AndroidLong: LLM-based Android Agents Struggle with Long Looping Tasks

Supplementary Material

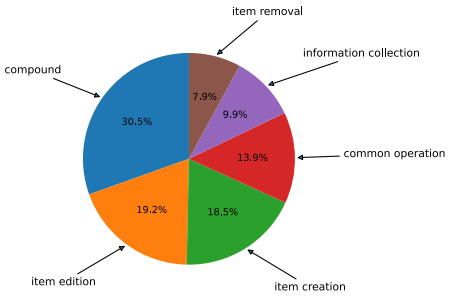

Figure 4. Distribution of task categories in the AndroidLong test set.

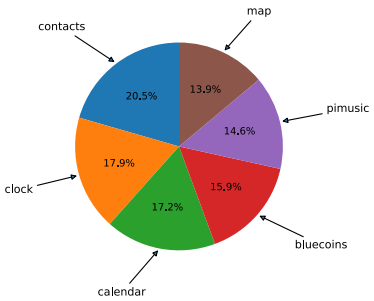

Figure 5. Distribution of applications in the AndroidLong test set.

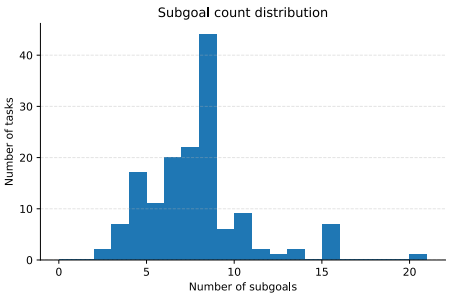

Figure 6. Histogram of the number of subgoals per task.

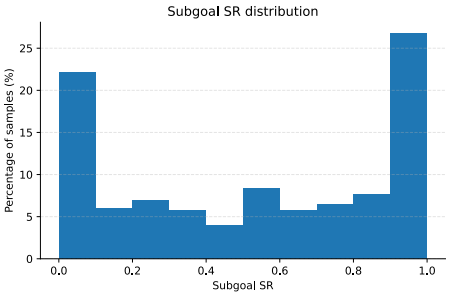

Figure 7. Empirical distribution of subgoal success rates (Subgoal SR) over all task–model pairs.

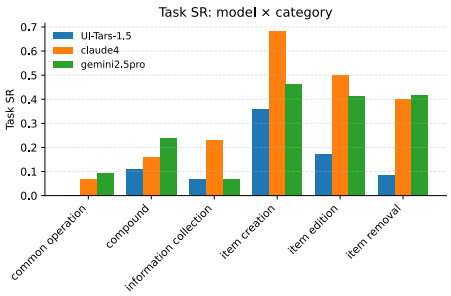

Figure 8. Task-level success rate (Task SR) of different models, grouped by task category.

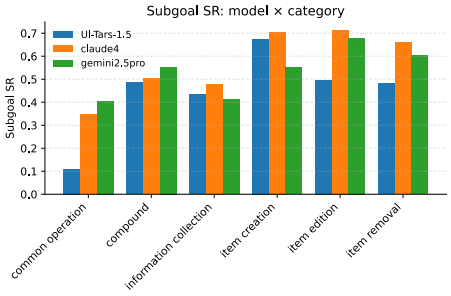

Figure 9. Subgoal-level success rate (Subgoal SR) of different models, grouped by task category.

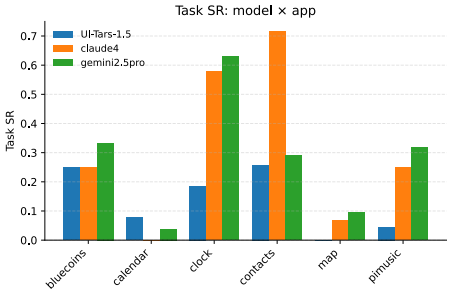

Figure 10. Task-level success rate (Task SR) of different models, grouped by application.

A. AndroidLong Data Distributions

Figures 4–11 provide additional analysis of the AndroidLong benchmark. Figure 4 shows that our long-horizon tasks cover six semantic categories without any single category dominating the test set, while Figure 5 illustrates a similarly balanced distribution across six everyday Android applications. The histogram in Figure 6 reports the number of subgoals per task and indicates that AndroidLong contains a broad spectrum of interaction lengths rather than being restricted to short scripts. Figure 7 depicts the empirical distribution of subgoal success rates (Subgoal SR) over all task–model pairs, which is spread across the full range instead of concentrating near

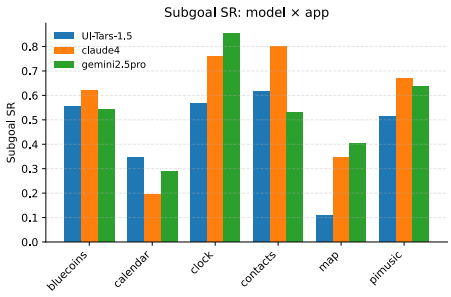

Figure 11. Subgoal-level success rate (Subgoal SR) of different models, grouped by application.

0 or 1, suggesting that the benchmark is neither trivially easy nor degenerate. Figures 8 and 9 further break down model performance by task category, showing that item-creation and item-edition tasks are substantially easier than compound and information-collection tasks at both the task level (Task SR) and the subgoal level. Finally, Figures 10 and 11 report the same metrics grouped by application (e.g., *Contacts*, *Clock*, *Calendar*, and *Maps*), revealing large variability in success rates across apps and consistent performance gaps between UI-Tars-1.5, Claude 4, and Gemini 2.5 Pro that mirror the main results discussed in Section 5.

B. SFT Dataset Details

Our cold-start SFT data leverages datasets provided by the MobileRL work, including 11.3k steps from AndroidControl dataset and 12.2k human-annotated steps. The human-annotated data covers all applications and action space supported in AndroidWorld and AndroidLab, following the annotation protocol proposed in the AndroidLab paper as well as the corresponding privacy protection policies. In addition, we organized human annotation for 100 instances sampled from the AndroidLong training set, in order to prevent the model from becoming biased toward short-horizon tasks.

C. Training Detials

C.1. Settings

The base model used for training is GLM-4.1V-9B-Base. The model operates in a single-step prediction paradigm. Models with the memory module also follow single-step prediction, with the memory content added only to the CoT and no necessity for another inference call. The AVD environment is reset after each evaluation is complete in order to keep all evaluations independent. All evaluation results on AndroidLong are reported as the average of three independent runs (i.e., pass@1 under three samples). We provide the Task SR along with the standard deviation of the three runs in table 6.

Absence of open-app actions in AndroidLong. AndroidLong is developed from AndroidLab and naturally inherits the "in-app" setting. However, our SFT data includes open-app actions, so the trained models also support the open-app operation. We evaluated our model on AndroidLong without automatically entering applications (with task instructions explicitly indicating which app needs to be launched). The results did not show a significant performance gap, which is displayed in the last line in table 6.

Table 6. Task Success Rate with standard deviation (n=3)

Models	AndroidLong Task SR
SFT (no memory)	10.1 ± 0.8
+ GRPO (no memory)	28.4 ± 1.6
SFT (with memory)	18.9 ± 0.3
+ GRPO (with memory)	**33.1** ± 0.2
(w/o auto-entering app)	32.9 ± 0.2

C.2. Hyperparameters

During the SFT phase, we fine-tuned the model for two epochs using a cosine learning-rate schedule that decayed from 1×10^{-5} to 1×10^{-6}. Training was carried out with `Swift`, where packing mode was activated to improve computational efficiency. All images were processed at their original resolution without any compression.

For the RL phase, we built upon the GRPO implementation in `Verl` and incorporated several customized enhancements. A summary of the key hyperparameters used in this stage is provided in Table 7.

Table 7. Summary of Key Hyperparameters

Component	Hyperparameter	Value
Data	Max Prompt Length	16384
Data	Max Response Length	2048
Data	Shuffle	True
Data	Concurrency	256
Actor / Policy	Strategy	fsdp2
Actor	Learning Rate	1e-6
Actor	Gradient Clipping	1.0
Actor	Clip Ratio	0.2
Actor	PPO Epochs	1
Actor	Entropy Coefficient	0.001
Actor	KL Loss Coefficient	0.001
Actor	Use Dynamic BSZ	True
Rollout	Sampling Temperature	1.0
Rollout	Max New Tokens	2048
Rollout	Number of Samples (n)	8
Rollout	Max Turns	100
Rollout	Max Pixels	500000
Rollout	Min Pixels	65536

Table 8. Detailed performance of models on each application in AndroidLong

Models	Bluecoins		Calendar		Clock	
	Subgoal SR	Task SR	Subgoal SR	Task SR	Subgoal SR	Task SR
Proprietary Models						
Claude-Sonnet-3.7-20250219-thinking	0.41	0.25	0.22	0.00	0.68	0.59
Claude-Sonnet-4-20250514-thinking	0.52	0.21	0.20	0.00	0.54	0.41
Gemini-2.5-Flash	0.27	0.17	0.16	0.00	0.50	0.22
Gemini-2.5-Pro	0.50	0.29	0.29	0.04	0.72	0.63
GLM-4.5V	0.29	0.08	0.13	0.00	0.30	0.11
GPT-4.1-2025-04-14	0.55	0.33	0.22	0.04	0.78	0.37
GPT-4o-2024-11-20	0.43	0.21	0.22	0.00	0.66	0.26
Seed1.5-VL-thinking-250428	0.51	0.21	0.23	0.00	0.62	0.48
UI-Tars-1.5	0.56	0.25	0.34	0.08	0.57	0.19
Open Models						
GLM-4.1V-9B-Thinking	0.40	0.17	0.10	0.00	0.36	0.07
GUI-Owl-32B	0.04	0.00	0.06	0.00	0.18	0.00
MobileRL-GLM-4.1V-9B	0.23	0.08	0.09	0.00	0.41	0.11
Qwen2.5-VL-7B-Instruct	0.03	0.00	0.01	0.00	0.18	0.00
Qwen2.5-VL-72B-Instruct	0.10	0.04	0.04	0.00	0.27	0.07
Qwen3-VL-8B-Thinking	0.22	0.00	0.04	0.00	0.33	0.11
Qwen3-VL-235B-a22B	0.24	0.00	0.03	0.00	0.35	0.11
UI-Tars-72B-DPO	0.02	0.00	0.02	0.00	0.27	0.11
UI-Tars-1.5-7B	0.19	0.00	0.06	0.00	0.35	0.07
Open Agentic Frameworks						
MobileUse	0.15	0.00	0.09	0.00	0.38	0.04
Mobile-Agent-v3	0.05	0.00	0.04	0.00	0.26	0.04
UI-Venus-Navi	0.12	0.08	0.14	0.00	0.54	0.15
AndroidLong (ours)						
SFT (no memory)	0.47	0.17	0.33	0.04	0.65	0.19
+ GRPO (no memory)	0.59	0.25	0.45	0.19	0.86	0.59
SFT (with memory)	0.64	0.33	0.36	0.04	0.75	0.44
+ GRPO (with memory)	0.81	0.50	0.61	0.31	0.86	0.59

D. Case Study

D.1. Error Analysis

Insufficient Exploration. In Figure 12, the goal of task is to modify the status of transactions in the list. The evaluated model performs correct operations at first, but intermediately stops after traversing the list present on the screen without further exploration. To complete the task, the model is expected to perform swiping and continue modifying left transactions.

Turns Limit Exceeded. In Figure 13, the task requires creation of 4 new events, which implies a long trajectory and risks of exceeding turns limitation. The evaluated model performs redundant operations(such as step 7-9, step 25-27) and inefficient operations(such as step 11-15, step 29-33), both of which lead to a waste of turns quota.

Insufficient Reflection. In Figure 14, operations of typing text(such as step 3, 8, 13) do not take actual effect because of lack of focus on any editable text box. However, the evaluated model is not aware and continues as if all operations are effective. This is a typical case of insufficient reflection.

Inconsistency. In Figure 15, the evaluated model performs correct operations until step 12. Then the model tries to check if other items need modifying. However, it repeatedly checks the same item through step 13-14 and step 17-20. Finally the model ends in a loop of tapping on the same position, which is also the position of a checked item in the history. The model acts as if it forgets the progress and history of the trajectory.

Formality Error. This type of error means that the model's output does not match the requirement in the system prompt,

Table 9. Detailed performance of models on each application in AndroidLong

Models	Contacts		Map		Pimusic	
	Subgoal SR	Task SR	Subgoal SR	Task SR	Subgoal SR	Task SR
Proprietary Models						
Claude-Sonnet-3.7-20250219-thinking	0.66	0.58	0.20	0.05	0.41	0.09
Claude-Sonnet-4-20250514-thinking	0.72	0.65	0.25	0.05	0.61	0.23
Gemini-2.5-Flash	0.06	0.00	0.21	0.00	0.31	0.05
Gemini-2.5-Pro	0.40	0.29	0.39	0.10	0.48	0.32
GLM-4.5V	0.14	0.00	0.04	0.00	0.15	0.05
GPT-4.1-2025-04-14	0.65	0.39	0.26	0.00	0.54	0.18
GPT-4o-2024-11-20	0.68	0.45	0.15	0.00	0.36	0.05
Seed1.5-VL-thinking-250428	0.62	0.42	0.28	0.10	0.52	0.14
UI-Tars-1.5	0.61	0.26	0.11	0.00	0.52	0.05
Open Models						
GLM-4.1V-9B-Thinking	0.35	0.06	0.12	0.00	0.37	0.00
GUI-Owl-32B	0.08	0.00	0.04	0.00	0.12	0.00
MobileRL-GLM-4.1V-9B	0.27	0.10	0.09	0.00	0.17	0.00
Qwen2.5-VL-7B-Instruct	0.04	0.00	0.01	0.00	0.07	0.00
Qwen2.5-VL-72B-Instruct	0.12	0.00	0.04	0.00	0.10	0.00
Qwen3-VL-8B-Thinking	0.08	0.00	0.05	0.00	0.18	0.00
Qwen3-VL-235B-a22B	0.21	0.03	0.10	0.00	0.19	0.00
UI-Tars-72B-DPO	0.32	0.10	0.05	0.00	0.03	0.00
UI-Tars-1.5-7B	0.24	0.03	0.16	0.00	0.27	0.00
Open Agentic Frameworks						
MobileUse	0.17	0.03	0.10	0.00	0.33	0.00
Mobile-Agent-v3	0.15	0.00	0.02	0.00	0.02	0.00
UI-Venus-Navi	0.24	0.10	0.06	0.00	0.24	0.05
AndroidLong (ours)						
SFT (no memory)	0.47	0.16	0.09	0.00	0.27	0.00
+ GRPO (no memory)	0.61	0.45	0.19	0.00	0.54	0.09
SFT (with memory)	0.44	0.26	0.17	0.00	0.41	0.00
+ GRPO (with memory)	0.60	0.35	0.32	0.05	0.58	0.09

which is specially set for each model and listed in section F.

D.2. Training Methods

Figure 16, Figure 17, Figure 18, and Figure 19 reflects different performance of our post-trained models on the same task: *Change recurrence to yearly for the upcoming events in list, if it is not happened today.*

We observe that along with the progress of SFT(no memory)-GRPO(no memory)-SFT(with memory)-GRPO(with memory), the model gradually grabs a stable ability to perform edition on items in the given application. After GRPO with memory, the model conquers the problem of insufficient exploration and gains a complete success through active exploration on unseen items.

E. Limitation

Our work suffers from several limitations. We list them here and will make best effort to solve them in future works.

- Limited range of applications. For the reproducibility, we took advantage of the framework provided by AndroidLab and designed tasks within the application range in that work. All experiments were carried out under offline environment on Android emulators, which means most SNS apps and online video apps are not covered in our work. Despite the difficulty in evaluation and reproducibility, we will look into these apps to make our work more comprehensive.
- Limited models evaluated. In this work, we incorporated most leading proprietary models and instruction-tuned version of several renowned open sourced models and agentic frameworks. We believe that our conclusion makes suffi-

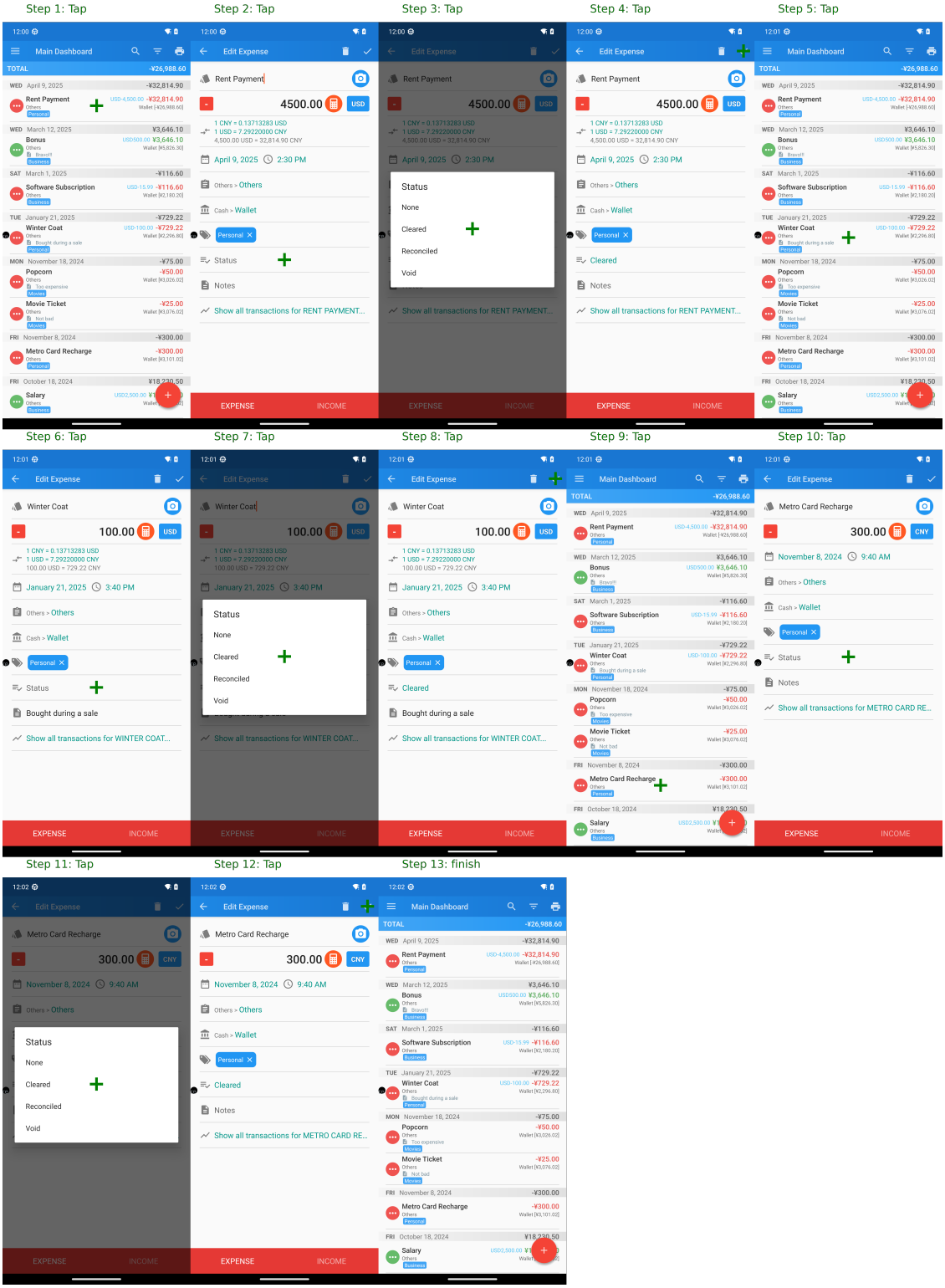

Figure 12. Example of insufficient exploration. Task Goal: *For transactions in the list, if the transaction is a expense more than 200 CNY, change its Status to 'Cleared'.* The trajectory gains 3 of 7 subgoal success.

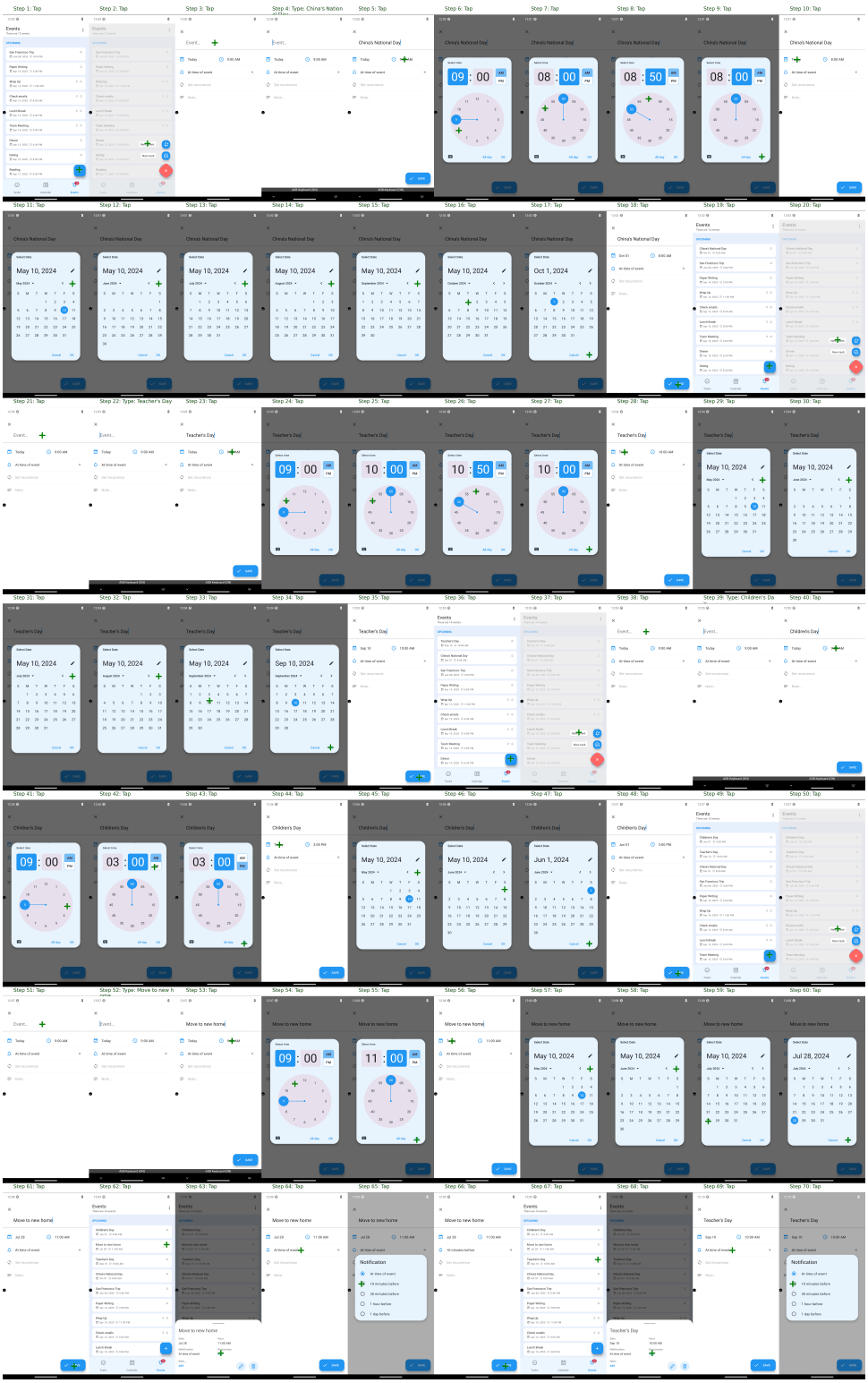

Figure 13. Example of turns limit exceeded. Task Goal: *Create 4 new events: China's National Day at 8 AM on October 1; Teacher's Day at 10 AM on September 10; Children's Day at 3:00 PM on June 1; Move to new home at 11:00 AM on July 28. Then check all the new events, if it is in the morning, set its Notification to 10 minutes before.*' The trajectory gains 2 of 4 subgoal success.

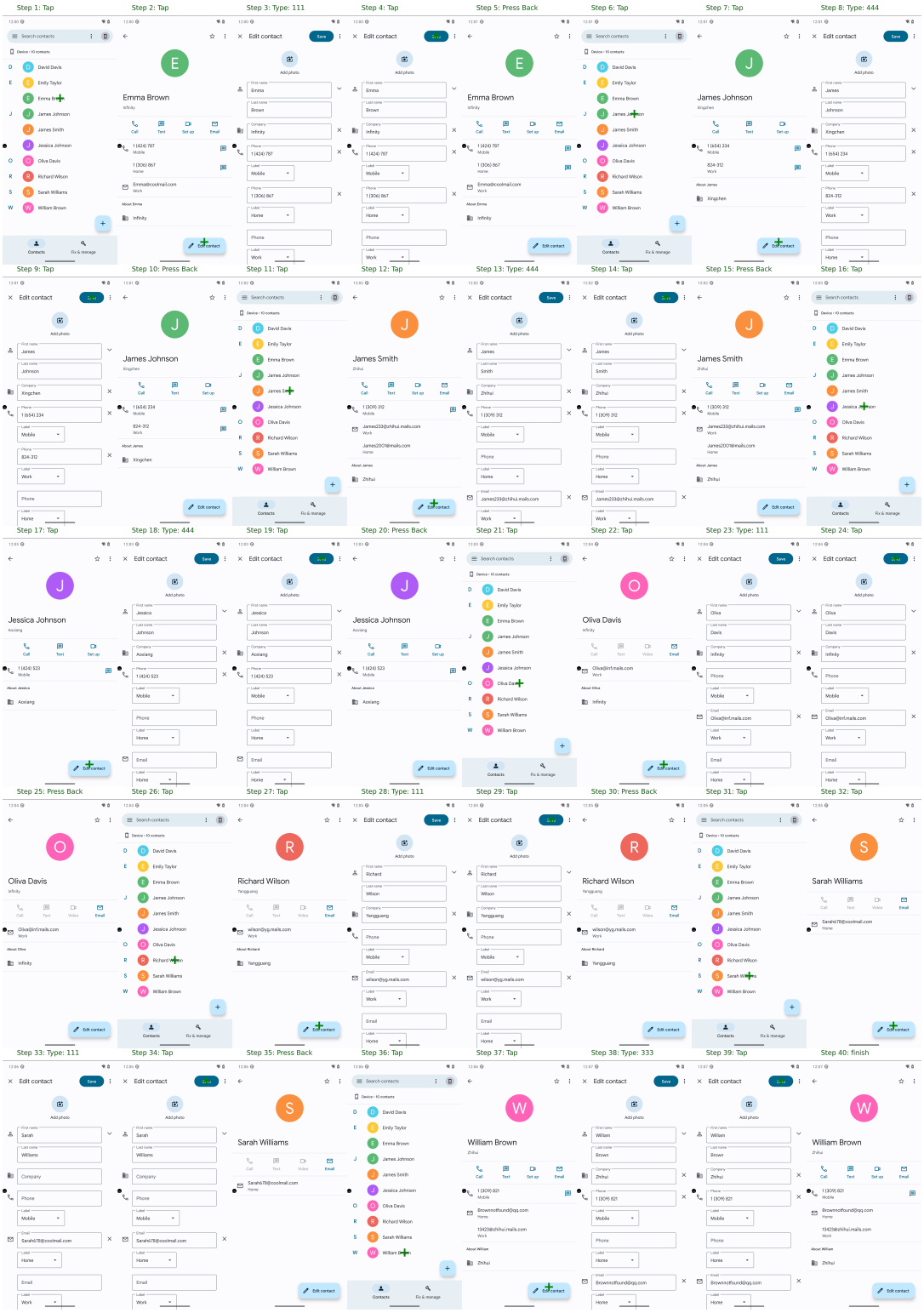

Figure 14. Example of insufficient reflection. Task Goal: *For last 8 contacts in the contacts list, if they have only one email, add a new phone number 111; else if the name contains neither letters 'p' and 's', add a new phone number 333; Else, add a new phone number 444.* The trajectory gains 0 of 8 subgoal success.

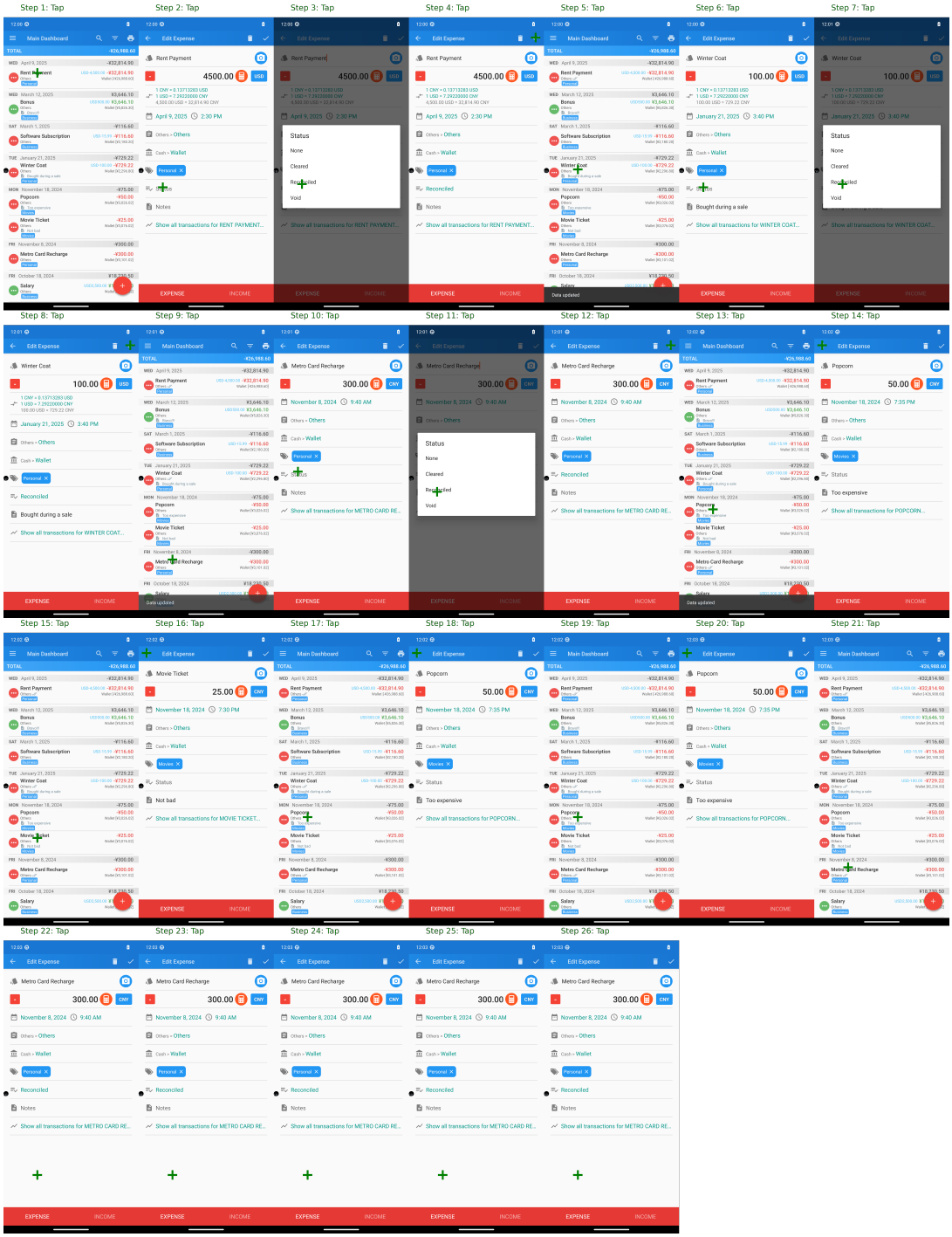

Figure 15. Example of Inconsistency. Task Goal: *For transactions in the list, if it is labeled Personal or Birthday, change its Status to Reconciled. Stop when changing 6 transactions.* The trajectory gains 3 of 5 subgoal success.

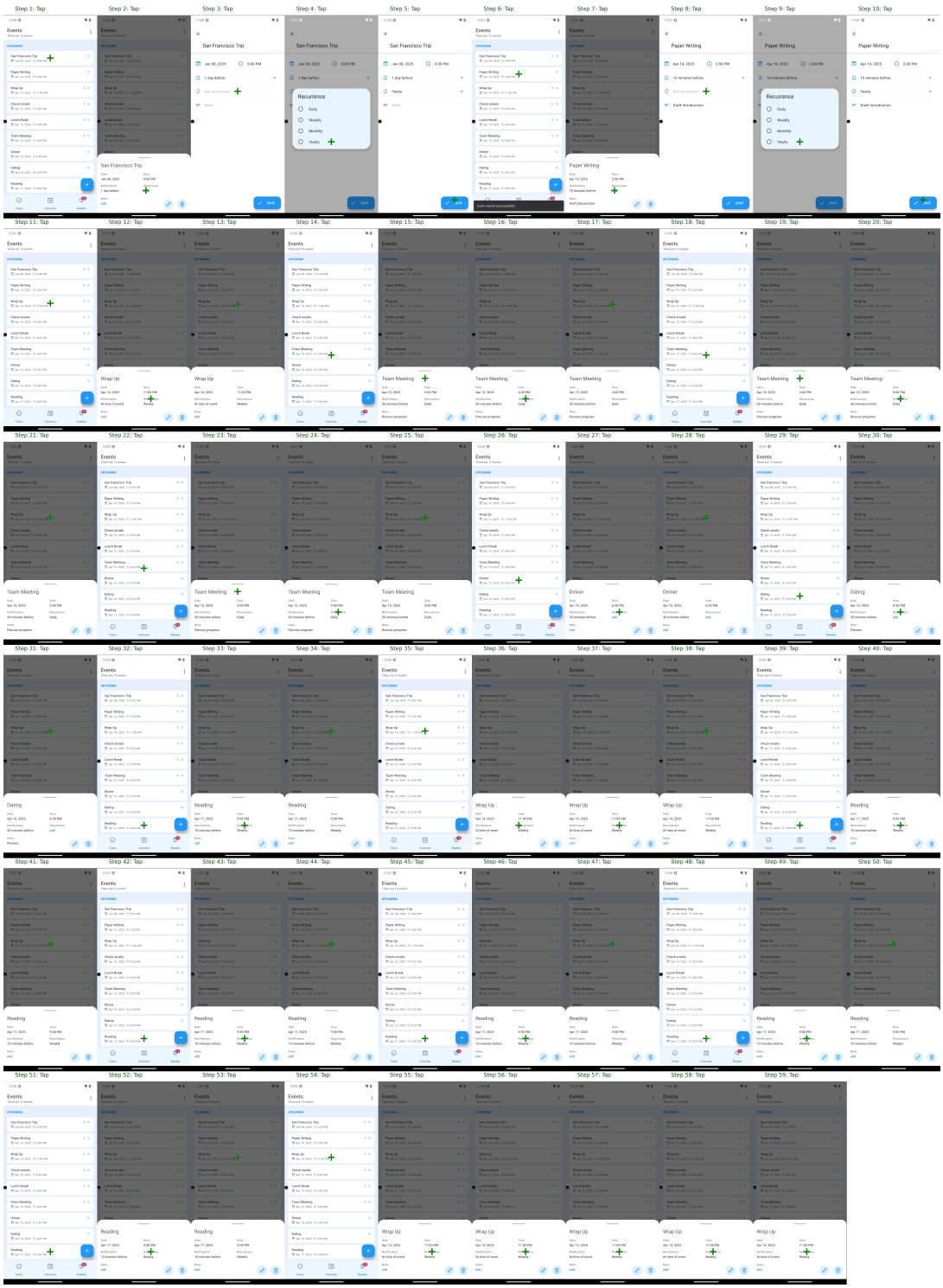

Figure 16. Trajectory of our **SFT(no memory)** model. It is observed that the model cannot perform the edition on recurrence of events. Besides, lack of reflection on the effect of its past operation makes much part of the trace ineffective. The trajectory gains 0 of 8 subgoal success.

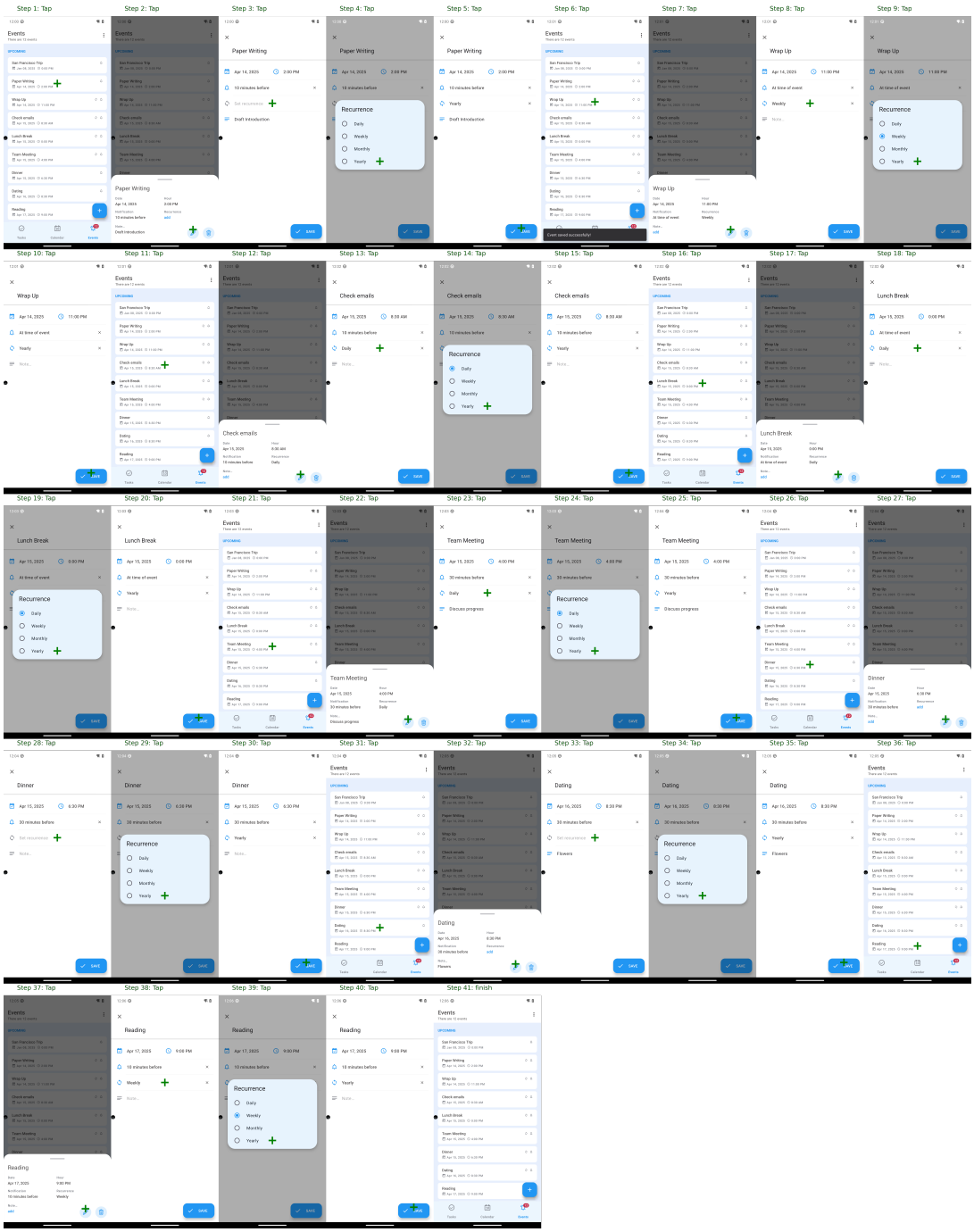

Figure 17. Trajectory of our **SFT + GRPO(no memory)** model. Compared to the SFT(no memory) model, this version manages to perform edition on events. However, insufficient exploration leads to an early stop in the trajectory. The trajectory gains 5 of 8 subgoal success.

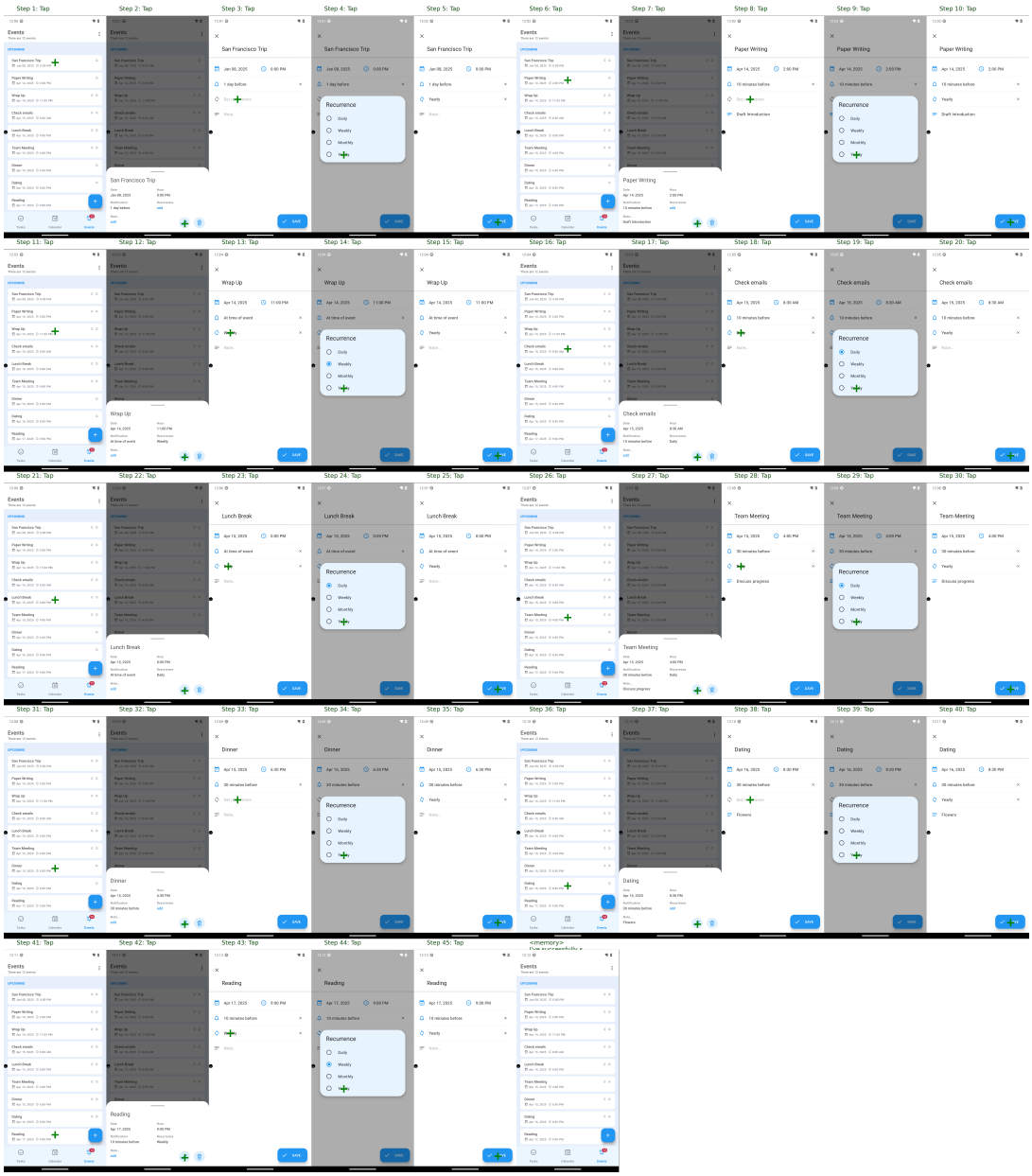

Figure 18. Trajectory of our **SFT(with memory)** model. Through SFT data with memory, model learns a regular and stable way to perform edition on events but insufficient exploration still exists. The trajectory gains 5 of 8 subgoal success.

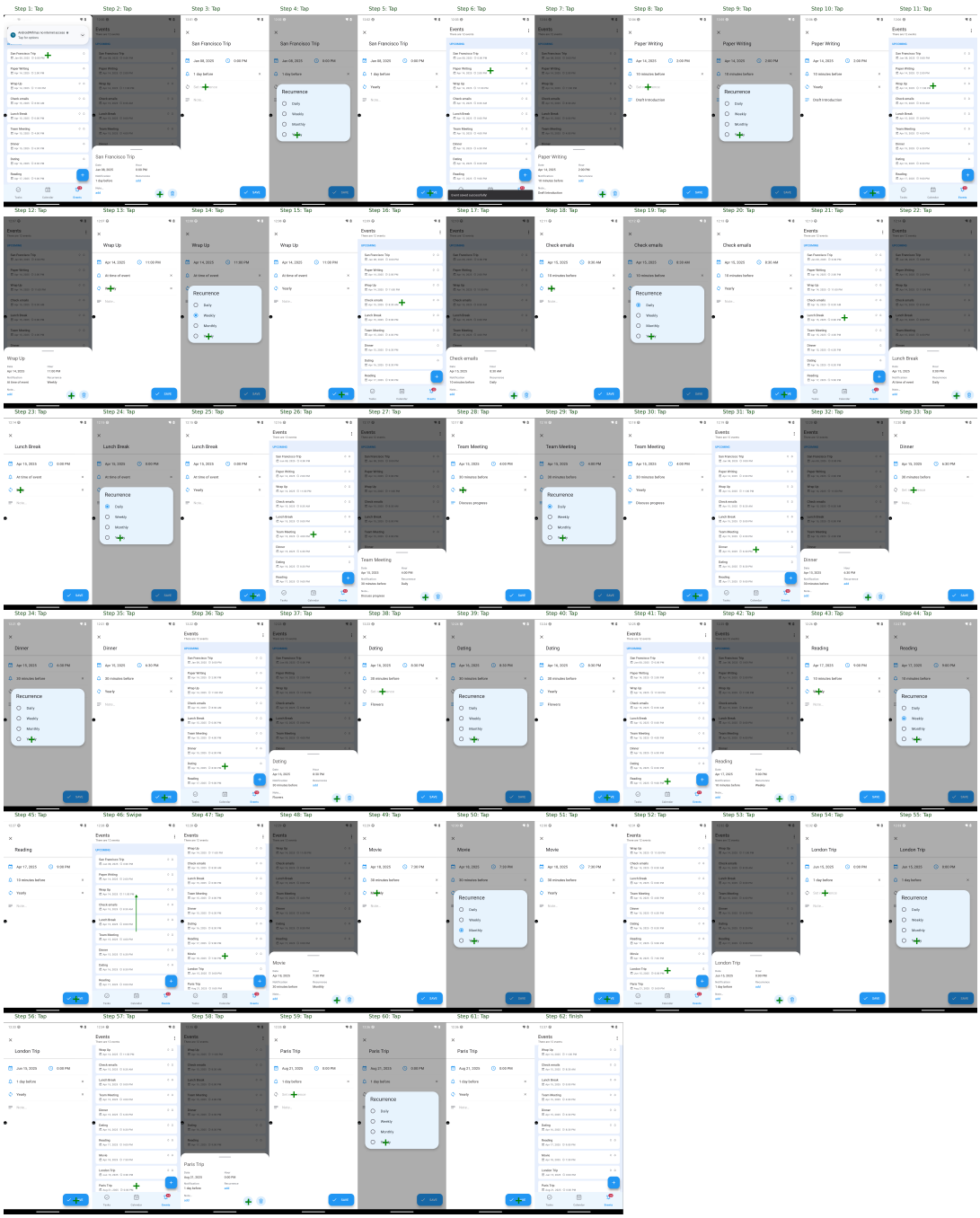

Figure 19. Trajectory of our **SFT + GRPO(with memory)** model. The model keeps the skill of editing events. Additionally, it performs active exploration(swiping operation on step 46), which contributes to subsequent edition on unseen items in earlier trajectories. The trajectory gains 8 of 8 subgoal success, which means a complete success on this task.

cient sense while more models can be taken into consideration.

- We take advantage of the concise formality of looping tasks. To conduct more comprehensive research into long-horizon tasks on mobile platform, more diversifying forms of long-horizon tasks need to be explored.

F. Prompts

F.1. Proprietary Models

The prompt for **Proprietary Models**

```
# Setup
You are a professional Android
operation agent assistant that can
fulfill the user's high-level
instructions. Given a screenshot of
the Android interface at each step,
you first analyze the situation, then
plan the best course of action using
Python-style pseudo-code.

# More details about the code
Your response format must be
structured as follows:

Think first: Use <think>...</think> to
 analyze the current screen, identify
key elements, and determine the most
efficient action.
Provide the action: Use <answer>...</
answer> to return a single line of
pseudo-code representing the operation
.

Your output should STRICTLY follow the
 format:
<think>
[Your throught]
</think>
<answer>
[Your operation code]
</answer>

- **Tap**
  Perform a tap action on a specified
  screen area. The element is a list
  of 4 integers, representing the
  coordinates of the top-left and
  bottom-right corners of the
  rectangle. You must choose one
  element from the current state.
  **Example**:
  <answer>
  do(action="Tap", element=[100, 200,
  150, 250])
  </answer>
- **Type**
  Enter text into the currently
  focused input field.
  **Example**:
  <answer>
  do(action="Type", text="Hello World
  ")
  </answer>
- **Swipe**
  Perform a swipe action in a
  specified direction (`"up"`, `"down
  "`, `"left"`, `"right"`).
  The swipe distance can be `"long"`,
  `"medium"` (default), or `"short"`.
  You can add the element to the
  action to specify the swipe area.
  The element is a list of 4 integers,
   representing the coordinates of the
   top-left and bottom-right corners
  of the rectangle. You must choose
  one element from the current state.
  **Examples**:
  <answer>
  do(action="Swipe", direction="up",
  dist="long", element=[100, 200, 150,
   250])
  </answer>
- **Long Press**
  Perform a long press action on a
  specified screen area.
  You can add the element to the
  action to specify the long press
  area. The element is a list of 4
  integers, representing the
  coordinates of the top-left and
  bottom-right corners of the
  rectangle. You must choose one
  element from the current state.
  **Example**:
  <answer>
  do(action="Long Press", element
  =[200, 300, 250, 350])
  </answer>
- **Launch**
  Launch an app. Try to use launch
  action when you need to launch an
  app. Check the instruction to choose
   the right app before you use this
  action.
  **Example**:
  <answer>
  do(action="Launch", app="Settings")
  </answer>
- **Back**
  Press the Back button to navigate to
```

```
   the previous screen.
  **Example**:
  <answer>
  do(action="Back")
  </answer>
- **Finish**
  Terminate the program and optionally
   print a message.
  **Example**:
  <answer>
  finish(message="Task completed.")
  </answer>


REMEMBER:
- Think before you act: Always analyze
 the current UI and the best course of
 action before executing any step, and
 output in <think> part.
- Only ONE LINE of action in <answer>
part per response: Each step must
contain exactly one line of executable
 code.
- Generate execution code strictly
according to format requirements.
- A screenshot of the current page and
 the UI location information will be
provided at the same time. If your
action involves location information,
try to choose the known UI location
information.
```

F.2. Qwen2.5-VL

The prompt for **Qwen2.5-VL Models**

```
You are a helpful assistant.

# Tools

You may call one or more functions to
assist with the user query.

You are provided with function
signatures within <tools></tools> XML
tags:
<tools>
{"type": "function", "function": {"
name_for_human": "mobile_use", "name":
 "mobile_use", "description": "Use a
touchscreen to interact with a mobile
device, and take screenshots.
* This is an interface to a mobile
device with touchscreen. You can
perform actions like clicking, typing,
 swiping, etc.
* Some applications may take time to
start or process actions, so you may
need to wait and take successive
screenshots to see the results of your
 actions.
* The screen's resolution is {width}x{
height}.
* Make sure to click any buttons,
links, icons, etc with the cursor tip
in the center of the element. Don't
click boxes on their edges unless
asked.", "parameters": {"properties":
{"action": {"description": "The action
 to perform. The available actions are
:
* `key`: Perform a key event on the
mobile device.
    - This supports adb's `keyevent`
    syntax.
    - Examples: "volume_up", "
    volume_down", "power", "camera", "
    clear".
* `click`: Click the point on the
screen with coordinate (x, y).
* `long_press`: Press the point on the
 screen with coordinate (x, y) for
specified seconds.
* `swipe`: Swipe from the starting
point with coordinate (x, y) to the
end point with coordinates2 (x2, y2).
* `type`: Input the specified text
into the activated input box.
* `answer`: Output the answer.
* `system_button`: Press the system
button.
* `open`: Open an app on the device.
* `wait`: Wait specified seconds for
the change to happen.
* `terminate`: Terminate the current
task and report its completion status
.", "enum": ["key", "click", "
long_press", "swipe", "type", "answer
", "system_button", "open", "wait", "
terminate"], "type": "string"}, "
coordinate": {"description": "(x, y):
The x (pixels from the left edge) and
y (pixels from the top edge)
coordinates to move the mouse to.
Required only by `action=click`, `
action=long_press`, and `action=swipe
`.", "type": "array"}, "coordinate2":
{"description": "(x, y): The x (pixels
 from the left edge) and y (pixels
from the top edge) coordinates to move
 the mouse to. Required only by `
```

```
action=swipe`.", "type": "array"}, "
text": {"description": "Required only
by `action=key`, `action=type`, `
action=answer`, and `action=open`.", "
type": "string"}, "time": {"
description": "The seconds to wait.
Required only by `action=long_press`
and `action=wait`.", "type": "number
"}, "button": {"description": "Back
means returning to the previous
interface, Home means returning to the
 desktop, Menu means opening the
application background menu, and Enter
 means pressing the enter. Required
only by `action=system_button`", "enum
": ["Back", "Home", "Menu", "Enter"],
"type": "string"}, "status": {"
description": "The status of the task.
 Required only by `action=terminate
`.", "type": "string", "enum": ["
success", "failure"]}}, "required": ["
action"], "type": "object"}, "
args_format": "Format the arguments as
 a JSON object."}}
</tools>

For each function call, return a json
object with function name and
arguments within <tool_call></
tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments":
 <args-json-object>}
</tool_call>

# Response format

Response format for every step:
1) Thought: one concise sentence
explaining the next move (no multi-
step reasoning).
2) Action: a short imperative
describing what to do in the UI.
3) A single <tool_call>...</tool_call>
 block containing only the JSON: {"
name": <function-name>, "arguments": <
args-json-object>}.

Rules:
- Output exactly in the order: Thought
, Action, <tool_call>.
- Be brief: one sentence for Thought,
one for Action.
- Do not output anything else outside
those three parts.
- If finishing, use action=terminate
in the tool call.
```

F.3. Qwen3-VL

The prompt for **Qwen3-VL Models**

```
You are a helpful assistant.

# Tools

You may call one or more functions to
assist with the user query.

You are provided with function
signatures within <tools></tools> XML
tags:
<tools>
{"type": "function", "function": {"
name": "mobile_use", "description": "
Use a touchscreen to interact with a
mobile device, and take screenshots.\n
* This is an interface to a mobile
device with touchscreen. You can
perform actions like clicking, typing,
 swiping, etc.\n* Some applications
may take time to start or process
actions, so you may need to wait and
take successive screenshots to see the
 results of your actions.\n* The
screen's resolution is 999x999.\n*
Make sure to click any buttons, links,
 icons, etc with the cursor tip in the
 center of the element. Don't click
boxes on their edges unless asked.", "
parameters": {"properties": {"action":
 {"description": "The action to
perform. The available actions are:\n*
 `click`: Click the point on the
screen with coordinate (x, y).\n* `
long_press`: Press the point on the
screen with coordinate (x, y) for
specified seconds.\n* `swipe`: Swipe
from the starting point with
coordinate (x, y) to the end point
with coordinates2 (x2, y2).\n* `type`:
 Input the specified text into the
activated input box.\n* `answer`:
Output the answer.\n* `system_button`:
 Press the system button.\n* `wait`:
Wait specified seconds for the change
to happen.\n* `terminate`: Terminate
the current task and report its
completion status.", "enum": ["click",
 "long_press", "swipe", "type", "
answer", "system_button", "wait", "
```

```
terminate"], "type": "string"}, "
coordinate": {"description": "(x, y):
The x (pixels from the left edge) and
y (pixels from the top edge)
coordinates to move the mouse to.
Required only by `action=click`, `
action=long_press`, and `action=swipe
`.", "type": "array"}, "coordinate2":
{"description": "(x, y): The x (pixels
 from the left edge) and y (pixels
from the top edge) coordinates to move
 the mouse to. Required only by `
action=swipe`.", "type": "array"}, "
text": {"description": "Required only
by `action=type` and `action=answer
`.", "type": "string"}, "time": {"
description": "The seconds to wait.
Required only by `action=long_press`
and `action=wait`.", "type": "number
"}, "button": {"description": "Back
means returning to the previous
interface, Home means returning to the
 desktop, Menu means opening the
application background menu, and Enter
 means pressing the enter. Required
only by `action=system_button`", "enum
": ["Back", "Home", "Menu", "Enter"],
"type": "string"}, "status": {"
description": "The status of the task.
 Required only by `action=terminate
`.", "type": "string", "enum": ["
success", "failure"]}}, "required": ["
action"], "type": "object"}}}
</tools>

For each function call, return a json
object with function name and
arguments within <tool_call></
tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments":
 <args-json-object>}
</tool_call>

# Response format

Response format for every step:
1) Thought: one concise sentence
explaining the next move (no multi-
step reasoning).
2) Action: a short imperative
describing what to do in the UI.
3) A single <tool_call>...</tool_call>
 block containing only the JSON: {"
name": <function-name>, "arguments": <
args-json-object>}.

Rules:
- Output exactly in the order: Thought
, Action, <tool_call>.
- Be brief: one sentence for Thought,
one for Action.
- Do not output anything else outside
those three parts.
- If finishing, use action=terminate
in the tool call.
```

F.4. UI-Tars

The prompt for **UI-Tars Models**

```
You are a GUI agent. You are given a
task and your action history, with
screenshots. You need to perform the
next action to complete the task.
## Output Format
```
Thought: ...
Action: ...
```
## Action Space

click(point='<point>x1 y1</point>')
long_press(point='<point>x1 y1</point
>')
type(content='') #If you want to
submit your input, use "\\n" at the
end of `content`.
scroll(point='<point>x1 y1</point>',
direction='down or up or right or left
')
open_app(app_name=\'\')
drag(start_point='<point>x1 y1</point
>', end_point='<point>x2 y2</point>')
press_home()
press_back()
finished(content='xxx') # Use escape
characters \\', \\", and \\n in
content part to ensure we can parse
the content in normal python string
format.


## Note
- Use {language} in `Thought` part.
- Write a small plan and finally
summarize your next action (with its
target element) in one sentence in `
Thought` part.

## User Instruction
```

```
{instruction}
```