic, user-aligned task instructions. Our user profiles and behaviors are not static; instead, they evolve over time, enabling the study of adaptive personalization strategies that reflect real usage patterns.

3. **Multi-Dimensional Task Difficulty.** While previous benchmarks typically define difficulty via task execution length, PSPA-BENCH introduces a richer difficulty model. We consider multiple dimensions, including both task execution length and instruction ambiguity. This enables a more nuanced evaluation of an agent's ability to reason about and adapt to varying personalized tasks.

4. **Fine-Grained and Incremental Personalized Evaluation Metrics.** Existing evaluation metrics such as success signal or task completion ratio are often coarse-grained and overly reliant on fixed action sequences. PSPA-BENCH introduces process-oriented evaluation via task decomposition graphs, enabling step-by-step tracking of progress toward user-specific goals. Furthermore, we support incremental evaluation metrics to evaluate the performance changes of agents in long-term personalized use, which is missing in existing benchmarks.

To summarize, PSPA-BENCH combines realistic environments, dynamic user modeling, and fine-grained evaluation to address key gaps in existing benchmarks for personalized GUI agents.

B Definition of Personalized GUI Agent Task

Let Q be the space of GUI task instructions, $E = (\mathcal{S}, \mathcal{A})$ be a GUI environment that contains a set $\mathcal{S} = \{s_i\}$ of observable interface states and a set $\mathcal{A} = \{a_i\}$ of executable GUI actions for state transitions. The definition of Personalized GUI Agent Task is given as follows:

Definition B.1 ***Personalized GUI Agent Task.*** *For a user u, the profile ρ_u induces a conditional distribution $\mathcal{D}_Q^{\rho_u}$ over the instruction space Q. At each time step t, a personalized instruction q_t^u is sampled from $\mathcal{D}_Q^{\rho_u}$. The agent operates in the GUI environment $E = (\mathcal{S}, \mathcal{A})$ under a user-specific policy π_u with two objectives: (i)* ***Immediate objective:*** *generate an action sequence that drives the environment E toward the target states satisfying the current instruction q_t^u; (ii)* ***Long-term objective:*** *accumulate execution experience $\mathcal{H}_u$ to update π_u, thereby improving expected performance on future personalized instruction sequences sampled from $\mathcal{D}_Q^{\rho_u}$.*

Table 4: Comparison of General and Personalized GUI Benchmarks

Type	Dataset & Benchmark	Environment with physical phones	Third-part apps	User profiles	User preferences	Task instructions	Difficulty dimensions	Fine-grained progress evaluation	Incremental evaluation
General GUI benchmark	AndroidArena	✗	✗	✗	✗	221	✗	✓	✗
	LlamaTouch	✗	✓	✗	✗	495	Step length	✗	✗
	MobileAgentBench	✗	✗	✗	✗	100	Step length	✗	✗
	AndroidWorld	✗	✗	✗	✗	∞	Step length	✗	✗
	SPA-bench	✓	✓	✗	✗	340	Step length	✗	✗
Personalized GUI benchmark	LearnGUI	✗	✗	✗	✗	2353	✗	✗	✗
	FingerTip	✓	✓	91	Fixed	21437	Step length	✗	✗
	PSPA-BENCH	✓	✓	100	Dynamic	12855	Step length & instruction clarity	✓	✓

C Construction Task Decomposition Graph

We illustrate the constructed task decomposition graph with four personalized scenarios: shopping, dining, navigation, and travel. In the these task decomposition graphs, the nodes with green color refer to fixed type, while nodes with blue color refers to personalized type.

D Instruction Instantiation

Once a template is selected, we instantiate it by filling each slot with concrete values drawn from the user's preference distribution. Given a user profile ρ, which contains a set of preferences, each slot l is filled by sampling a value $v_l \sim D^{\rho}_{\tau(l)}$, where D^{ρ}_{τ} is a distribution over admissible values conditioned on ρ. The instantiation function $f(\mathsf{TP}_{\mathcal{G}}, \{v_l\}_{l\in\mathcal{L}}) \mapsto I$ produces a concrete instruction I by replacing each slot with the sampled value, while ensuring that the chosen values respect the execution dependencies in $\mathcal{G}$. In this framework, We model user preferences as a combination of: (i) *Long-term preferences*, capturing stable patterns over time (e.g., favoring eco-friendly brands), with weight $w_{\text{long}}(t)$ modeled by a logarithmic growth function. (ii) *Short-term preferences*, capturing transient or contextual interests (e.g., recent searches or trends), with weight $w_{\text{short}}(t)$ modeled by a sinusoidal function. At each time step t, we compute the probability of sampling from each component as: $P_{\text{long}}(t) = w_{\text{long}}(t)/[w_{\text{long}}(t) + w_{\text{short}}(t)]$, and $P_{\text{short}}(t) = w_{\text{short}}(t)/[w_{\text{long}}(t) + w_{\text{short}}(t)]$, respectively.

These probabilities guide the instantiation process for each slot. For example, the <product_category> slot might be filled with "organic snacks" based on long-term preferences, or "Halloween costumes" based on short-term seasonal trends. This dynamic sampling ensures that the personalized instructions reflect both enduring habits and recent behaviors, rather than being static or randomly constructed. The detailed formulation of preference weights, along with experiments evaluating the quality of the generated instructions, is provided in Appendix D.1 and D.2, respectively.

D.1 Details of Preference Weights Calculation

We now describe the preference weights used to model long-term and short-term user patterns. These formulations balance tractability with behavioral plausibility, so that generated instructions reflect both stable needs and dynamic fluctuations.

Long-Term Preference weights. Long-term preferences capture stable, gradually accumulating needs that reset upon satisfaction (e.g., recurring but not continuous interests). Let

$$t \in \mathbb{Z}_{\geq 0}, \quad L \in \mathbb{Z}_{\geq 0}, \quad C \in \mathbb{Z}_{\geq 1}, \tag{2}$$

denote the current day, the last day of satisfaction, and the cycle length in days, respectively. With a base weight $b \in (0, 1]$ and maximum allowed weight $W_{\max} \in (0, 1]$, we define

$$\Delta(t) = \max\{0,\, t-L\}, \qquad \tau(t) = \Delta(t) \bmod C. \tag{3}$$

The long-term preference weight is given by:

$$w_{\text{long}}(t) = \min\left\{W_{\max},\, b + \frac{\ln(1+\tau(t))(1-b)}{\ln(1+C)}\right\}. \tag{4}$$

The logarithmic form reflects the intuition that the *marginal increase* in preference weight diminishes over time: interest accumulates steadily but slows as it approaches saturation. Resetting $\tau(t)$ upon satisfaction models the release of long-term need after fulfillment. The normalization by $\ln(1 + C)$ guarantees that the weight grows smoothly to a normalized scale within each cycle, while clamping ensures that weights remain

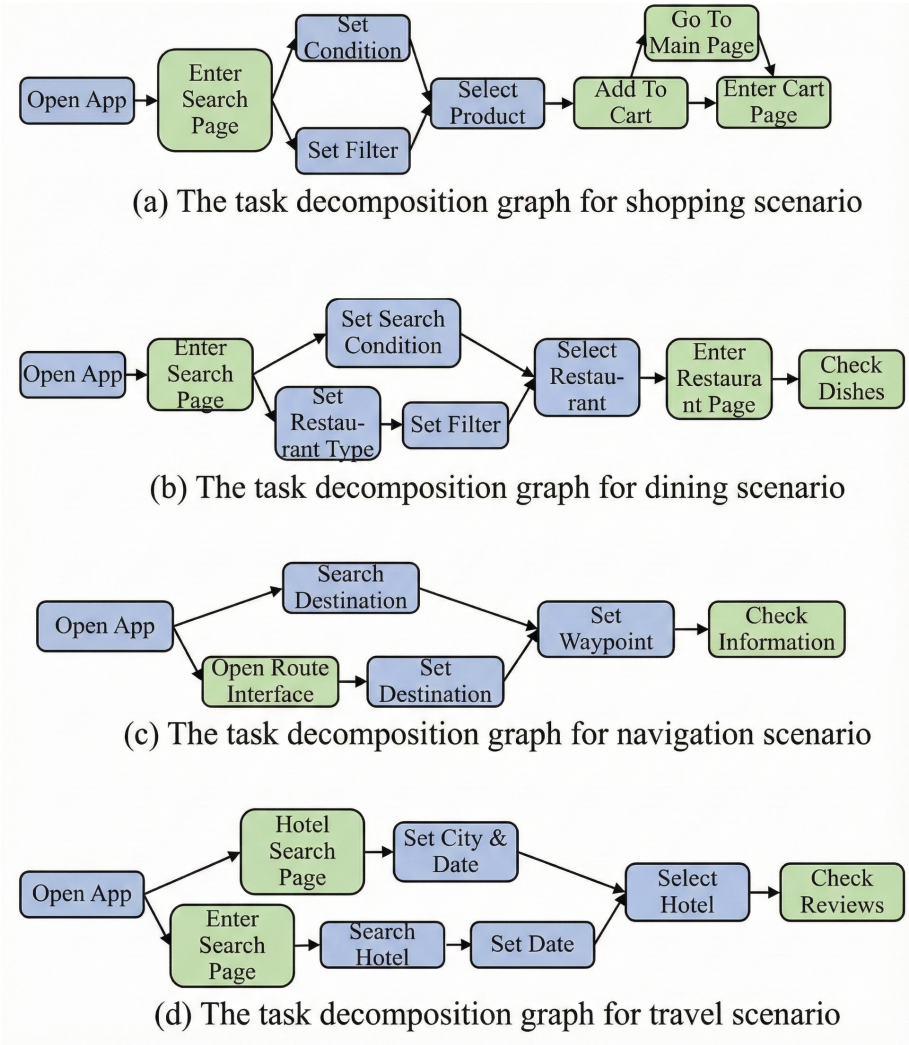


Figure 5: The task decomposition graphs of 4 personalized scenarios: shopping, dining, navigation, and travel. The nodes with green color refer to fixed type, while nodes with blue color refers to personalized type.

bounded by $W_{\max}$. This choice is particularly suitable for modeling durable interests such as dietary habits, weekly routines, or recurring educational needs, which build up gradually until satisfied.

Short-Term Preference weights. Short-term preferences capture immediate, oscillatory needs that fluctuate with context. Let

$$t \in \mathbb{Z}_{\geq 0}, \quad C \in \mathbb{Z}_{\geq 1}, \quad \phi \in \mathbb{R}, \tag{5}$$

denote the current day, the cycle length in days, and a phase offset, respectively. With bounds $0 < w_{\min} < w_{\max} \leq 1$, we define:

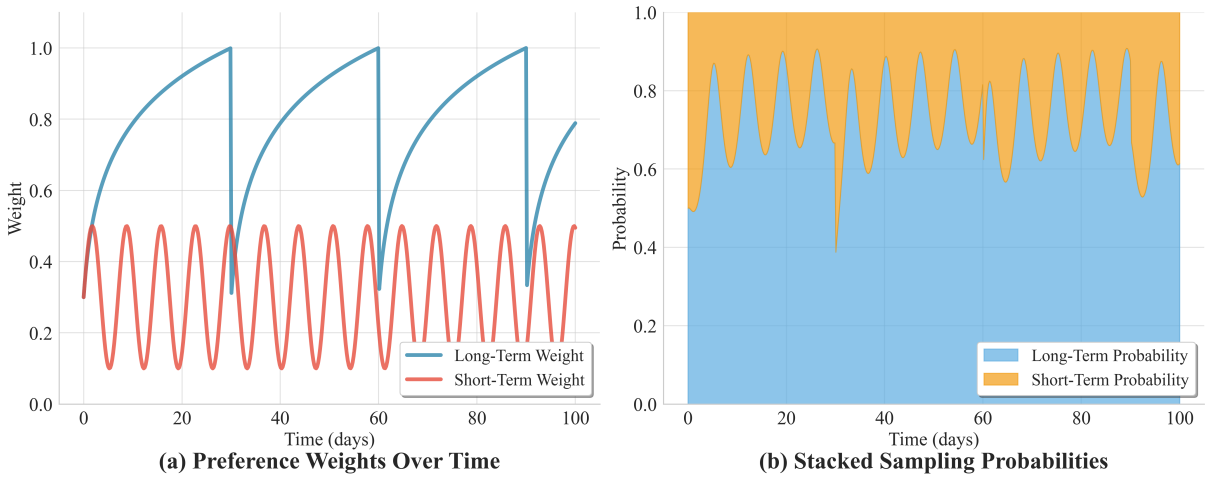
$$w_{\text{short}}(t) = w_{\min} + (w_{\max} - w_{\min}) \cdot \frac{\sin\left(\frac{2\pi}{C}t + \phi\right) + 1}{2}. \tag{6}$$

The sinusoidal structure is chosen for its natural ability to model periodic fluctuations. It ensures smooth transitions between high and low preference intensities, while the parameters $w_{\min}$ and $w_{\max}$ bound the oscillation range. This reflects short-term needs such as daily mood, time-of-day activity cycles, or context-specific demands (e.g., preferring entertainment in the evening versus productivity tools in the morning). Compared to stochastic fluctuations, sinusoidal oscillation avoids unnatural abrupt shifts and instead captures the rhythmic, recurrent nature of short-term preferences.

Normalized Sampling probabilities. Given both weights, we define the probabilities of drawing from long-term and short-term preferences as:

$$P_{\text{long}}(t) = \frac{w_{\text{long}}(t)}{w_{\text{long}}(t) + w_{\text{short}}(t)}, \tag{7}$$

$$P_{\text{short}}(t) = \frac{w_{\text{short}}(t)}{w_{\text{long}}(t) + w_{\text{short}}(t)}. \tag{8}$$



(a) Preference Weights Over Time **(b) Stacked Sampling Probabilities**

Figure 6: (a) **Preference Weights Over Time**: The blue curve represents the long-term preference weight, which gradually increases with time, reflecting the accumulation of stable needs. The red curve depicts the short-term preference weight, which oscillates periodically, reflecting dynamic, context-dependent needs. (b) **Stacked Sampling Probabilities**: The stacked area plot shows the normalized sampling probabilities over time. The blue region corresponds to the probability of sampling the long-term preference, while the orange region corresponds to the probability of sampling the short-term preference. The relative areas of these regions change over time, depending on the weights of each preference.

This normalization yields a valid probability distribution and balances stability with variability: when long-term needs are strong, they dominate; when short-term fluctuations peak, they take precedence. A visual illustration of these dynamics can be found in Figure 6.

D.2 Instruction Quality Evaluation

To rigorously evaluate the quality of generated task instructions, we conducted a human evaluation study across **10 distinct scenarios** involving a total of **100 users**. For each user, we selected the **first 10 generated instructions** from each method and used them as the evaluation material. This design ensures a fair and comparable assessment across methods, while also reflecting the diversity of tasks encountered in practical settings.

We compared three approaches for generating user instructions:

1. **Static Preference.** Each user is assigned a *fixed preference profile* that does not change over time. Instructions are generated by filling task templates strictly according to this static preference. While this captures general user tendencies, it cannot adapt to temporal shifts in user needs.

2. **Random Preference.** At each time step, a *random preference* is assigned to the user without considering historical data. Instructions are then generated from this random preference. Although this produces variety, the lack of temporal or contextual grounding leads to incoherent or irrelevant instructions.

3. **Dynamic Preference Modeling (Ours).** We model user preferences as a distribution combining both *long-term patterns* (stable, accumulated needs) and *short-term patterns* (immediate, oscillating needs). At each time step t, the probability of sampling long-term versus short-term preferences is determined by their respective weights. Task instructions are generated based on the sampled preferences, enabling outputs that capture both the *stable characteristics* of the user and their *short-term dynamics*.

Annotators were presented with **pairs of instruction sets** (10 instructions per set, per user). Each pair was evaluated **blindly**, comparing two methods at a time. Annotators judged which set was better overall, based on two criteria:

- **Plausibility / Realism:** Are the instructions natural and reasonable, resembling tasks that a real user might genuinely perform?

- **Relevance to User Context:** Are the instructions aligned with the user's scenario, needs, and historical behavior?

Annotators were allowed to declare a **Tie** if the two sets were of equal quality. Each pair was evaluated by multiple annotators, and final results were aggregated using majority voting.

As shown in Table 5, the results clearly demonstrate the advantage of **dynamic preference modeling**:

- **Ours vs. Static:** Our method achieved a significantly higher win-rate, confirming that accounting for temporal dynamics provides superior alignment with user preferences compared to fixed profiles.
- **Ours vs. Random:** The large margin shows that random preferences fail to capture user needs, leading to inconsistent instruction quality.
- **Static vs. Random:** While Static outperforms Random to some extent, it still lags far behind our method due to its inability to adapt to evolving preferences.

The **low tie percentage** across all comparisons indicates that evaluators could generally distinguish between methods, supporting the reliability of the results.

E Evaluation Details

Table 6 organizes the evaluation metrics for the Personalized GUI Agent Task into two key dimensions—**performance** and **efficiency**—each assessed through both immediate and long-term objectives. Immediate metrics capture task-level outcomes, including the **A-process Ratio (APR)**, measuring the proportion of completed unit instructions, and the **P-process Ratio (PPR)**, restricted to flexible preference-sensitive nodes. Efficiency is evaluated via **Completion Time (CT)**, the average execution time per task, and **Cost per Task (CPT)**, the average token-based monetary cost. Long-term metrics (ΔAPR, ΔPPR, ΔCT, ΔCPT) reflect improvements across task sequences, defined as the difference between performance at a later task T_k and the initial task T_1. This framework jointly measures execution quality, personalization, time, and cost, providing a comprehensive view of both short-term performance and long-term adaptability.

E.1 Metric Comparison

Experiment Setup. To evaluate the effectiveness of different metrics, we sampled 25 personalized GUI tasks from PSPA-BENCH and used **M3A** to execute each task and obtain the execution trajectory. For every task, we then measured the performance based on the trajectory using four metrics: **Success Signal**, **Task Completion Ratio (TCR)**, **A-process Ratio (APR)**, and **Human Ratings** (collected from third-party annotators). Success Signal provides a binary success/failure outcome, TCR measures the proportion of actions completed along a predefined reference path, APR reflects the average completion rate across unit instructions along the execution path, and annotators rate the completion ratio (human ratings) of each execution trajectory, which serves as the ground truth.

Results and Analysis. Figure 7 compares the four metrics across tasks (sorted by human ratings). Success Signal is highly coarse-grained, producing only binary outcomes (0 or 1). This representation collapses diverse execution trajectories into two categories, obscuring important differences between agents that achieve partial progress versus those that fail entirely. TCR provides more variation but is still tightly coupled to a fixed path, which leads to penalizing valid alternative behaviors. As a result, TCR diverges notably from human ratings, with weaker correlation ($\rho = 0.47$) and higher error (MSE=0.062).

By contrast, APR demonstrates strong consistency with human judgments. As shown in Figure 7, APR closely follows human ratings across all tasks, with almost all values lying within the human-acceptable tolerance band (± 0.05). Quantitatively, APR achieves a high correlation with human ratings ($\rho = 0.95$) and near-zero error (MSE=0.005). These results show that APR captures fine-grained progress, tolerates diverse execution strategies, and aligns well with human evaluators. Compared to Success Signal and TCR, APR offers a more reliable assessment of agent performance in personalized, long-horizon GUI tasks.

E.2 Metrics Calculation

While the metrics above provide a fine-grained view of agent behavior, computing them reliably is non-trivial. (Xing et al., 2024) measure task completion by comparing executed actions against a fixed ground-truth sequence. Although useful, this method is tied to a single action path and cannot capture an agent's ability to follow alternative strategies or adapt to user preferences. To address this limitation, we leverage the TDG as an intermediate representation and convert it into a struc-

Table 5: Pairwise win-rates of different methods in human evaluation.

Comparison (Pairwise)	Win-rate Ours ↑	Win-rate Baseline	Tie (%)
Ours vs. Static	72.5%	24.1%	3.4%
Ours vs. Random	81.3%	16.2%	2.5%
Static vs. Random	59.6%	38.7%	1.7%

Table 6: Evaluation metrics for the Personalized GUI Agent Task. Short-term metrics measure absolute performance on individual tasks, while long-term metrics capture improvements across instruction sequences. N denotes the number of tasks.

Dimension	Immediate Objective	Long-term Objective
Performance	APR: $\frac{1}{N}\sum \frac{\text{completed unit inst.}}{\text{total unit inst.}}$	ΔAPR: $\text{APR}_{T_k} - \text{APR}_{T_1}$
	PPR: $\frac{1}{N}\sum \frac{\text{completed flexible unit inst.}}{\text{total flexible unit inst.}}$	ΔPPR: $\text{PPR}_{T_k} - \text{PPR}_{T_1}$
Efficiency	CT: $\frac{1}{N}\sum t_i$	ΔCT: $\text{CT}_{T_k} - \text{CT}_{T_1}$
	CPT: $\frac{1}{N}\sum \text{Cost}_i$	ΔCPT: $\text{CPT}_{T_k} - \text{CPT}_{T_1}$

tured checklist. The checklist encodes execution dependencies, alternative paths, and the distinction between fixed and flexible unit instructions. We then employ a LLM such as GPT-5 as an evaluator: given the agent's execution trace and the checklist, the model determines which unit instructions were successfully completed. This graph-driven approach allows APR and PPR to be computed consistently, even when tasks admit multiple valid solutions or require personalization. To ensure the validity and reproducibility of this LLM-based evaluation, we conduct a comprehensive human validation study, as detailed in Section E.3. Details of checklist template are provided in Appendix E.4.

Path Selection Algorithm. As described in Equation 1, selecting the optimal path from the TDG is crucial for accurate metric computation. Algorithm 1 formalizes our two-stage alignment procedure:

The tie-breaking procedure (Lines 17–18) ensures deterministic path selection: we first prefer paths with more matched flexible nodes (prioritizing personalization), then shorter paths (favoring efficiency), and finally paths whose first matched node appears earliest in the trace (reflecting primary intent). This hierarchical criterion guarantees reproducible APR/PPR computation across different evaluation runs.

Algorithm 1 Trace-Graph Alignment with Path Selection

Require: Trace $\mathcal{T} = (a_1, \ldots, a_L)$, TDG $\mathcal{G}$ with valid paths $\mathcal{P}$
Ensure: Optimal path P^*, alignment mapping M
1: **Stage 1: Action-to-Instruction Alignment**
2: **for** each action $a_i \in \mathcal{T}$ **do**
3: $m(a_i) \leftarrow$ LLM_Align$(a_i, \mathcal{U})$ {Map action to unit instruction or $\varnothing$}
4: **end for**
5: **Stage 2: Path Selection**
6: best_score $\leftarrow -1$; candidates $\leftarrow \emptyset$
7: **for** each path $P \in \mathcal{P}$ **do**
8: score$(P) \leftarrow |\{u \in P : \exists a_i, m(a_i) = u\}|$ {Count matched nodes}
9: **if** score$(P) >$ best_score **then**
10: best_score $\leftarrow$ score(P); candidates $\leftarrow \{P\}$
11: **else if** score$(P) =$ best_score **then**
12: candidates $\leftarrow$ candidates $\cup \{P\}$
13: **end if**
14: **end for**
15: **Stage 3: Tie-Breaking** (if |candidates| > 1)
16: $P^* \leftarrow$ TieBreak(candidates) {Apply rules: flexible $\rightarrow$ length $\rightarrow$ earliest}
17: **return** $P^*, \{(a_i, m(a_i)) : m(a_i) \in P^*\}$

E.3 Human Validation of LLM-based Alignment

To validate the reliability of our LLM-based trace–graph alignment, we conduct a comprehensive human agreement study.

Annotation Protocol. We randomly sampled 100 execution traces from our benchmark, covering all four scenarios (Shopping, Dining, Navigation, and Travel) with balanced complexity levels. Three trained annotators independently labeled each trace by marking which unit instructions in the corre-

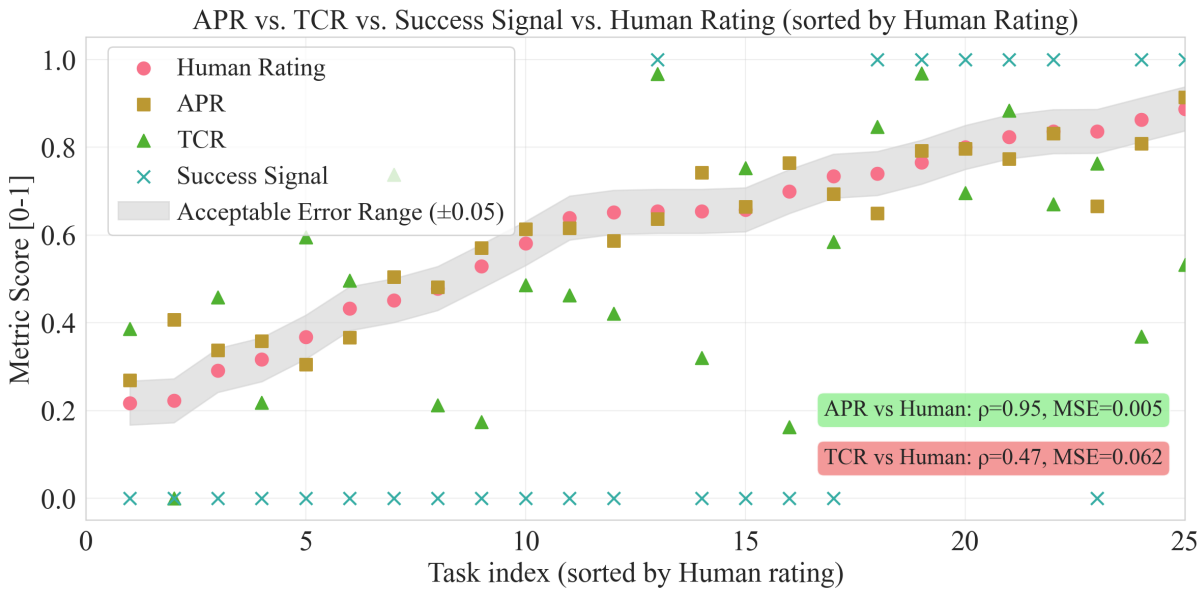


Figure 7: Comparison of evaluation metrics across tasks sorted by human ratings. Each point represents one task. Human ratings (pink dots) are shown with an acceptable tolerance band (±0.05, gray area). A-process Ratio (APR, brown squares) closely follows human ratings with minimal deviation, demonstrating strong alignment (ρ=0.95, MSE=0.005). In contrast, Task Completion Ratio (TCR, green triangles) shows weaker consistency with human ratings (ρ =0.47, MSE=0.062), while Success Signal (blue crosses) provides only binary outcomes, lacking granularity.

sponding TDG were successfully completed. Annotators were provided with the execution trace (screenshots and action logs), the TDG checklist, and detailed annotation guidelines. Prior to the main annotation, annotators completed a calibration phase on 20 pilot traces to ensure consistent interpretation of completion criteria.

Agreement Analysis. We measure inter-annotator agreement using Fleiss' Kappa (κ), which accounts for chance agreement among multiple raters. The overall inter-annotator agreement reached $\kappa = 0.87$, indicating *almost perfect agreement* according to standard interpretation guidelines (Landis and Koch, 1977). For cases with disagreement, we resolved conflicts through majority voting to establish the ground-truth labels.

LLM Alignment Accuracy. We then compared the LLM-based alignment results against the human-annotated ground truth. Table 7 summarizes the results across different instruction types:

The LLM-based alignment achieves an overall accuracy of 93% and Cohen's $\kappa = 0.86$ with human annotations, demonstrating strong agreement. Fixed instructions show slightly higher accuracy (94%) than flexible instructions (91%), which is expected given that flexible instructions involve more nuanced judgments about user preference satisfaction.

Table 7: Validation of LLM-based alignment against human annotations. We report Accuracy, Precision, Recall, F1-score, and Cohen's κ for agreement with human ground truth.

Instruction Type	Acc.	Prec.	Rec.	F1	κ
Fixed Instructions	0.94	0.93	0.96	0.94	0.88
Flexible Instructions	0.91	0.90	0.93	0.91	0.83
Overall	0.93	0.92	0.95	0.93	0.86

Reproducibility Analysis. To assess the reproducibility of the LLM-based alignment, we ran the evaluation pipeline five times on the same set of 100 traces using identical prompts and model settings (temperature = 0). The alignment results showed 98.7% consistency across runs, with a standard deviation of only 0.8% for the computed APR values. This high consistency demonstrates that our LLM-based alignment method is reproducible when using deterministic inference settings.

Error Analysis. We analyzed the 7% of cases where LLM alignment disagreed with human annotations. The primary sources of disagreement were: (i) ambiguous partial completions where an

instruction was partially but not fully satisfied (42% of errors), (ii) alternative valid paths not explicitly enumerated in the checklist (31%), and (iii) edge cases involving UI state changes between screenshots (27%). These insights inform future improvements to both the checklist design and the evaluation prompts.

E.4 Details of Checklists and Prompt Templates

Checklist Structure. We provide four domain-specific checklist templates corresponding to the four scenarios in our benchmark: **Shopping**, **Dining**, **Navigation**, and **Travel**. Each checklist is derived from the Task Decomposition Graph (TDG) and contains the following structured elements for each unit instruction:

- **Checkbox**: A visual marker (□) indicating completion status.
- **Step Name**: The name of the action, annotated with its operation type (*Fixed* or *Flexible*).
- **Description**: A clear explanation of what the step accomplishes.
- **Dependency**: The prerequisite step(s) that must be completed before this step can be executed.
- **Path Options**: For flexible steps with multiple valid execution paths, we enumerate them as Path A, Path B, etc., each with its own sub-steps.

The distinction between *Fixed* and *Flexible* operations is important: Fixed steps are mandatory with no alternatives, while Flexible steps allow multiple valid approaches or are influenced by user preferences. This design lets the checklist accommodate personalized execution strategies while keeping evaluation criteria consistent.

Prompt Templates. We design two prompts to facilitate checklist-based evaluation:

- **Prompt 1** (Checklist Generation): Converts a visual Task Decomposition Graph into a structured textual checklist that can be processed by LLMs. The prompt instructs the model to extract step names, descriptions, dependencies, and operation types while preserving the graph's hierarchical structure.
- **Prompt 2** (LLM-based Evaluation): Given an agent's execution trace (as a sequence of screenshots) and the corresponding checklist, the evaluator LLM determines which unit instructions were successfully completed. The prompt specifies evaluation rules, including handling of alternative paths and dependency constraints.

The complete checklist templates for all four scenarios and the detailed prompt templates are presented in Checklist 1–4 and Prompt 1–2.

F Templates of personalized GUI instruction

We provide examples of templates in personalized scenarios. The templates are categorized by *clarity level* (high, medium, low) and *complexity level* (low, medium, high). In each level, we provide 3 templates with different expressions, which improve the variety of personalized instructions.

G LLM Usage

Large Language Models (LLMs) were used to aid in the writing and polishing of the manuscript. Specifically, we used an LLM to assist in refining the language, improving readability, and ensuring clarity in various sections of the paper. The model helped with tasks such as sentence rephrasing, grammar checking, and enhancing the overall flow of the text.

The authors take full responsibility for the content of the manuscript, including any text generated or polished by the LLM. We have ensured that the LLM-generated text adheres to ethical guidelines and does not contribute to plagiarism or scientific misconduct.

Checklist 1: Shopping sceneario

- ☐ **Open Taobao Fixed**
  Description: Open the Taobao homepage and enter the main platform page
- ☐ **Enter Search Page Fixed**
  Description: Enter the search interface to prepare for product search
  Dependency: Must be executed after "Open Taobao"
- ☐ **Set Conditions (choose one path)**
  Description: Set search constraints (either method is acceptable) *Dependency:* Must first complete "Enter Search Page"
  - **Path A:**
    - ☐ **Set Search Conditions Flexible**
      Description: Enter keywords, categories, etc. for searching
  - **Path B:**
    - ☐ **Set Filter Conditions Flexible**
      Description: Set filters such as price range, sales sorting, etc.
- ☐ **Select Product Flexible**
  Description: Select a product from search results
  Dependency: Must first complete "Set Conditions" in either path
- ☐ **Add to Cart Fixed**
  Description: Click the "Add to Cart" button
  Dependency: Must first complete "Select Product"
- ☐ **Navigate to Cart Page (choose one path) Flexible**
  Description: Enter the shopping cart page (flexible path) *Dependency:* Must first complete "Add To Cart"
  - **Path A:**
    - ☐ **Enter Cart Page (directly) Fixed**
      Description: Directly enter the shopping cart page after adding to cart
  - **Path B:**
    - ☐ **Return to Home Page Fixed**
      Description: First return to the homepage
    - ☐ **Enter Cart Page (from Home) Fixed**
      Description: Enter the shopping cart page from the homepage
      Dependency: Must first complete "Return to Home Page"

Checklist 2: Dining sceneario

- ☐ **Open App Flexible**
  Description: Open the app and enter its main page
- ☐ **Enter Search Page Fixed**
  Description: Enter the search interface to prepare for product search
  Dependency: Must be executed after "Open App"
- ☐ **Set Conditions (choose one path) Flexible**
  Description: Set Searching Conditions (flexible path)
  Dependency: Must be executed after "Enter Search Page"
  - **Path A:**
    - ☐ **Set Search Conditions Flexible**
      Description: Directly search for the restaurant using keywords
  - **Path B:**
    - ☐ **Set Restaurant Type Flexible**
      Description: Search for the type or name of the restaurant
    - ☐ **Set Filter Conditions Flexible**
      Description: Set filter conditions such as price, distance,etc.
      Dependency: Must first complete "Set Restaurant Type"
- ☐ **Select Restaurant Flexible**
  Description: Select a restaurant from the searching results
  Dependency: Must be executed after "Set Searching Conditions"
- ☐ **Enter Main Page of Restaurant Fixed**
  Description: Click to enter the main page of the restaurant
  Dependency: Must be executed after "Select Restaurant"
- ☐ **Check Recommended Dishes Fixed**
  Description: Check the restaurant's recommended dishes
  Dependency: Must be executed after "Enter Main Page of Restaurant"

Checklist 3: Navigation sceneario

- ☐ **Open App Flexible**
  Description: Open the app and enter its main page
- ☐ **Set Route To Destination Flexible**
  *Description:*Set the navigation destination (flexible path)
  Dependency: Must be executed after "Open App"
  - **Path A:**
    - ☐ **Search For Destination Flexible**
      Description: Directly search for the destination in the searching box
  - **Path B:**
    - ☐ **Open Route Planning Interface Fixed**
      Description: Enter the route planning and destination searching interface
    - ☐ **Set Destination Flexible**
      Description: Set destination in the route planning interface
      Dependency: Must first complete "Open Route Planning Interface"
- ☐ **Set Waypoint Flexible**
  Description: Set the waypoint for the route
  Dependency: Must be executed after "Set Route To Destination"
- ☐ **Check Route Information Fixed**
  Description: Check the information of the route
  Dependency: Must be executed after "Set Waypoint"

Checklist 4: Travel sceneario

- ☐ **Open App Flexible**
  Description: Open the App homepage and enter the main platform page
- ☐ **Search For Hotel (choose one path)**
  Description: Set search constraints (either method is acceptable) *Dependency:* Must first complete "Open App"
  - **Path A:**
    - ☐ **Enter Hotel Search Page Fixed**
      Description: Enter the search category for hotel
    - ☐ **Set City And Date Flexible**
      Description: Set the searching conditions like city,type, and time *Dependency:* Must first complete "Enter Hotel Search Page"
  - **Path B:**
    - ☐ **Enter Search Page Fixed**
      *Description:*Enter the search page of the APP
    - ☐ **Search For Hotel In Target City Flexible**
      Description: Set searching conditions like city and hotel type *Dependency:* Must first complete "Enter Search Page"
    - ☐ **Set Date Fixed**
      Description: Set dates of the stay *Dependency:* Must first complete "Set Date"
- ☐ **Select Hotel Flexible**
  Description: Select a hotel from the results *Dependency:* Must first complete "Search For Hotel"
- ☐ **Check Reviews Fixed**
  Description: Check reviews of the hotel on the APP *Dependency:* Must first complete "Select Hotel"

Prompt 1: Prompt for turning task decomposition graphs into checklist:

Your task is to turn a task decomposition graph in a picture into textual checklist that can be understood by another large language model easily. The task decomposition graph is provided below:
{TASK_GRAPH}
The checklist you generate should include:

-A checkbox
-Step Name: The name of the action from the graph.
-Description :A clear explanation of what the step does.
-Dependency :The prerequisite step(s) that must be completed before this step.
-Operation Type: Mark as Fixed if the step is mandatory with no alternatives, or Flexible if there are multiple paths/options, or the step is affected by user preference. (including grouped parallel steps).
-Path Options (if applicable): If the step has multiple paths (e.g., parallel steps leading to the same next step), list them as Path A, Path B, etc., with a description with the same format as the steps you generate.

A example of the checklist you generate is provided below, make sure you follow the correct format.
Example:

- ☐ **Set Route To Destination Flexible**
  *Description:*Set the navigation destination (flexible path)
  Dependency: Must be executed after "Open App"
  - **Path A:**
    - ☐ **Search For Destination Flexible**
      Description: Directly search for the destination in the searching box
  - **Path B:**
    - ☐ **Open Route Planning Interface Fixed**
      Description: Enter the route planning and destination searching interface
    - ☐ **Set Destination Flexible**
      Description: Set destination in the route planning interface
      Dependency: Must first complete "Open Route Planning Interface"

Prompt 2: Prompt for LLM-based evaluation:

You are an expert in evaluating the performance of a GUI Agent. Based on the input checklist and the agent's execution flow images, you will determine one by one whether the agent has completed all sub-tasks in the checklist. The checklist will be provided to you in JSON format, while the images will be base64 encoded. Your output should also be a JSON, where you set the complete marker for each sub-task in the input JSON to 1 if the sub-task is completed and 0 if not completed.

{IMAGES}
{CHECKLIST}

Rules: 1. Check the subtasks and determine whether they are completed one by one. 2. For subtask with various paths, the completion of any path marks the completion of the subtask. 3. If the dependency of a subtask is not completed, the subtask itself is not completed as well.

Table 8: Instruction templates of shopping scenario.

Clarity level	Complexity level	Template Examples (English)
High clarity	Low	1. I want to search for {brand} {category} on {APP}, with prices between {price range} yuan. 2. Help me find a {brand} {category} on {APP}, with prices in the range of {price range} yuan. 3. Check {APP} for a {brand} {category}, keeping the price within {price range} yuan.
	Medium	1. I want to buy a {brand} {category} on {APP}, with prices between {price range} yuan, and stop at the product details page. 2. Help me find a {brand} {category} on {APP}, with prices in the range of {price range} yuan, and stay on the product details page. 3. Check {APP} for a {brand} {category}, with prices around {price range} yuan, and stay on the product details page.
	High	1. I want to buy a {brand} {category} on {APP}, with prices between {price range} yuan, add the item to the cart, and finally stay on the cart page. 2. Help me find a {brand} {category} on {APP}, with prices around {price range} yuan, add it to the cart, and finally stay on the cart page. 3. Check {APP} for a {brand} {category}, within {price range} yuan, add it to the cart, and finally stay on the cart page.
Medium clarity	Low	1. Search {brand} {category} on {APP}, price about {price range}. 2. Help me find {brand} {category} in {APP}, budget {price range}. 3. Check {APP} for {brand} {category}, price around {price range}.
	Medium	1. Want to buy {brand} {category} on {APP}, price {price range}, just open the details page. 2. {APP}, find {brand} {category}, around {price range}, go to the details page. 3. Help me search {brand} {category} on {APP}, price {price range}, open the details page.
	High	1. Buy {brand} {category} on {APP}, budget {price range}, add to cart and stay there. 2. {APP}, find {brand} {category}, price about {price range}, add to cart. 3. Help me add {brand} {category} ({price range}) in {APP} to the cart, and stay on the cart page.
Low clarity	Low	1. {APP} {brand}{category} {price range} 2. Find {brand}{category} {price range} @{APP} 3. {brand}{category} {price range} /{APP}
	Medium	1. {APP} {brand}{category} {price range} details 2. {brand}{category} {price range} → details @{APP} 3. {APP}: {brand}{category} {price range} details page
	High	1. {APP} {brand}{category} {price range} cart 2. Add to cart {brand}{category} {price range} @{APP} 3. {brand}{category} {price range} → cart /{APP}

Table 9: Instruction templates for music scenario.

Clarity level	Complexity level	Template Examples (English)
High clarity	Low	1. I want to search for {singer} on {APP} and go to the singer's homepage. 2. Help me search for {singer} on {APP} and enter the singer's homepage. 3. Check {APP} for {singer} and click to open the singer's homepage.
	Medium	1. I want to search for {singer} on {APP}, go to the singer's homepage, and view the albums. 2. Help me find {singer} on {APP}, enter the singer's homepage, and check the albums. 3. On {APP}, search for {singer}, open the singer's homepage, then view the albums.
	High	1. I want to search for {singer} on {APP}, open the singer's homepage, check the albums, and select a song to play. 2. Help me find {singer} on {APP}, enter the singer's homepage, view the albums, and then play a song. 3. On {APP}, search for {singer}, open the homepage, view the albums, and finally play a song.
Medium clarity	Low	1. Search {singer} on {APP}, go to homepage. 2. Help me find {singer} in {APP}, check homepage. 3. {APP} search {singer}, open homepage.
	Medium	1. Want to find {singer} on {APP}, check homepage and albums. 2. {APP} search {singer}, enter homepage, view albums. 3. Help me search for {singer} on {APP}, view albums in homepage.
	High	1. Find {singer} on {APP}, view albums, select a song to play. 2. {APP} search {singer}, choose a song from album to listen. 3. Help me find {singer} on {APP}, select a song from album and play.
Low clarity	Low	1. {APP} {singer} homepage 2. Find {singer} @{APP} 3. {singer} homepage /{APP}
	Medium	1. {APP} {singer} album 2. {singer} album @{APP} 3. {APP}: {singer} album page
	High	1. {APP} {singer} play album song 2. Play {singer} album song @{APP} 3. {singer} $\rightarrow$ play album song /{APP}

Table 10: Instruction templates for dining scenario.

Clarity level	Complexity level	Template Examples (English)
High clarity	Low	1. I want to search for {cuisine type} restaurants on {APP}, with an average spending between {price range} yuan. 2. Help me find a {cuisine type} restaurant on {APP}, with average spending in the range of {price range} yuan. 3. Check {APP} for {cuisine type} restaurants, priced between {price range} yuan.
	Medium	1. I want to search for {cuisine type} restaurants on {APP}, with an average spending between {price range} yuan, and stay on the restaurant details page. 2. Help me find a {cuisine type} restaurant on {APP}, with average spending in the range of {price range} yuan, and remain on the details page. 3. Check {APP} for {cuisine type} restaurants, priced between {price range} yuan, and stay on the details page.
	High	1. I want to search for {cuisine type} restaurants on {APP}, with an average spending between {price range} yuan, enter the details page, and view all recommended dishes. 2. Help me find a {cuisine type} restaurant on {APP}, with average spending in the range of {price range} yuan, go to the details page, and view all recommended dishes. 3. Check {APP} for {cuisine type} restaurants, priced between {price range} yuan, enter the details page, and view all recommended dishes.
Medium clarity	Low	1. Search {cuisine type} on {APP}, around {price range}. 2. Help me find {cuisine type} in {APP}, budget {price range}. 3. Check {APP} for {cuisine type}, price about {price range}.
	Medium	1. Want to find {cuisine type} on {APP}, price between {price range}, open details page. 2. {APP} search {cuisine type}, in range {price range}, open details. 3. Help me search {cuisine type} on {APP}, price {price range}, view details.
	High	1. Find {cuisine type} on {APP}, budget {price range}, check recommended dishes. 2. {APP} search {cuisine type}, price {price range}, view recommended dishes. 3. Help me find {cuisine type} ({price range}) on {APP}, view recommended dishes.
Low clarity	Low	1. {APP} {cuisine type} {price range} 2. Find {cuisine type} {price range} @{APP} 3. {cuisine type} {price range} /{APP}
	Medium	1. {APP} {cuisine type} {price range} details 2. {cuisine type} {price range} $\rightarrow$ details @{APP} 3. {APP}: {cuisine type} {price range} details page
	High	1. {APP} {cuisine type} {price range} view recommended dishes 2. Recommended dishes {cuisine type} {price range} @{APP} 3. {cuisine type} {price range} $\rightarrow$ view recommended dishes /{APP}