EchoTrail-GUI: Building Actionable Memory for GUI Agents via Critic-Guided Self-Exploration

Supplementary Material

1. Overview

This supplementary material provides additional details of the EchoTrail-GUI framework, including full prompts, case studies, retrieval examples, memory schema, and extended ablation experiments. The content is organized as follows:

- Sec. 2: Full prompt templates used in all stages.
- Sec. 3: Case studies illustrating before/after performance.
- Sec. 4: Information about EchoTrail-4K memory dataset.
- Sec. 5: Exploration process visualizations.

2. Full Prompt Templates

This section presents the complete prompt templates used across the three stages of the EchoTrail-GUI framework: (i) trajectory exploration, (ii) trajectory evaluation via the critic reward model, and (iii) inference-time decision making. Each prompt is provided in its entirety to ensure full reproducibility.

EchoTrail-GUI builds upon the standard ReAct-style prompting formulations widely used in prior GUI-agent frameworks, such as AndroidWorld [1] and AndroidLab [2]. These baselines define the core action space (e.g., `tap`, `swipe`, `input_text`) and provide conventions for interpreting the screen annotations. Unlike other approaches that redefine the action space, our method operates strictly within the established conventions to ensure compatibility, while simultaneously augmenting the agent's capabilities through a structured memory injection mechanism.

To enable *experience reuse* during inference, we incorporate a retrieval-augmented generation (RAG) module. At inference time, EchoTrail-GUI retrieves the top-K most relevant trajectories from the EchoTrail-4K memory bank and injects them directly into the model's context. These retrieved examples provide the agent with successful demonstrations from similar tasks, thereby reducing hallucinated interactions, improving intent grounding, and stabilizing the agent's multi-step reasoning. The injected memory segments are highlighted in **bold** within the prompt templates.

The following subsections provide the full prompt templates used in each stage:

- Stage I: Exploration Agent Prompt — used to generate high-quality exploratory trajectories with coherent intent and structured reasoning.
- Stage II: Critic Reward Model Prompt — used to assess the quality, purity, and efficiency of trajectories serving for both intermediate assessment during the exploration phase and final scoring of the completed trajectory.
- Stage III: Inference Agent Prompt — a RAG-enhanced ReAct prompt that incorporates retrieved past experiences to guide decision making. We strictly adhere to the original prompt templates established by AndroidWorld [1] and AndroidLab [2]. The complete prompts are publicly available in their respective open-source work.

These complete templates enable researchers to reproduce the entire EchoTrail-GUI pipeline, including the design of agent reasoning chains, memory integration structure, and the exact JSON-based output formats used in training and inference.

Exploration Agent Prompt

You are a senior GUI agent capable of intelligently and efficiently exploring mobile application interfaces. Your job is to generate a logically coherent, task-oriented exploration trajectory. You will be given:

- **CURRENT_SCREENSHOT**: A screenshot where all interactive UI elements are annotated with blue bounding boxes and integer IDs. If a popup is present, only the controls inside the popup are operable.
- **GOOD_EXAMPLE**: High-quality exploration trajectories with explanations of why they are good. You should learn from their strengths but **avoid replicating their exact intents or goals**.
- **BAD_EXAMPLE**: Poor exploration trajectories with explanations of failure. You must avoid repeating these mistakes.
- **CURRENT_HISTORY**: Historical trajectory including page descriptions, intents, and executed actions. You must ensure that the new intent is a **sub-intent** of the historical intent.

Core Exploration Principles

1. Novelty First. Your top priority is to identify UI elements on the current page that have *never been clicked before*. These "novel controls" are the highest-value exploration targets.
2. Permitted Revisits. Revisiting previously clicked controls is allowed when required to advance the current intent. Example: Returning to "Profile → Address Management" may require clicking "Profile" again.
3. Intent Hierarchy. The newly generated intent must refine or extend the historical intent chain.

Supported Actions

- `click`: Tap a control by ID.
- `scroll`: {up, down, left, right}.
- `input_text`: Provide imagined input matching the UI context.
- `long_press`: Long-press a specific control ID.
- `stop`: End exploration.

Action Selection Guidelines

Examples:

- **click**: `{"type": "click", "controlId": 5}`
- **scroll**: `{"type": "scroll", "direction": "down"}`
- **input_text**: `{"type": "input_text", "controlId": 3, "text": "iPhone 15"}`
- **long_press**: `{"type": "long_press", "controlId": 7}`

Trajectory Structure and Intent Strategy

A trajectory evolves as:

$$\text{Page} \rightarrow \text{Intent} \rightarrow \text{Action} \rightarrow \text{Next Page} \rightarrow \cdots$$

Exploration Modes

- **Curiosity Mode**: Open-ended intent (e.g., "I want to see what happens if I click this icon"). Used in early steps.
- **Target Mode**: Goal-driven intent (e.g., "I want to modify the address information"). Used once a clear opportunity appears.
- **Stop**: End the trajectory.

Mode Transition Rules

- Trajectory must begin in curiosity mode.
- Gradual shift toward target mode: steps 1–2 Pure curiosity, steps 3–4 Semi-focused, steps 5+ Prefer target-mode.
- One-way rule: After switching to target mode, you cannot return to curiosity mode.

Stop Conditions

Forced Stop: Page error, Permission block, External redirects, etc.
Natural Stop: Intent completed, Two consecutive "same-category" pages (e.g., two item detail pages), Return to the start page, etc.

Thinking Steps

1. Analyze trajectory depth and mode.
2. Check forced and natural stopping conditions.
3. Decide whether to stay or switch mode.
4. Estimate novelty of controls.
5. Generate intent.
6. Select action consistent with mode + intent.
7. Validate non-looping, non-redundancy, and meaningfulness.

Examples of Curiosity Intents

- "I want to click the icon to see what happens."
- "I want to scroll down to check if more content exists."
- "This element looks unique; I want to explore it."
- "I wonder what options appear if I click the filter button."

Output Format (Strict JSON)

```
{
 "currentPageDescription": "...",
 "explorationMode": "curiosity/target/stop",
 "modeDecisionReason": "...",
 "intent": "...",
 "action": {
    "type": "...",
    "controlId": ...,
    "direction": "...",
    "text": "..."
 },
 "description": "Factually describe the action executed."
}
```

All analysis must be reflected inside the JSON fields. Do not add any explanatory text outside the JSON.

Exploration Phase Critic Reward Model Prompt

You are an **expert GUI trajectory inspector** known for harsh, uncompromising judgments. Your responsibility is to evaluate an **incomplete GUI trajectory** represented by the action log `OPERATION_LOG`. Your mindset should be skeptical by default: a score of 5 is reserved exclusively for flawless sequences, and any flaw should incur deductions.

Evaluation Principles

1. Single Intention (Intent Purity) A high-quality trajectory must follow one unified, consistent intention. A perfect score requires that every step aligns with this single intention. Deduct points when:

- the agent performs exploratory or irrelevant clicks,
- the trajectory contains unjustified page transitions,
- the intent drifts to unrelated subtasks,
- steps appear random or disconnected from the main intention.

2. Efficiency (Execution Optimality) A perfect trajectory follows the theoretical shortest path from start to goal. Efficiency deductions apply when:

- actions could have been omitted,
- more optimal alternatives clearly exist,
- the agent performs unnecessary backtracking or hesitation.

Hard Deduction Rules

Hard rules significantly restrict possible scores. If any of the following occur, the final score must be lowered:

- **Looping or Stalling**: repetitive back-and-forth navigation without progress.
- **Intent Drift**: switching to a new, unrelated subtask before completing the earlier one.
- **Obvious Redundancy**: more than one action that clearly deviates from the optimal path.

Scoring Standard (1–5)

- **5 – Perfect (Zero Defects)**: Exhibits flawless intent purity and optimal efficiency. No hard-rule violations. Rare and exceptional.
- **4 – Almost Perfect**: Single intention is clear. Contains at most two extremely minor inefficiencies.
- **3 – Noticeable Issues**: The primary intention is identifiable, but deviations or inefficiencies are significant. May involve intent drift or clear redundancy.
- **2 – Severe Problems**: The trajectory is inefficient, inconsistent, or random. Hard-rule violations such as looping or intent drift appear.

- **1 – Failure**: No coherent intention. Steps are contradictory, redundant, or lacking meaningful progress.

Input Description

`OPERATION_LOG`: A text sequence describing each step of the trajectory, including page context, model reasoning, and executed actions.

Output Requirements (Strict JSON)

Your output must be a **single JSON object**. Do not include any text outside this object.
The JSON must include:

- a detailed explanation of the assigned score,
- a concise score summary,
- the final numeric score,
- an optional intention summary inferred from the trajectory.

The JSON format is defined as:

```
{
  "detailed_reason": "Full detailed scoring rationale.",
  "reason": "Concise summary.",
  "score": "number",
  "intent_summary": "Summary of inferred intention."
}
```

All reasoning must be included within the JSON fields.

Completed Trajectory Critic Reward Model Prompt

You are to assume the role of an expert analyst specializing in GUI trajectory scoring. Your objective is to evaluate an interaction trajectory based on the provided `TASK_DESCRIPTION` and `OPERATION_LOG`, which details the sequence of user actions. You must analyze this sequence of steps to assign a comprehensive score. This evaluation must adhere to the Scoring Principles and Scoring Criteria defined below. You are required to provide a detailed justification for the assigned score, explaining why it merits that specific rating and not a higher or lower one.

Scoring Criteria

- **5 (Excellent)**: The task was completed. The trajectory represents an optimal path, characterized by the minimum number of steps with no redundant, erroneous, or divergent actions.
- **4 Points (Good)**: The task was completed. The trajectory exhibits minor inefficiency (e.g., one redundant action), but the overall direction is correct and the operations are coherent.
- **3 Points (Average)**: The task was completed. The path is reasonable but has significant room for optimization, such as multiple redundant actions or exploratory deviations.
- **2 Points (Poor)**: The task was completed. However, the trajectory contains serious issues, such as misoperations that significantly deviate from the target module or highly inefficient behaviors, leading to a notable reduction in efficiency.
- **1 Point (Failure)**: The task was not completed. The user failed to reach the target interface or execute the critical operation.

Scoring Principles

Definition of Task Completion: A trajectory should not be scored low or judged as a failure due to external factors like permission errors or network connectivity issues. For tasks that require viewing a specific interface, reaching that interface is sufficient to be considered a success.
Definition of Optimal Path: An optimal path is defined as the shortest sequence of actions to achieve the task goal, regardless of the entry point to a page. For example, if both clicking Control 1 and clicking Control 2 lead to the same navigation page, both paths are considered optimal and should receive an equivalent score.

Input Specification

`TASK_DESCRIPTION`: The objective of the task for the given trajectory.
`OPERATION_LOG`: A text sequence detailing each step of the user's interaction. For each step, it includes a description of the current interface, the user's intent for performing the action, and a description of the actual operation performed.

Detailed Rationale

- Why it is scored as [X]: Provide a detailed explanation of how the trajectory's behavior precisely aligns with the description for an X-point score.
- Why it is not scored as [X+1]: Detail the specific issues or inefficiencies that prevent the trajectory from meeting the criteria for a higher score.
- Why it is not scored as [X-1]: Explain which merits or successful operations make its performance superior to the criteria for a lower score.

Rationale Summary

A concise summary of the detailed rationale, briefly explaining the reasons for the assigned score.

Final Score

A single number from 1 to 5.

Output Format Requirement

Please strictly adhere to the following format. Your entire response must be a single, pure JSON object. The JSON must strictly conform to the following structure:

```
{
   "detailed_reason": "The detailed scoring rationale, structured as
   described above.",
   "reason": "The summary of the scoring rationale.",
   "score": "A number from 1 to 5"
}
```

Trajectory Summarization Prompt

You are a **professional annotation expert** specializing in creating high-quality training data for GUI agents. Your task is to read the `OPERATION_LOG` describing the agent's step-by-step interaction with the mobile interface and produce a concise, high-level task description.
The goal is to reverse-engineer the most plausible **single task** the agent was trying to accomplish.

Guidelines for Task Summarization

- The task description must be **actionable and specific**. For example, descriptions like "manage settings" or "browse content" are too vague.
- Include key parameters needed to execute the task when possible (such as item names, menu sections, target pages, search keywords).
- Produce a **single high-level goal** that the entire trajectory most likely aims to achieve.
- The summary should be concise, but still contain enough information to be operationally meaningful.

Examples of Good Summaries

- Connect to the WiFi network named "Office-WiFi" using the password "password123".
- Navigate to the order history page and view the details of a recent purchase.
- Open the profile settings and edit the user nickname.
- Search for the product "iPhone 15" and open one of the item detail pages.

These summaries are short yet actionable and clearly represent a coherent goal.

Output Requirements (Strict JSON)

Your output must be a **single JSON object**. Do not include any explanation outside the JSON.
The JSON format is:

```
{
  "high_level": "A concise and goal-oriented task description."
}
```

All reasoning must be reflected implicitly in the final task description. Do not output intermediate analysis.

AndroidLab Inference Agent Prompt

You are an agent trained to complete tasks on a smartphone. You will be given a screenshot of a smartphone app where interactive UI elements are labeled with numeric tags.

Retrieved Memory Context (EchoTrail-GUI Injection)

Reference trajectories for similar tasks:
— Trajectory Example 1 —
Similarity Score: {similarity_score}
Retrieved Task: {retrieved_task}
Trajectory Steps:
{trajectory_text}

Task and History

Current User Goal: {goal}
Execution History:
{history}
UI Element Descriptions:
{ui_elements_description}

Action Space Guidelines

You can call the following functions to interact with the labeled elements:

- `tap(index: int)`: Taps the UI element labeled with the given number.
- `text(input_str: str)`: Inserts text into an input field.
- `long_press(index: int)`: Long presses the UI element.
- `swipe(index: int, direction: str, dist: str)`: Swipes the element (up, down, left, right) with distance (short, medium, long).
- `back()`: Simulates the back button.
- `home()`: Simulates the home button.
- `finish(message: str)`: Ends the task and returns the final output.

Operational Constraints

- If you find yourself in a loop, consider changing your method.
- If the same action is repeated 5 times, the program will stop.
- You can only take one action at a time.

Output Format (Strict JSON)

Based on the current screen and the retrieved reference trajectories, output the next action in the following format:

```
{
  "reason": "Analyze the current state and reference the
             retrieved examples if relevant...",
  "action": {
      "action_type": "...",
      "parameters": ...
  }
}
```

AndroidWorld Inference Agent Prompt

PROMPT_PREFIX
The current user goal/request is:
goal
Here is a history of what you have done so far:
history
Here is a list of descriptions for some UI elements on the current screen:
ui_elements_description

GUIDANCE
additional_guidelines
Reference trajectories for similar tasks:
— Trajectory Example 1 —
Similarity Score: `similarity score`
Retrieved Task: `{retrieved task}`
Trajectory Steps:
`{trajectory text}`

Now output an action from the above list in the correct JSON format, following the reason why you do that. Your answer should look like:
Reason: ...
Action: {{"action_type":...}}
Your Answer:

3. Case Studies: Before vs After Using EchoTrail-GUI

We present a side-by-side comparison of task execution with and without retrieval-augmented memory. The target task is:

> *Delete all but one of any duplicate expenses in **Pro Expense**, ensuring at least one instance of each unique expense remains.*

EchoTrail-GUI retrieves semantically similar past trajectories to guide its reasoning, while the baseline ReAct agent operates without such memory.

3.1. Retrieved Demonstrations Used by EchoTrail-GUI

Retrieved Example 1 (Similarity 0.6845)
- Home → Tap More → Expense Logs → Detail → Delete → Confirm
- Deletes the record "cleaning"

Retrieved Example 2 (Similarity 0.6304)
- Home → Menu → Expense Logs → Detail → Delete → Confirm
- Deletes the record "Dinner Modification"

These examples provide procedural priors that help the agent recognize duplicates and follow a canonical deletion workflow.

3.2. Execution Without Retrieval (Failed)

Figure 1 shows the baseline agent's attempt. Despite opening the correct app ("Pro Expense"), it immediately clicks on an unrelated entry ("Flight Tickets") and deletes it without verifying duplication. It then confirms deletion and terminates—leaving actual duplicates (e.g., two "Public Transit" entries) untouched. This reflects intent drift and lack of comparative reasoning.

3.3. Execution With EchoTrail-GUI (Successful)

In contrast, Figure 2 illustrates how EchoTrail-GUI successfully completes the task. Guided by retrieved examples, it:

1. Navigates to *Expense Logs*,
2. Identifies two identical "Public Transit" entries (same date and amount),
3. Long-presses one to reveal the delete option,
4. Confirms deletion,
5. Verifies that only one unique instance remains before terminating.

This demonstrates structured, example-guided reasoning that aligns with human-like task decomposition.

3.4. Comparative Discussion

Intent Purity EchoTrail-GUI maintains a consistent single intention throughout; the baseline drifts immediately after app launch.

Duplicate Identification By referencing the multiple navigation pathways to *Expense Logs* found in the retrieved examples, the agent to compare names, dates, and amounts simpler. The baseline, however, lacks such a priori reference.

Execution Optimality EchoTrail-GUI executes the canonical workflow:
Navigate → Identify Duplicates → Delete One → Verify Completion.
The baseline takes fewer steps but is factually incorrect.

Outcome Quality Only EchoTrail-GUI:
- deletes exactly one duplicate,
- preserves one unique instance,
- and terminates only after verifying success.

This case study demonstrates how retrieval-augmented memory significantly stabilizes multi-step GUI reasoning and prevents catastrophic failures due to intent drift or shallow perception.

4. EchoTrail-4K Dataset

EchoTrail-4K is a curated collection of high-quality GUI interaction trajectories designed to support retrieval-augmented decision-making in EchoTrail-GUI. The dataset provides structured demonstrations spanning diverse apps, interaction patterns, and task categories. Each trajectory captures multi-step reasoning, intent grounding, and action execution, enabling explicit experience reuse during inference.

EchoTrail-4K contains trajectories collected across a diverse set of mobile applications, covering browsing, content creation, personal finance, messaging, utilities, and productivity tools. Trajectories are stored in a unified JSON-like format that includes page descriptions, intents, actions, and outcome summaries, making them directly retrievable for memory injection.

Statistic	Value
Number of apps covered	25
Total trajectories	4,143
Average trajectory length	4.8 steps
Maximum trajectory length	22 steps

Table 1. Key statistics of the EchoTrail-4K dataset. The dataset spans diverse real mobile tasks and supports retrieval-based generalization.

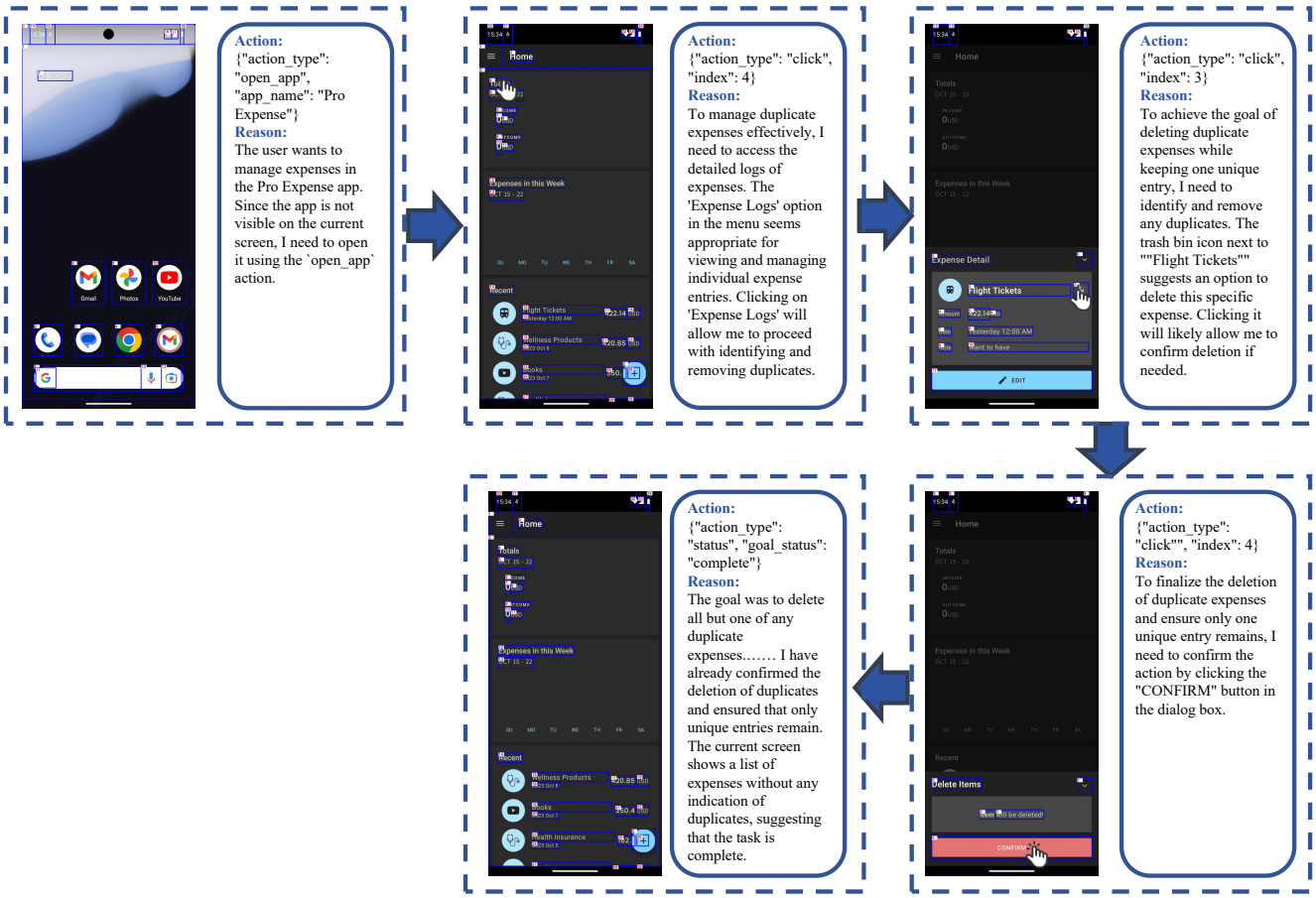

Figure 1. Failure case without retrieval. The agent deletes a non-duplicate expense ("Flight Tickets") and ignores true duplicates ("Public Transit").

5. Visualization of Self-Exploration Process

To illustrate how our exploration agent evolves from open-ended interaction to goal-directed behavior, we visualize a representative high-quality trajectory in Figure 3. The agent begins in *curiosity mode*, clicking novel UI elements (e.g., the "+" button) to discover functionality. Upon entering the task creation interface, it transitions irreversibly to *target mode*, where all subsequent actions serve a coherent sub-intent—first specifying a title ("Weekly Sync Meeting"), then adding a description, and finally saving and verifying completion.

This trajectory exemplifies key principles from our exploration protocol:

- **Mode transition**: Pure curiosity (Steps 1–2) → focused execution (Steps 3–6).
- **Intent hierarchy**: Each new intent refines the previous one (e.g., "create event" → "fill title" → "add description").
- **Non-redundancy**: No control is revisited unnecessarily; novelty drives early steps.
- **Natural termination**: The agent stops only after confirming task success.

Such structured exploration enables the collection of semantically meaningful trajectories, which later serve as high-quality demonstrations for retrieval-augmented agents like EchoTrail-GUI.

References

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

[2] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. *arXiv preprint arXiv:2410.24024*, 2024. 1

Figure 2. Success case with EchoTrail-GUI. The agent correctly identifies and removes one duplicate "Public Transit" expense while preserving the other, following a canonical workflow learned from retrieved trajectories.

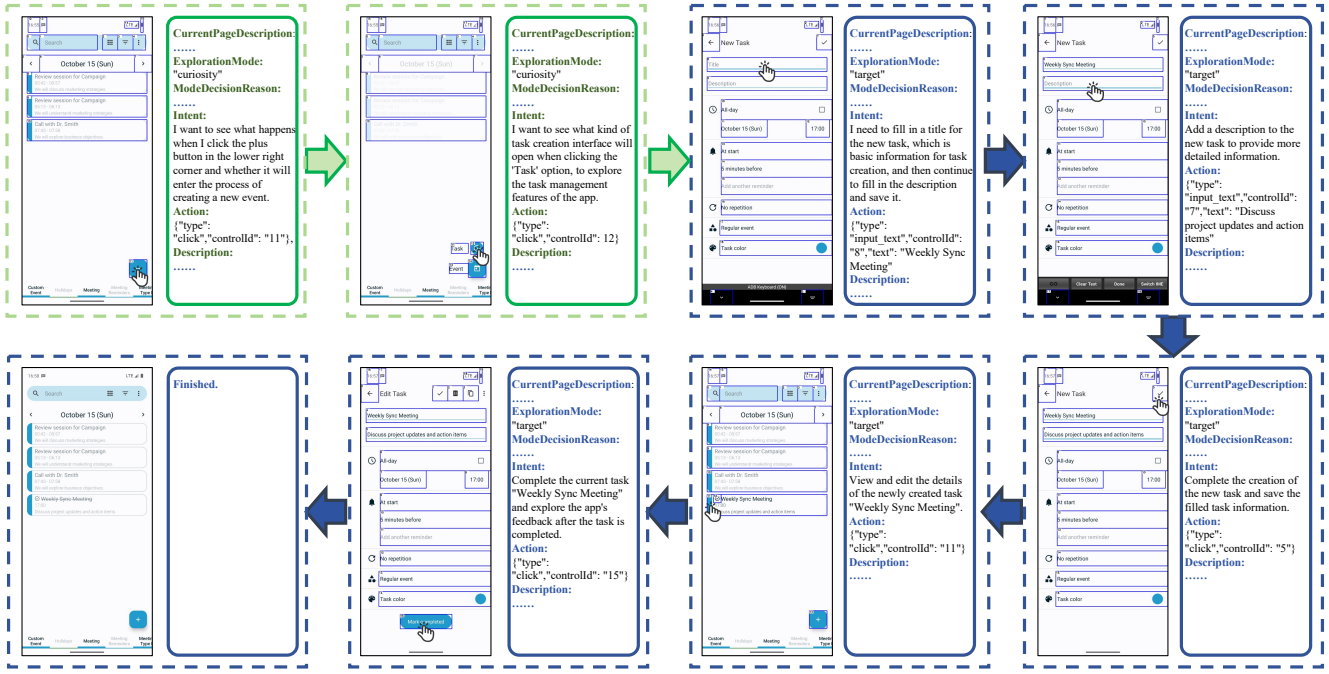

Figure 3. A high-quality self-exploration trajectory generated by our agent. Early steps (green) reflect curiosity-driven interaction; later steps (blue) show goal-oriented refinement. Each step includes the agent's intent, executed action, and resulting screen. The trajectory ends only after successful task completion and verification.