Uncertainty-Aware GUI Agent: Adaptive Perception through Component Recommendation and Human-in-the-Loop Refinement

Anonymous ACL submission

Abstract

Graphical user interface (GUI) agents have shown promise in automating mobile tasks but still struggle with input redundancy and decision ambiguity. In this paper, we present **RecAgent**, an uncertainty-aware agent that addresses these issues through adaptive perception. We distinguish two types of uncertainty in GUI navigation: (1) perceptual uncertainty, caused by input redundancy and noise from comprehensive screen information, and (2) decision uncertainty, arising from ambiguous tasks and complex reasoning. To reduce perceptual uncertainty, RecAgent employs a component recommendation mechanism that identifies and focuses on the most relevant UI elements. For decision uncertainty, it uses an interactive module to request user feedback in ambiguous situations, enabling intent-aware decisions. These components are integrated into a unified framework that proactively reduces input complexity and reacts to high-uncertainty cases via human-in-the-loop refinement. Additionally, we propose a dataset called **ComplexAction** to evaluate the success rate of GUI agents in executing specified single-step actions within complex scenarios. Extensive experiments validate the effectiveness of our approach. The code and dataset will be open-sourced.

1 Introduction

Graphical user interface (GUI) agents aim to automate human interactions with mobile applications, enabling users to accomplish tasks such as ordering food, checking the weather, or booking tickets by simply specifying high-level goals (Zheng et al., 2024; Rawles et al., 2024). While recent progress in large language models (LLMs) (Achiam et al., 2023; Team et al., 2024; Bai et al., 2023) and vision-language systems has empowered GUI agents with improved planning and action capabilities (Wang et al., 2025a; Nguyen et al., 2024; Li et al., 2025), several core challenges remain unresolved. In particular, two critical issues hinder their reliability and generalizability in complex real-world applications: **input redundancy** and **decision ambiguity**, as shown in Figure 1.

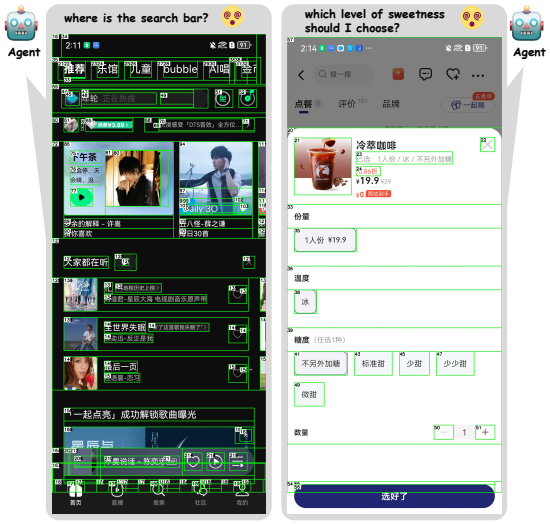

Figure 1: The two major challenges faced by existing GUI Agents: (a) Perceptual uncertainty caused by input redundancy. For example, when searching in a music application, excessive input redundancy prevents the Agent from locating the search box component. (b) Decision uncertainty caused by the lack of interactive mechanisms. For example, the Agent does not know which sweetness level to select for the user when helping with coffee ordering. The input used here is in SoM (Yang et al., 2023) format.

Many existing GUI agents take a comprehensive approach to perception, incorporating both full-screen screenshots and complete UI element lists as input (Rawles et al., 2024; Xie et al., 2025a; Wang et al., 2024a). While this exhaustive strategy ensures access to all available information, it also introduces significant noise and redundancy. This input redundancy has two clear drawbacks: on one hand, it leads to low computational efficiency; on the other hand, it interferes with the model's ability to perceive truly useful information (Zhou et al.,

2024; Wu et al., 2025; Zhang et al., 2024; Niu et al., 2025; Xu et al., 2025a). For instance, dozens or even hundreds of UI elements may be irrelevant to the current subtask, overwhelming the agent and complicating both perception and reasoning. We define this problem as *perceptual uncertainty*, an uncertainty arising from the agent's inability to focus on truly relevant components due to overloaded input. As shown in Figure 1(a), due to excessive input redundancy, the Agent cannot locate the desired search box to complete the search action.

Another major challenge is the agent's inability to handle *decision uncertainty*, especially in ambiguous scenarios that involve user preferences or require disambiguation. For example, as shown in Figure 1(b), when ordering coffee, users often provide only rough instructions such as "help me order a coffee". However, the actual operation typically requires selecting the user's desired sweetness or temperature. Most existing systems adopt a purely autonomous approach and lack mechanisms to interactively solicit user feedback during execution (Rawles et al., 2024; Zhang et al., 2023), leading to unsatisfactory or failed actions.

To address these two challenges, we propose **RecAgent**, an uncertainty-aware GUI agent that enhances both perception and decision-making through adaptive mechanisms. Specifically, RecAgent is composed of several functional agents: *Planning Agent*, *Decision Agent*, *Reflection Agent*, and *Interaction Agent*, alongside two auxiliary modules: a *Component Recommendation Module* and a *Memory Unit*. The overall process starts with the Planning Agent, which generates the current subgoal based on the user task and the observed environment state. To combat perceptual uncertainty, we introduce a recommendation-based perception mechanism: the Component Recommendation Module selectively filters and ranks relevant UI elements from the environment, using keyword matching, semantic similarity, and historical context. Instead of providing the Decision Agent with an entire UI tree, only the top-ranked elements (e.g., top 10) are passed forward, substantially reducing input size while preserving essential information.

Next, the Decision Agent takes the current subgoal and the filtered UI components to predict the optimal action. This action is executed in the environment, resulting in a new state. The Reflection Agent evaluates whether the current subgoal has been successfully completed. If it fails, the agent rolls back, excludes the previous UI element choice, and attempts alternative actions until success.

In cases where the system encounters high decision uncertainty (e.g., multiple valid options or missing preferences), the Interaction Agent determines whether user input is required. If necessary, it dynamically generates a query to the user (e.g., "What level of sweetness do you prefer? "), and integrates the feedback into the ongoing execution. All intermediate observations and decisions are recorded and updated in the Memory Module for continual learning and future planning.

Furthermore, we introduce a new evaluation dataset, **ComplexAction**, designed to assess the agent's capability to execute fine-grained single-step actions within visually and semantically complex environments. Unlike previous benchmarks that focus solely on end-to-end task completion (Xu et al., 2025b; Chen et al., 2025b; Zhou et al., 2025), ComplexAction focuses exclusively on the success rate of executing a specified single-step action within complex scenarios, thereby providing a better validation of the effectiveness of our component recommendation module.

In summary, our contributions are threefold:

- We identify and tackle two forms of uncertainty in GUI agents: perceptual and decision uncertainty, that hinder performance in real-world applications.
- We propose RecAgent, a novel uncertainty-aware GUI agent featuring a recommendation-based perception mechanism and a human-in-the-loop interaction module.
- We construct ComplexAction, a new benchmark dataset for evaluating single-step GUI action accuracy in complex GUI scenarios.

Extensive experiments demonstrate that RecAgent outperforms existing baselines in both overall success rate and step-wise accuracy, especially in complex environments with high uncertainty.

2 Related Work

GUI Agents. Recent advancements in GUI agents have largely leveraged the powerful understanding capabilities of Multimodal Large Language Models (MLLMs) (Achiam et al., 2023; Team et al., 2024; Bai et al., 2025). Representative works include the AppAgent series (Zhang et al., 2023; Li et al., 2024; Jiang et al., 2025), the Mobile-Agent series (Wang et al., 2024b,a, 2025b), and so on.

Early systems like AppAgent (Zhang et al., 2023) and AutoDroid (Wen et al., 2024) demonstrated task automation using foundation models. The integration of visual perception with LLMs has been explored in systems such as SeeAct (Zheng et al., 2024), which utilizes GPT-4V for web task automation, and MobileAgentV2 (Wang et al., 2024a), which employs a multi-agent architecture with memory for mobile tasks. M3A (Rawles et al., 2024) combines ReAct-style reasoning with Set-of-Mark (Yang et al., 2023) visual annotations for Android control, showcasing strong generalization.

Addressing Input Redundancy: The problem of input redundancy, which we identify as a source of *perceptual uncertainty*, has garnered attention. Some work focuses on improving efficiency through better action grounding or hierarchical perception (Zhou et al., 2024; Chen et al., 2025a). Techniques like the Set-of-Mark (SoM) representation (Yang et al., 2023) aim to simplify the input space by annotating key elements. Our approach differs by introducing an adaptive, recommendation-based mechanism to actively filter and prioritize relevant UI components, directly mitigating the negative impact of overloaded inputs on perception and reasoning.

Handling Decision Ambiguity: Decision ambiguity, or *decision uncertainty*, particularly in user preference elicitation, is another recognized challenge. Interactive frameworks have been proposed where agents ask questions to resolve ambiguities, such as those for mobile navigation (Liu et al., 2024) or general communication (Seed., 2025). However, integrating such interactive capabilities seamlessly into the GUI agent's execution loop, especially for mobile tasks, remains less explored. Our work proposes a dedicated interaction module within RecAgent that dynamically identifies high-uncertainty states and solicits user feedback, enabling intent-aware decision-making in ambiguous scenarios like customization choices.

3 Method

In this section, we present the architecture and workflow of **RecAgent**, an uncertainty-aware GUI agent designed to reduce perceptual and decision uncertainty during task execution. The overall agent consists of four major components: *Planning Agent*, *Decision Agent*, *Reflection Agent* and *Interaction Agent*, along with two key modules: the *Component Recommendation Module* and a shared *Memory Unit* maintaining intermediate and historical information. The framework is illustrated in Figure 2.

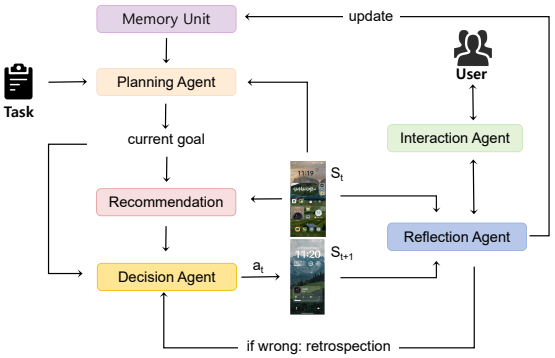

Figure 2: Schematic overview of the RecAgent.

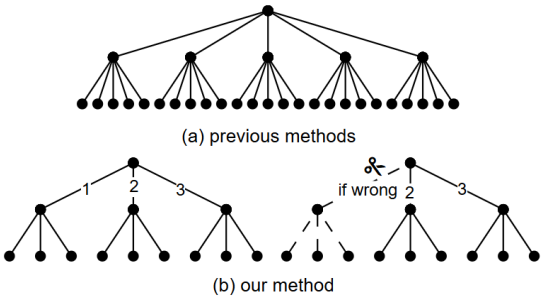

Figure 3: Comparison with previous methods. Schematic diagram of using the component recommendation module in conjunction with the retrospection mechanism. The recommendation module reduces the complexity of the input and narrows down the path choices. When it is determined that a previous action was ineffective, the retrospection mechanism deletes the previously chosen path and reselects a possible path (i.e., action).

3.1 Task Formalization

Given a high-level task instruction T, RecAgent interacts with the mobile device environment $\mathcal{E}$ to generate a sequence of actions $\{a_1, a_2, \ldots, a_n\}$ that completes the task. At each time step t, the agent observes the environment state s_t, makes an adaptive decision, and executes an action a_t to reach the next state s_{t+1}. Through continuous interactions, we obtain a trajectory $\mathcal{J} = (s_1, a_1), (s_2, a_2), (s_3, a_3), ..., (s_L, a_L)$, where L represents the length of the trajectory. If the final state s_L reaches the goal state, the task execution is considered successful.

3.2 Task Decomposition via Planning Agent

The Planning Agent is responsible for decomposing the overall task T into a sequence of intermediate subgoals. At each step t, it generates the current subgoal g_t based on the task T, current environ-

ment state s_t, and historical memory M_{t-1}:

$$g_t = \text{Planner}(T, s_t, M_{t-1}). \quad (1)$$

Here, g_t represents the immediate semantic intent (e.g., "click the search bar", "choose delivery method") and serves as the guiding signal for perception and decision-making. This task decomposition enables the agent to gradually transform complex high-level tasks into executable concrete actions, thereby reducing the complexity and uncertainty in task execution, while also aligning with humans' step-by-step operational habits.

3.3 Component Recommendation

The GUI environment state s_t consists of a list of UI elements $\mathcal{U}_t = \{u_t^{(1)}, u_t^{(2)}, \ldots, u_t^{(N)}\}$ and current screenshot c_s (optional), where each $u_t^{(i)}$ is a structured representation containing text, bounds, type, etc. Some complex interfaces can have as many as hundreds of UI elements, processing all elements leads to redundant and noisy inputs (Xie et al., 2025b; Chen et al., 2025a).

Inspired by the "multi-channel recall" mechanism in recommendation systems (Isinkaye et al., 2015; Bobadilla et al., 2013), we treat the current subgoal g_t as a query and the UI elements in $\mathcal{U}_t$ as candidate items. To efficiently identify components most relevant to the current task intent, we design a component recommendation module (CRM) that employs multiple independent recommendation pathways. Each pathway generates a candidate subset in parallel based on distinct matching logic, and the final perception input is formed by taking the union of all outputs: ensuring high coverage while significantly reducing input scale.

Specifically, we define the following recommendation pathways:

Keyword Matching Pathway: Identifies keywords in g_t (e.g., "search", "submit", "open") and matches them exactly or fuzzily against the text of UI elements;

Semantic Matching Pathway: Leverages a pretrained language model (e.g., BERT (Devlin et al., 2018)) to assess the semantic relevance between g_t and the text of each UI element, retrieving controls with implicit functional alignment;

LLM-based Intent Recommendation Pathway: Uses a LLM to perform contextual understanding of both g_t and each $u_t^{(i)}$, determining whether their functional roles align, and outputs high-confidence recommendations.

Let $\mathcal{R}_k(g_t, \mathcal{U}_t)$ denote the candidate set returned by the k-th recommendation pathway. The final perception input set $\mathcal{U}_t'$ is defined as the union of all pathway outputs:

$$\mathcal{U}_t' = \bigcup_k \mathcal{R}_k(g_t, \mathcal{U}_t). \quad (2)$$

This set $\mathcal{U}_t'$ includes any UI element deemed relevant by at least one pathway, thereby mitigating the risk of missed detections due to reliance on a single strategy, while effectively compressing the input space. Compared to processing all N elements, this approach significantly reduces computational overhead and noise in the perception module, enhancing the system's robustness and response efficiency in complex GUI environments.

3.4 Decision Making via Decision Agent

Given the current subgoal g_t generated by the Planning Agent and the filtered UI element list $\mathcal{U}_t'$ produced by the Component Recommendation Module, the Decision Agent selects an executable action a_t and generates a natural language description d_t of the intended behavior:

$$a_t, d_t = \text{Decision}(g_t, \mathcal{U}_t'). \quad (3)$$

The action space follows the M3A Agent (Rawles et al., 2024) framework and includes a comprehensive set of GUI operations such as `click`, `double-click`, `text input`, and so on. d_t provides a human-readable explanation (e.g., "Click on the search bar to enter the query").

By operating on the compact and semantically relevant subset $\mathcal{U}_t'$, the Decision Agent reduces the search space for candidate targets, thereby mitigating decision uncertainty and improving accuracy and efficiency. The inclusion of d_t enables better interpretability and facilitates downstream components such as reflection and memory updating.

For actions that require interaction, each action is structured as a tuple $(\text{action\_type}, \text{target\_element})$, where $\text{target\_element} \in \mathcal{U}_t'$ is the UI component selected for interaction. For predefined navigation actions such as `scroll up` or `navigate back`, the Operator directly invokes the corresponding system API without requiring precise element localization.

3.5 Retrospection via Reflection Agent

After executing action a_t and transitioning to the next state s_{t+1}, the Reflection Agent evaluates the

outcome and produces both a success indicator and a contextual summary:

$$(\text{Success}_t, \text{Summary}_t) = \text{Reflect}(g_t, s_t, s_{t+1}). \tag{4}$$

Here, $\text{Success}_t \in \{\text{True}, \text{False}\}$ indicates whether the action a_t successfully advanced progress toward subgoal g_t, while Summary_t is a natural language or structured description summarizing the observed changes, such as "The search bar was clicked, but no keyboard appeared" or "The page scrolled down, revealing more product items."

We introduce a retrospection mechanism, illustrated in Figure 3. If Success_t is False, indicating that the action failed to achieve the intended effect, the agent reverts to the previous state s_t, removes the previously selected action from the filtered candidate set $\mathcal{U}'_t$ and reinvokes the Decision Agent to select an alternative action. This backtracking process enables the agent to dynamically revise its decisions without restarting the entire task.

The tight integration between the retrospection mechanism and the Component Recommendation Module plays a critical role in enhancing system robustness. As shown in Figure 3, this coordination reduces the effective state space by eliminating invalid candidates from future consideration, thereby avoiding repeated failures on the same element. Furthermore, it mitigates the impact of perceptual inaccuracies (e.g., mislocalized elements) or reasoning errors (e.g., incorrect action selection), allowing the agent to recover gracefully and explore alternative interaction paths. This closed-loop feedback design significantly improves the agent's adaptability and reliability in complex and dynamic GUI environments.

3.6 User Feedback via Interaction Agent

To proactively manage *decision uncertainty*, the Interaction Agent evaluates the need for user feedback after each successful action execution. Given the current subgoal g_t, the environment state s_{t+1} (resulting from action a_t), and the action description d_t, the Interaction Agent determines whether user input is required:

$$\text{need\_feedback}_t, q_t = \text{Interact}(g_t, a_t, s_{t+1}, d_t). \tag{5}$$

Here, $\text{need\_feedback}_t \in \{\texttt{True}, \texttt{False}\}$ is a boolean flag indicating whether feedback is necessary. If need_feedback_t is `True`, q_t is a natural language query string (e.g., "what level of sweetness do you prefer?") posed to the user. If need_feedback_t is `False`, q_t is set to `None`.

If user feedback is required (need_feedback_t is `True`), the agent awaits the user's response u_t. This response is then incorporated to refine the next subgoal:

$$g'_{t+1} = \text{UpdateGoal}(g_{t+1}, u_t). \tag{6}$$

The agent then proceeds with subsequent steps using the (potentially updated) subgoal g_{t+1}. This allows RecAgent to resolve ambiguities and align its actions with user intent in real-time.

3.7 Memory Unit and Task Termination

Throughout the interaction process, all subgoals, actions, description, success indicators, summary, query and user feedback (if any) are stored in the Memory Unit:

$$M_t = M_{t-1} \cup \{(g_t, a_t, d_t, \text{Success}_t, \text{Summary}_t, q_t, u_t)\}. \tag{7}$$

The task terminates when the Decision Agent outputs a special `COMPLETE` action indicating that the task has been completed, or the maximum number of allowed steps L_{max} is reached.

3.8 Overall Algorithm

The full execution loop is summarized in Algorithm 1, combining planning, recommendation, decision, reflection, and interaction in an uncertainty-aware loop.

4 The ComplexAction Dataset

To effectively evaluate the capability of GUI agents in handling *perceptual uncertainty* within complex environments and to specifically validate the effectiveness of our Component Recommendation Module, we introduce the ComplexAction dataset.

Motivation and Design Principles. Unlike existing benchmarks that primarily focus on end-to-end task completion rates, the ComplexAction Dataset is designed to assess an agent's accuracy in executing *fine-grained, single-step actions* (e.g., clicking a specific button) within visually and semantically complex GUI scenes. This focus allows for a more precise measurement of an agent's core perception and decision-making abilities, particularly its ability to locate relevant UI elements amidst significant input redundancy. Our dataset directly targets the evaluation of mechanisms designed to mitigate perceptual uncertainty.

Algorithm 1 RecAgent Execution Loop

Input: Task T, initial state s_0
Initialize memory $M_0 = \emptyset$
$s \leftarrow s_0$
for $t = 1$ to $L_{\max}$ **do**
  $g_t \leftarrow \text{Planner}(T, s, M_{t-1})$
  // Component Recommendation
  $\mathcal{U}'_t \leftarrow \bigcup_k \mathcal{R}_k(g_t, \mathcal{U}_t)$
  // Decision Making
  $(a_t, d_t) \leftarrow \text{Decision}(g_t, \mathcal{U}'_t)$
  Execute a_t, observe new state s'
  // Reflection
  $(\text{Success}_t, \text{Summary}_t) \leftarrow \text{Reflect}(g_t, s, s')$
  if Success_t == `False` **then**
    Remove selected element from $\mathcal{U}'_t$
    Reinvoke Decision Agent with updated $\mathcal{U}'_t$
    Continue
  end if
  // Interaction
  $(\text{need\_feedback}_t, q_t) \leftarrow \text{Interact}(g_t, s', d_t)$
  if need_feedback_t == `True` **then**
    Receive user input u_t
    $g_{t+1} \leftarrow \text{UpdateGoal}(g_{t+1}, u_t)$
  else
    $u_t \leftarrow$ None
  end if
  // Memory Update
  $M_t \leftarrow M_{t-1} \cup \{(g_t, a_t, d_t, \text{Success}_t, \text{Summary}_t, q_t, u_t)\}$
  $s \leftarrow s'$
  if a_t == `[COMPLETE]` **then**
    break
  end if
end for

Agent	Input	Base Model	Task Success Rate (%)
Human (Rawles et al., 2024)	*screen*	-	*80.0*
Aguvis (Xu et al., 2025b)	screen	GPT-4o	37.1
AppAgent (Zhang et al., 2023)	SoM	GPT-4o	14.9
Aria-UI (Yang et al., 2025)	screen	GPT-4o	44.8
AutoDroid (Wen et al., 2024)	a11y tree	GPT-4o	15.7
T3A (Rawles et al., 2024)	a11y tree	GPT-4o	37.6
M3A (Rawles et al., 2024)	SoM	GPT-4o	40.5
Ponder & Press (Wang et al., 2024c)	screen	GPT-4o	34.5
SeeAct (Zheng et al., 2024)	SoM	GPT-4-turbo	15.5
UGround (Gou et al., 2025)	screen	GPT-4o	32.8
Mirage-O (Xie et al., 2025b)	screen	GPT-4o	42.2
GUI-explorer (Xie et al., 2025a)	SoM	GPT-4o	47.4
RecAgen (Ours)	a11y tree	GPT-4o	43.5
RecAgen (Ours)	SoM	GPT-4o	**47.8**

Table 1: Performance comparison on AndroidWorld.

Agent	Input	Base Model	Task Success Rate (%)
Human (Rawles et al., 2024)	*screen*	-	*100*
SeeAct (Zheng et al., 2024)	SoM	GPT-4 Turbo	66.1
AppAgent (Zhang et al., 2023)	SoM	GPT-4o	56.1
T3A (Rawles et al., 2024)	a11y tree	GPT-4o	68.1
M3A (Rawles et al., 2024)	SoM	GPT-4o	68.5
OS-Atlas* (Wu et al., 2024)	screen	GPT-4o	51.1
UGround* (Gou et al., 2025)	screen	GPT-4o	48.4
Mirage-O (Xie et al., 2025b)	screen	GPT-4o	60.9
RecAgen (Ours)	a11y tree	GPT-4o	68.8
RecAgen (Ours)	SoM	GPT-4o	**69.8**

Table 2: Performance comparison on MobileMiniWoB++. * means the results are reproduced under the same prompt setting.

Data Collection and Structure. We identify five common and representative action types: *Click Search Box*, *Create New Content*, *Like/Upvote*, *Refresh Interface*, and *Sort Items*. For each action type, we collect scenarios from popular mobile applications in China (e.g., Pinduoduo, Tencent Video, Xiaohongshu, etc.), resulting in a total of 62 diverse and complex scenes. A scene is considered "complex" if its GUI state contains hundreds of UI elements, presenting a significant challenge for agents that rely on full-state inputs. Figure 1(a) provides an example of such a scenario.

For each collected scene, we provide the raw screen screenshot and the parsed list of UI elements as input. The ground truth is defined by the specific target UI element for the designated action. Evaluation can be performed by checking if the agent's predicted action targets the correct element or by verifying if the subsequent UI state transition aligns with the expected outcome on a real device, as judged by human annotators.

This dataset facilitates a targeted evaluation of an agent's ability to focus on relevant components, thereby providing a more granular assessment of its robustness in challenging perceptual conditions. More details can be found in the Appendix B.

5 Experiments

5.1 Experimental Setting

Implementation Details. We build RecAgent based on the M3A Agent (Rawles et al., 2024), adopting the same action space. For the newly introduced modules, we design the prompts in a style consistent with M3A. Additionally, similar to M3A, we also implement two input modalities: accessibility tree (a11y tree) and SoM (a11y tree + screenshot) (Yang et al., 2023). Following most work in the field (Xie et al., 2025a,b), we use the widely adopted GPT-4o as the base model. In the semantic matching pathway, we use the text-embedding-3-small model to compute text similarity. The maximum number of steps per task is set to 30.

Datasets. We evaluate the proposed RecAgent against several related works on the following datasets: **AndroidWorld**: AndroidWorld is an Android environment that includes 116 tasks drawn from 20 real-world applications. **MobileMiniWoB++**: MobileMiniWoB++ (Rawles et al., 2024) adapts the MiniWoB++ (Shi et al., 2017; Liu et al., 2018) benchmark to the Android environment within the AndroidWorld framework and supports 92 tasks after compatibility filtering. **Com-**

Agent	Input	Base Model	Action Success Rate (%)
Human	*screen*	-	*100*
SeeAct (Zheng et al., 2024)	SoM	GPT-4 Turbo	61.2
AppAgent (Zhang et al., 2023)	SoM	GPT-4o	53.2
T3A (Rawles et al., 2024)	a11y tree	GPT-4o	59.7
M3A (Rawles et al., 2024)	SoM	GPT-4o	64.5
OS-Atlas (Wu et al., 2024)	screen	GPT-4o	58.0
UGround (Gou et al., 2025)	screen	GPT-4o	54.8
Mirage-O (Xie et al., 2025b)	screen	GPT-4o	58.0
RecAgen (Ours)	a11y tree	GPT-4o	64.5
RecAgen (Ours)	SoM	GPT-4o	**69.3**

Table 3: Performance comparison on ComplexAction.

plexAction: The dataset proposed in this paper for evaluating the success rate of given single-step actions in complex scenarios is also built upon the AndroidWorld framework.

Comparative Baselines. We select several SOTA methods for performance comparison. Relevant methods compared include SeeAct (Zheng et al., 2024), M3A/T3A (Rawles et al., 2024), MobileAgentV2 (Wang et al., 2024a) etc., among which M3A is the most closely related to our approach. Additionally, we evaluate approaches that solely use screenshots as input, such as Mirage (Xie et al., 2025b) and CogAgent (Hong et al., 2024).

5.2 Comparision With SOTA Methods

As shown in Table 1, RecAgent demonstrates outstanding performance on the AndroidWorld benchmark, with its SoM-based input variant achieving the best results among all compared methods. Compared to the most relevant baseline methods M3A and T3A, it achieves performance improvements of 17.6% and 15.7% under two input conditions respectively, which fully validates the effectiveness of our proposed approach.

Notably, while both GUI-Explorer and Mirage also deliver strong performance, their methods require either pre-exploration or iterative knowledge accumulation in relevant environments due to their exploration and knowledge extraction capabilities. In contrast, our RecAgent can be deployed directly and exhibits superior generalization ability.

Similarly, the proposed RecAgent's SoM-based input variant achieves optimal performance on the MobileMiniWoB++ dataset, as shown in Table 2. While all methods demonstrate decent performance on this relatively simpler dataset (with human achieving 100% accuracy), our RecAgent still attains the best performance at 69.8%, further validating RecAgent's generalization capability. Since the scenarios in this dataset are relatively simple, our method shows only marginal improvements over M3A and T3A.

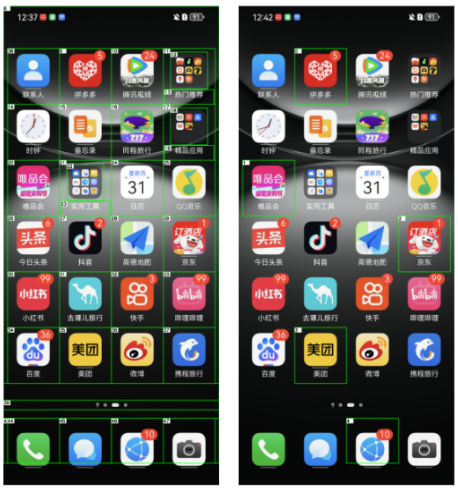

Figure 4: Visualization of SoM outputs before and after using the Component Recommendation Module. Left: original; right: with module. Current goal: "open a shopping app".

As shown in Table 3, RecAgent demonstrates strong performance on the ComplexAction dataset. Specifically, the SoM-based input variant achieves a success rate of 69.3%, outperforming all other methods except human-level performance (100%). This result highlights the effectiveness of our approach in handling complex scenarios.

Compared to the most relevant baseline methods (M3A and T3A), RecAgent shows significant improvements in action success rates. While M3A achieves 64.5% and T3A achieves 59.7%, RecAgent's SoM-based variant surpasses both by a notable margin. This further validates the robustness and generalization capability of RecAgent in challenging environments.

5.3 Qualitative Visualization

Reducing Perceptual Uncertainty. In Figure 4, we present the visualization of the component recommendation module. It is clearly evident that, without the component recommendation module, all UI elements are annotated using the SoM (Yang et al., 2023) method and displayed with numbered bounding boxes, regardless of the number of elements. In practice, lists of UI elements in the form of text are also included as input, leading to substantial input redundancy. When the component recommendation module is applied, only the most relevant UI elements are dynamically retained. In this example, where the goal is to open a shopping application, the preserved elements are the most popular shopping apps in China, such as Pinduoduo and Jingdong. Irrelevant text-based UI element lists are filtered out before input; in this case, the

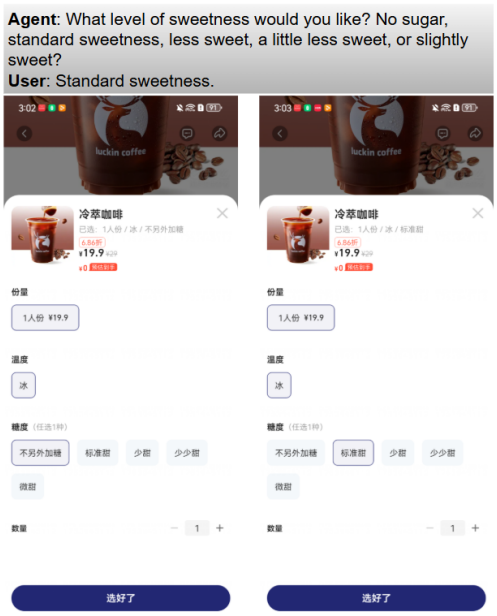

Figure 5: Visualization of the Interaction Agent. The left image shows a coffee-ordering scenario, where the interface presents multiple sweetness options. In this case, the agent proactively asks the user for their preference and, based on the user's response, selects the desired sweetness level, as shown in the right image.

Ablation Setting		AndroidWorld
CRM	RM	Success Rate (%)
✗	✗	40.5
✗	✓	42.1
✓	✗	43.5
✓	✓	47.8

Table 4: Ablation study on component recommendation module (CRM) and retrospection mechanism (RM).

number of UI elements is reduced from 47 to 5. In more complex scenarios, the initial count can even reach hundreds. This significantly reduces input redundancy, thereby decreasing the agent's perceptual uncertainty and enabling more accurate localization of the target UI element.

Reducing Decision Uncertainty. In Figure 5, it can be clearly seen that in scenarios with ambiguous user intent, such as ordering coffee, the interaction agent proactively inquires about the user's preferences and makes appropriate selections based on the user's responses. This interactive mechanism is absent in most existing methods, and it helps reduce decision uncertainty, thereby better aligning with real-world user needs. Without this interactive mechanism, the agent would have to make decisions randomly or rely on default settings, making it difficult to achieve user-satisfying results.

5.4 Ablation Study

Effectiveness of component recommendation module and retrospection mechanism. We present a schematic diagram in Figure 3 demonstrating the effect of using the component recommendation module (CRM) and the retrospection mechanism (RM), which significantly reduces the complexity of the path space. We conducted ablation experiments in Table 4 to quantitatively analyze its effectiveness. It can be seen that using either part alone achieves some performance improvement, but the gains are not very significant. However, when the two are used together, they achieve the best results. Compared to the M3A baselin, our major improvements lie in CRM and RM, the ablation experiments validates the effectiveness of the proposed method.

Recommendation Pathways			ComplexAction
KMP	SMP	LRP	Success Rate (%)
✗	✗	✗	64.5
✓	✗	✗	53.2
✗	✓	✗	56.4
✗	✗	✓	66.1
✓	✓	✓	69.3

Table 5: Ablation study on different recommendation pathways. KMP, SMP and LRP indicate keyword matching pathway, semantic matching pathway and LLM-based intent recommendation pathway, respectively.

Effectiveness of different recommendation pathways. As shown in Table 5, we present quantitative results using different recommendation pathways. It can be observed that when only KMP or SMP is used, performance decreases, as they cannot guarantee accurate recall of the required components. Using only LRP leads to a certain improvement in performance, but the gain is not significant. Only when all three are used together can the best performance be achieved.

6 Conclusion

In this paper, we present RecAgent, an uncertainty-aware GUI agent that addresses input redundancy and decision ambiguity in mobile task automation. RecAgent reduces perceptual uncertainty through a component recommendation mechanism that selectively focuses on relevant UI elements. To handle decision uncertainty, it incorporates an interactive module that seeks user feedback in ambiguous situations. These components are integrated into a unified framework that proactively simplifies inputs and reactively resolves uncertainties. Furthermore, we introduce the ComplexAction dataset to evaluate the success rate of agents in executing specific actions within complex scenarios. Extensive experiments demonstrate the effectiveness of our proposed method.

Limitations

This paper is motivated by addressing two key uncertainties faced by GUI agents during task execution: perceptual uncertainty and decision uncertainty. In the experimental section, we evaluate these two aspects separately, and the Interaction Agent, our proposed component designed to address decision uncertainty, is not involved in the quantitative evaluations. Furthermore, due to limitations in datasets and evaluation frameworks, we have not designed or conducted appropriate experiments to quantitatively assess this interaction mechanism. Instead, we provide only a qualitative evaluation of its interaction capability through Figure 5. This constitutes a major limitation of our work. We plan to explore, in future research, how to quantitatively evaluate this interaction mechanism to validate its effectiveness in reducing decision uncertainty, and to develop relevant benchmark datasets accordingly.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. 2013. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132.

Gongwei Chen, Xurui Zhou, Rui Shao, Yibo Lyu, Kaiwen Zhou, Shuai Wang, Wentao Li, Yinchuan Li, Zhongang Qi, and Liqiang Nie. 2025a. Less is more: Empowering gui agent with context-aware simplification. *Preprint*, arXiv:2507.03730.

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. 2025b. SPA-BENCH: A COMPREHENSIVE BENCHMARK FOR SMARTPHONE AGENT EVALUATION. In *The Thirteenth International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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 *The Thirteenth International Conference on Learning Representations*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

F.O. Isinkaye, Y.O. Folajimi, and B.A. Ojokoh. 2015. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273.

Wenjia Jiang, Yangyang Zhuang, Chenxi Song, Xu Yang, Joey Tianyi Zhou, and Chi Zhang. 2025. Appagentx: Evolving gui agents as proficient smartphone users. *Preprint*, arXiv:2503.02268.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. 2024. Appagent v2: Advanced agent for flexible mobile interactions. *Preprint*, arXiv:2408.11824.

Yinchuan Li, Xinyu Shao, Jianping Zhang, Haozhi Wang, Leo Maxime Brunswic, Kaiwen Zhou, Jiqian Dong, Kaiyang Guo, Xiu Li, Zhitang Chen, Jun Wang, and Jianye Hao. 2025. Generative models in decision making: A survey. *Preprint*, arXiv:2502.17100.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *Preprint*, arXiv:1802.08802.

Xiao Liu, Bo Qin, Dongzhu Liang, Guang Dong, Hanyu Lai, Hanchen Zhang, Hanlin Zhao, Iat Long Iong, Jiadai Sun, Jiaqi Wang, Junjie Gao, Junjun Shan, Kangning Liu, Shudan Zhang, Shuntian Yao, Siyi Cheng, Wentao Yao, Wenyi Zhao, Xinghan Liu, and 11 others. 2024. Autoglm: Autonomous foundation agents for guis. *Preprint*, arXiv:2411.00820.

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 10 others. 2024. Gui agents: A survey. *Preprint*, arXiv:2412.13501.

Wenzhe Niu, Zongxia Xie, Yanru Sun, Wei He, Man Xu, and Chao Hao. 2025. Langtime: A language-guided unified model for time series forecasting with proximal policy optimization. *Preprint*, arXiv:2503.08271.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

ByteDance Seed. 2025. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. *Preprint*, arXiv:2504.13914.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR.

Gemini Team, Petko Georgiev, Ving Ian Lei, and etal. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *Preprint*, arXiv:2403.05530.

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*.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. 2025a. Gui agents with foundation models: A comprehensive survey. *Preprint*, arXiv:2411.04890.

Yiqin Wang, Haoji Zhang, Jingqi Tian, and Yansong Tang. 2024c. Ponder & press: Advancing visual gui agent towards general computer control. *Preprint*, arXiv:2412.01268.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025b. 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. *Preprint*, arXiv:2308.15272.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2025. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations*.

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*.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025a. Gui-explorer: Autonomous exploration and mining of transition-aware knowledge for gui agent. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yuquan Xie, Zaijing Li, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Dongmei Jiang, and Liqiang Nie. 2025b. Mirage-1: Augmenting and updating gui agent with hierarchical multimodal skills. *arXiv preprint arXiv:2506.10387*.

Jun Xu, Yunji Zhao, Wenming Bao, and Chao Hao. 2025a. Fault diagnosis of motor bearing in complex scenarios based on mamba and indicative contrastive learning. *Engineering Applications of Artificial Intelligence*, 146:110216.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2025b. Aguvis: Unified pure vision agents for autonomous gui interaction. *Preprint*, arXiv:2412.04454.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *Preprint*, arXiv:2310.11441.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025. Aria-ui: Visual grounding for gui instructions. *Preprint*, arXiv:2412.16256.

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

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024. Chain of agents: Large language models collaborating on long-context tasks. *Preprint*, arXiv:2406.02818.

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

Yuqi Zhou, Shuai Wang, Sunhao Dai, Qinglin Jia, Zhaocheng Du, Zhenhua Dong, and Jun Xu. 2025. Chop: Mobile operating assistant with constrained high-frequency optimized subtask planning. *Preprint*, arXiv:2503.03743.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. 2024. A survey on efficient inference for large language models. *Preprint*, arXiv:2404.14294.

A The Use of Large Language Models

LLMs were used only during the writing phase, including for polishing the text and providing suggestions to improve the paper's figures.

B Detailed Information of ComplexAction

In this section, we provide a detailed introduction to ComplexAction, the dataset introduced in this work for evaluating GUI agents' success rate in completing specified single-step actions within complex scenarios. In the supplementary materials, we include a folder containing detailed information about the dataset, including a JSON file of all the scenarios ("ComplexAction.json"), as well as a sample that provides a screenshot input in image format, the corresponding accessibility tree input in text format, the specified action, and the ground truth.

As can be seen, the ComplexAction dataset defines a total of five common actions: "click search box", "create new content", "like", "refresh interface", and "sort". The dataset contains 62 complex scenarios in total, with each action corresponding to 20, 10, 12, 10, and 10 scenarios, respectively. These complex scenarios are selected from commonly used Chinese apps, such as the home page of QQ Music. We define a scenario as "complex" if its corresponding screenshot contains a large number of UI elements (e.g., more than 150). Such a high number of UI elements leads to input redundancy and increases the difficulty for large language models (LLMs) to locate key components within the lengthy UI element list.

There are two evaluation approaches. The first is based on given inputs and specified target actions, assessing whether the GUI agent produces the correct target action. Agents can choose their preferred input format, which generally falls into three types: pure visual input (based on screenshots), pure text input (based on the a11y tree), and SoM-based (Yang et al., 2023) input (screenshots with bounding boxes combined with the a11y tree). The second approach involves testing in a real-world environment: the mobile device is manually navigated to the corresponding interface (as specified in the provided JSON file), and the GUI agent's action is executed to determine whether the device reaches the expected resulting state. This outcome must be verified manually; however, given that there are only 62 scenarios, the required effort is manageable. Due to file size limitations, we only include one sample in the supplementary materials. We will release the full dataset after the paper is published, but in the meantime, evaluation using the second approach can still be conducted based on the provided "ComplexAction.json" file.

C Reducing Input Redundancy

In the Introduction section of the main text, we discuss several drawbacks of input redundancy, with one key challenge being the increased computational cost. In Figure 4 of the main text, we present a visualization of the redundancy reduction achieved through our component recommendation module. For clarity and ease of presentation, we only show the changes in the screenshot input with bounding box annotations. In practice, however, the SoM input format uses both annotated screenshots and a11y tree together as input.

During GUI agent execution, the input primarily consists of four components: the screenshot, the a11y tree, a fixed prompt, and historical data stored in the memory unit. Among these, the a11y tree typically consumes the largest number of tokens, even accounting for more than 70% of the total input tokens in many cases. This highlights the significance of reducing redundancy in the a11y tree to improve efficiency and scalability.

Based on our statistical analysis, on average, RecAgent reduces token consumption by approximately **50%** per task execution cycle compared to the M3A (Rawles et al., 2024) baseline method, thanks to the use of the component recommendation module. Furthermore, as demonstrated by the experimental results in Table 5 of the main paper, our approach also enhances the agent's ability to perceive critical information. This indicates that our method effectively mitigates many of the drawbacks caused by input redundancy.

D More Infomation of Interaction Agent

The overall framework of our proposed RecAgent is a modification based on the M3A (Rawles et al., 2024) architecture. Most prompts, including those defining the action space and the overall workflow, follow M3A's original design. However, the Interaction Agent is our novel contribution and does not exist in M3A; accordingly, we designed its prompt from scratch. Its function is straightforward: to issue a request to the user at specific moments to resolve decision uncertainty, as detailed in Sec-

tion 3.6 of the maintext. The specific prompt is as follows:

Listing 1: Prompt template for the Interaction Agent

```
You are an intelligent clarification
    module within a GUI assistant. Your
    role is to determine whether the
    user's intent requires further
    clarification or additional input
    before proceeding.

Ask for clarification **only when**:
1. There are multiple equally valid
    choices that cannot be resolved by
    context or history;
2. A critical piece of information is
    missing, which is necessary to
    proceed with the task;
3. The user's preference is ambiguous
    and would significantly affect the
    outcome.

Make your judgment based on:
1. The user's original instruction:
{goal}
2. Interaction history so far:
{history}
3. Current UI elements available:
{current_elements}

Output a JSON object in the following
    format:
{
  "need_clarification": true or false,
  "clarification_question": "If
      clarification is needed, provide a
       specific and minimal question;
      otherwise leave it as an empty
      string."
}
```

It should be noted that in the quantitative experiments reported in the main text, existing benchmarks such as AndroidWorld do not account for user interaction. Consequently, their experimental setups involve only clear, unambiguous action instructions and exhibit no decision uncertainty, rendering the Interaction Agent unnecessary. Therefore, the Interaction Agent is not employed in these quantitative evaluations. As specified in its prompt, the Interaction Agent requests user feedback only when absolutely necessary. In scenarios where decision uncertainty is absent, which is the case in most benchmark tasks, it simply outputs "false". This behavior does not interfere with the overall execution flow and incurs negligible additional latency. We turned off this function during the quantitative experiments described in the paper.

We primarily verified its effectiveness through the qualitative experiment shown in Figure 5. The test results showed that RecAgent can complete most daily needs, such as ordering takeout and other scenarios that require human feedback.

E Limitations and Future Work

This paper is motivated by addressing two key uncertainties faced by GUI agents during task execution: perceptual uncertainty and decision uncertainty. In the experimental section, we evaluate these two aspects separately, and the Interaction Agent, our proposed component designed to address decision uncertainty, is not involved in the quantitative evaluations. Furthermore, due to limitations in datasets and evaluation frameworks, we have not designed or conducted appropriate experiments to quantitatively assess this interaction mechanism. Instead, we provide only a qualitative evaluation of its interaction capability through Figure 5. This constitutes a major limitation of our work. We plan to explore, in future research, how to quantitatively evaluate this interaction mechanism to validate its effectiveness in reducing decision uncertainty, and to develop relevant benchmark datasets accordingly.

Our work primarily relies on two input modalities: a11y tree and SoM format, and does not consider the recently popular pure vision-based input approaches. In fact, when using only screenshots as input, the screen itself often contains significant redundancy, as critical UI components typically occupy only a small portion of the display. Prior works (Chen et al., 2025a) have mentioned this issue to some extent, but they mainly apply random masking of image regions to increase input diversity, without actively identifying which parts of the image constitute redundant information. Exploring how to reduce perception uncertainty in a vision-only setting by actively identifying and focusing on relevant visual regions remains an important direction for future research. In the future, we hope to extend the idea of reducing perception uncertainty proposed in this paper to vision-only input methods, which should significantly enhance the capabilities of GUI grounding models (Gou et al., 2025; Wu et al., 2024).