

UINavBench: A Framework for Comprehensive Evaluation of Interactive Digital Agents

Supplementary Material

The supplementary material is structured as follows:

1. In Appendix A, we share the specific prompt used to design the baselines discussed in the main manuscript.
2. In Appendix B, we compare our benchmark against other popular existing online benchmarks.
3. In Appendix C, we mention a few more details of our automatic task validation criteria with examples.
4. In Appendix D, we specify additional details on how the devices are staged.
5. In Appendix E, we show multiple examples of how our agents perform on UINavBench tasks.
6. To help the readers better understand the task taxonomy, we visualize a small set of trajectories and their corresponding task taxonomy annotations. These are attached as an HTML file with the name `task_taxonomy.html`
7. Finally, We also share the full trajectory of the Reflexion + Text baseline. The trajectories contain the full-text representation, reasoning, and reflection steps. As part of this supplementary, these are attached as the HTML files in `Reflexion_Text_Trajectories` folder.

A. Additional Details on Methods

We use the OpenAI API to implement our agent. We use the GPT4o (gpt-4o-2024-11-20) snapshot. To ensure that the outputs from the model can be parsed correctly, we use the structured outputs and represent actions as tools.

A.1. ReACT

We provide the system prompt in Fig. 5. The structured output prompt used to guide the agent to produce valid actions is shown in Fig. 6

A.2. ReACT + Text

The system prompt used by this agent is the same as the one used for ReACT + Text baseline (Fig. 5). We modify the system prompt to include an additional instruction to use the textual description of the UI. Similarly, we modify the user-message at every time step to include those details. This agent uses the same prompt Fig. 6 for the output as the previous baseline.

A.3. Reflexion + Text

The system prompt used by this agent is presented in Fig. 7. Note that here we give additional instructions to the agent to use the action feedback provided by the agent as part of the action history. The prompt used by the critic is shown in

Fig. 8. This agent also uses the same prompt Fig. 6 for the output as the previous baseline. For the critic, we use the structured outputs feature to produce a consistent style of feedback at every time step. The prompt is shown in Fig. 9

A.4. Reflexion + UGround

The system prompt and the prompt for structured outputs are shared in Fig. 10 and Fig. 11 respectively. Note, how the prompts are changed to instruct the GPT agent to produce text descriptions of the element instead of set-of-marks id. The UGround 72B model then uses the text descriptions to ground the description into a pixel-level location on the screen.

A.5. UI-TARS

As mentioned in the UI-TARS code repository, the UI-Tars agent can store upto 5 observations in input. We use the prompts provided by the authors in the UI-Tars code repository. We noticed that the UI-TARS agent often fails to call terminate action, even after the goal is achieved. To address this, we modified the prompt to ask the agent to terminate if the goal is achieved. This led to a small boost in performance. The final prompt is shown in Fig. 12 As shown in the figure, UI-Tars was trained with a different scroll action. Instead of predicting the four directions, the agent was turned to predict the start and the end coordinates $[x_1, y_1, x_2, y_2]$. We converted the coordinates into one of the four directions to be compatible with our environment.

B. Comparison to Existing Online Benchmarks

We provide a comparison against two popular existing online benchmarks in Tab. 8 – OSWorld and AndroidWorld. Our benchmark is different from these benchmark in multiple unique ways. First, compared to desktop benchmarks like OSWorld, our benchmark focuses on mobile-use cases. Computer use often involves complex text-entry and input while mobile usage is dominated by visual content, search and navigation. Indeed, most OSWorld tasks (68%) are text-entry on atleast one of the three apps: Sheets, PPTs and Word, whereas UINav balances assistance, data-entry, search and navigation (Tab. 8. Second, mobile UIs present unique navigation challenges different from desktop UIs. Navigation is usually performed by clicking on visual icons, scrolling or navigating through hierarchical menus. Whereas in desktop UIs, larger parts of the apps are visible on the screen, and can be interacted with. Quantitatively, our

System Prompt for GPT-SOM ReACT Agent

You are an AI assistant designed to help users navigate and interact with mobile applications. At the beginning of each task, you will be provided with a description of the task.

Then, at each step, you will be provided with:

1. the screen image with detected elements.
2. the history of actions taken on the environment.
3. the list of available tools that you can use.

Here are some more guidelines:

1. Your task is to analyze the current screen, think step-by-step and provide the next action.
2. If the task has been completed, you should call the terminate action.
3. Avoid getting stuck trying to call the same action repeatedly. If you are stuck, try to find alternative ways to complete the task.
4. Try to accomplish the task with the least number of actions.

User Message per step for GPT-SOM + Reflect Agent

Goal: {goal}
 Set of Marks Image
History: {action history with chain of thought}
Instructions: Based on the goal, current screen, and history of actions, think step-by-step and provide the next action.

Figure 5. System Prompt and step-by-step User Message for GPT-SoM + ReACT Agent

benchmark evaluates agents on diverse set of tasks on 36 apps, compared to OSWorld’s 12 apps.

Compared to AndroidWorld, our benchmark focuses on a larger set of apps (36 vs 20). 25% of tasks in **UINavBench** require multi-app navigation, compared to AndroidWorld’s 7%. Scaling to diverse number of apps, was made possible by using VLM +UI Detection as evaluators instead of AXTree/Database. Writing evaluations that leverage the internal database of apps is prohibitively expensive. It will require understanding the internal API of each app, which is not scalable and for most apps, not even possible due to privacy concerns and closed-source nature of the apps.

We believe, co-existence of multiple benchmarks across use-cases and platforms ensures robust evaluations.

C. Task Validator Details

We provide some examples of detailed task validation metrics for multiple tasks in this part. For each task, we will describe the user prompt as the goal, and task validation

criteria that will be used to evaluate the partial success and final success of agent trajectories. We will cover key examples to showcase all types of task validators mentioned previously, including rule-based criteria, multi-screen criteria, VLM-based criteria, and logical compositions of these criteria.

C.1. Notes task

User prompt "Create a new note ‘Groceries’, containing a bulleted list of apple, banana, and yogurt".

Criteria

- TextContains(text="<")
- TextContains(text="Groceries"),
- TextClose(
text_1="< ",
text_2="Groceries",
x_axis=True,
y_axis=True)
- TextContains(
text="apple",



Figure 6. Structured Output Prompt provided to the GPT based agent such that it produces valid actions

- label="Checkbox_NotSelected"),
- TextContains(
 text="banana",
 label="Checkbox_NotSelected"),
- TextContains(
 text="yogurt",
 label="Checkbox_NotSelected")
- LogicalComposition: All the above criteria must be satisfied (AND logic) on the same screen.

Criteria Category. Rule-based Criteria.

Explanation. Using TextContains on the text "<" and "Groceries" makes sure that the back button and "Groceries" text are both on the same screen; using TextClose to make sure the text "Groceries" and "<" are close, ensuring the text "Groceries" is the title instead of only as text. Checking TextContains on "apple", "banana" and "yogurt" with the label "Checkbox_NotSelected" ensures these items are listed and are listed as bullet lists not just as text. We show an example of successful and failure trajectory as marked by this task criteria in Figure 13.

C.2. Calendar Task

User prompt: "Create a recurring weekly event called 'Sync' starting today with Tester3 at 3:00 PM."

Criteria

- TextContains_MultiImage(text="Sync")
- TextContains_MultiImage(text="Tester3")
- TextContains_MultiImage(text="3:00 PM")
- TextContains_MultiImage(text="Repeat, Every Week")
- VLM-Eval_MultiImage(
 prompt="Here are multiple phone frames concatenated together in sequential order on completing a task 'Create a recurring weekly event called Sync starting today with Tester3 at 3:00 pm', could you help to verify from this sequence of image whether the task has been completed? Check if the event name contains the word 'Sync' (not case sensitive), and the event starts at 3:00 pm today and the event is with Tester3 and the event repeats every week.")
- LogicalComposition: All the above criteria must be individually satisfied (AND logic) on at least one screen.

System Prompt for GPT-SOM + Text + Reflexion Agent

You are an AI assistant designed to help users navigate and interact with mobile applications.

At the beginning of each task, you will be provided with a description of the task.

Then, at each step, you will be provided with:

1. the screen image with detected elements.
2. textual description of all detected elements.
3. the history of actions taken on the environment and any feedback from the critic about the actions
4. the list of available tools that you can use.

Your task is to analyze the goal, current screen, history of actions and feedback about the last action, think step-by-step and provide the next action.

Here are some more guidelines:

1. If the task has been completed, you should call the terminate action.
2. Avoid getting stuck trying to call the same action repeatedly. If you are stuck, try to find alternative ways to complete the task.
3. Try to accomplish the task with the least number of actions.
4. If the previous action failed, take corrective action by taking an alternative action.

User Message per step for GPT-SOM + Reflect Agent

Goal: {goal}

 Set of Marks Image

Text Representation of UI: {text representation of UI}

History: {action history with chain of thought}

Instructions: Based on the goal, current screen, history of actions, and feedback about the last action, think step-by-step and provide the next action.

Figure 7. System Prompt and step-by-step User Message for GPT-SOM + Text + Reflexion Agent

Criteria Category: Rule-based, Multi-screen, VLM-based.

Explanation: The `TextContains_MultiImage` criteria ensure that across multiple screenshots, the event is named "Sync," includes the participant "Tester3," starts at "3:00 PM" today, and is set to repeat every week. The `VLM-Eval_MultiImage` criterion uses a Vision-Language Model to analyze the sequence of images and verify that the event has been set up correctly with all specified details. The `LogicalComposition` indicates that all criteria must be met to consider the task successfully

completed. We show an example of successful and failure trajectory as marked by this task criteria in Figure 14.

D. Device Setup and Task Staging Details

We evaluate all our episodes using a VNC connection to 20-30 *physical devices* for benchmarking. Devices undergo a *full factory reset* after each episode, and are re-staged with user accounts, apps, and data ensuring a clean state every time. Cloud sync is disabled so local changes do not persist on the cloud. Note, that no tasks in the same app depend on

System Prompt for the Critic in the Reflexion Agent

You are an expert in interacting with and navigating mobile applications. Your task is to provide useful feedback for a user to achieve a provided goal .

You will be given:

1. the screen image with detected elements before the human takes an action.
2. the description of the action the human takes along with any reason and references to detected elements before.
3. the screen image with detected elements after the human takes an action.

Your task is to think carefully and analyze the current action, and the screen before and after the action. Use this information to describe the changes in the screen and provide feedback about whether the action was successful or not.

- If the action was supposed to change the state on the screen, focus on the element that was supposed to be affected to assess success.
- If the action was a scroll or swipe and the UI elements did not change, the action likely failed.
- If the action was to type in the text field, the evaluation should be whether or not the text field has the exact text typed in.
- Don't tell the user what to do, just provide feedback on whether the action was successful or not.

Figure 8. System Prompt and step-by-step User Message for the Critic Agent used in in Reflexion based agent.

	UINavBench	AndroidWorld	OSWorld
No. of Apps	36	20	12
No. of Tasks	116	116	485
Platform	iOS	Android	Desktop
Eval	VLM+UIDetection	AXTree/Database	AXTree/Files
Depth			
Surface	22%	11%	20%
Shallow	45%	58%	49%
Complex	33%	31%	31%
Task Intent			
Assistance	52%	68%	30%
Pure Nav	12%	10%	1%
Search	35%	19%	10%
Data Entry	41%	45%	68%
Multi-App	25%	7%	25%
World Knowledge	13%	4%	15%

Table 8. Comparison of **UINavBench** with OSWorld and AndroidWorld. Our tasks are more complex, cover a wider range of apps, and cover more task intents. Compared to existing mobile-benchmarks, our tasks require both task-memory and world-knowledge for a larger fraction of tasks. Many of these metrics (e.g., navigation depth, determinism) require inspecting each step of a task, which is infeasible. We estimated all of these values as accurately as possible, but they should not be considered ground truth.

```

class ActionEnum(str, Enum):
    tap_on_element = 'tap_on_element'
    type_text_in_element = 'type_text_in_element'
    scroll_screen = 'scroll_screen'
    terminate = 'terminate'
    navigate_home = 'navigate_home'
    wait = 'wait'

class DirectionEnum(str, Enum):
    UP = 'UP'
    DOWN = 'DOWN'
    LEFT = 'LEFT'
    RIGHT = 'RIGHT'

class ActionResponse(BaseModel):
    """
    The action to be taken. Don't tap on labels. If there is an icon present above the label, tap on the
    icon instead.
    """
    action: ActionEnum = Field(
        """
        description='The action to be taken. Choose one of the available actions. tap_on_element if you
        want to tap on an element, type_text_in_element if you want to type text in an element, scroll_screen if
        you want to scroll the screen, terminate if you want to terminate the session, navigate_home if you want
        to navigate to the home screen, wait if you want to wait for 5 seconds before continuing.',
        """
    )
    element_id: Optional[int] = Field(
        None,
        description='The ID of the UI element to be interacted with. Never set the element ID to be None
        for tap_on_element or type_text_in_element actions.',
    )
    text: Optional[str] = Field(
        None, description='The text to be entered in the UI element. Only set the this when the action is
        type_text_in_element'
    )
    direction: Optional[DirectionEnum] = Field(
        None,
        description='The direction to scroll the screen. Can be one of [UP, DOWN, LEFT, RIGHT]. Only set
        this when the action is scroll_screen',
    )
    reason: Optional[str] = Field(
        None, description='Step-by-step thinking for the action to be taken.'
    )

```

Figure 9. Structured Output Prompt provided to the GPT based agent such that it produces valid actions

each other and can be evaluated independently. Additionally, unrelated tasks from different apps run on the same device but tasks for the same app run on different devices. For example, a Calendar task can run together with a Notes task, but multiple tasks involving the same app won't run on the same device to ensure reproducible environmental setup.

We host *mock mail servers* so that tasks involving sending/deleting to another recipient can be mocked. Mock servers are local to the device, and are created at the beginning of the task from a backup, similar to other staged data. Similarly, other staged data (notes, calendar events, reminders, messages, contacts) are also staged from a backup at the beginning of a task.

E. Qualitative Examples

In the next few pages, we show the trajectories of all three agents on the same task. In each figure, the first row is Reflexion + Text baseline. Row two is ReACT + Text baseline, and row three is ReACT baseline.

System Prompt for GPT-SOM + Text + Reflexion Agent

You are an AI assistant designed to help users navigate and interact with mobile applications.

At the beginning of each task, you will be provided with a description of the task.

Then, at each step, you will be provided with:

1. the screenshot of the current screen.
2. the history of actions taken on the environment and any feedback from a critic
3. the list of available tools that you can use.

Your task is to analyze the goal, current screen, history of actions and feedback about the last action, think step-by-step and provide the next action. The action will involve generating natural language descriptions of the element the user should interact with.

Here are some more guidelines:

1. If the task has been completed, you should call the terminate action.
2. Avoid getting stuck in trying to call the same action repeatedly. If you are stuck, try to find alternative ways to complete the task.
3. Try to accomplish the task with the least number of actions.
4. If the previous action failed, take corrective action by taking an alternative action.

User Message per step for GPT-SOM + Reflect Agent

Goal: {goal}

 Set of Marks Image

Text Representation of UI: {text representation of UI}

History: {action history with chain of thought}

Instructions: Based on the goal, current screen, history of actions, and feedback about the last action, think step-by-step and provide the next action.

Figure 10. System Prompt and step-by-step User Message for UGround + Reflexion Agent

```

class GroundingActionResponse(BaseModel):
    """
    The action to be taken. Don't interact with static labels. If there is an icon present above the
    label, interact with the icon instead.
    """
    action: ActionEnum = Field(
        ...,
        description='The action to be taken. Choose one of the available actions. \
tap_on_element if you want to tap on an element, type_text_in_element if you want to type text in an
element, scroll_screen if you want to scroll the screen, terminate if you want to terminate the session,
navigate_home if you want to navigate to the home screen, wait if you want to wait for 5 seconds before
continuing.',
    )
    element_description: List[str] = Field(
        ...,
        description='A descriptions containing visual and textual details as well as spatial location of
the UI element to be interacted with. Only set the element description if the action is tap_on_element or
type_text_in_element, otherwise set the element description to be an empty string. E.g.: circular design
with a pattern resembling a camera shutter or a mandala, in white on a light brown background. Select the
checkbox next to the label "Remember me".',
    )
    text: Optional[str] = Field(
        None, description='The text to be entered in the UI element. Set this only if you are calling
type_in_element action'
    )
    direction: Optional[DirectionEnum] = Field(
        None,
        description='The direction to scroll the screen. Can be one of [UP, DOWN, LEFT, RIGHT]. Only set
this when you are calling scroll_screen action.',
    )
    reason: Optional[str] = Field(
        None, description='Step-by-step thinking for the action to be taken.'
    )

```

Figure 11. Structured Output Prompt provided to the GPT for UGround + Reflexion Agent.

System Prompt for GPT-SOM + Text + Reflexion Agent

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.

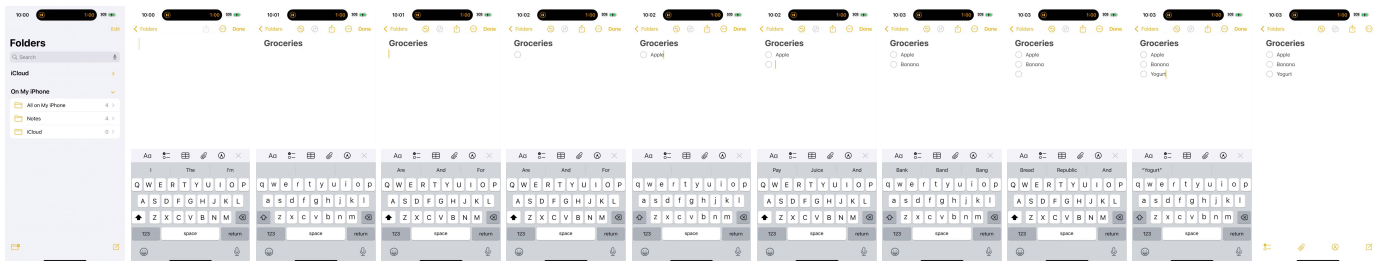
Output Format
 ``\nThought: ...
 Action: ...\n``

Action Space
 click(start_box='<|box_start|>(x1,y1)<|box_end|>')
 type(content='')
 scroll(start_box='<|box_start|>(x1,y1)<|box_end|>', end_box='<|box_start|>(x3,y3)<|box_end|>')
 press_home()
 finished(content='') # Submit the task regardless of whether it succeeds or fails.

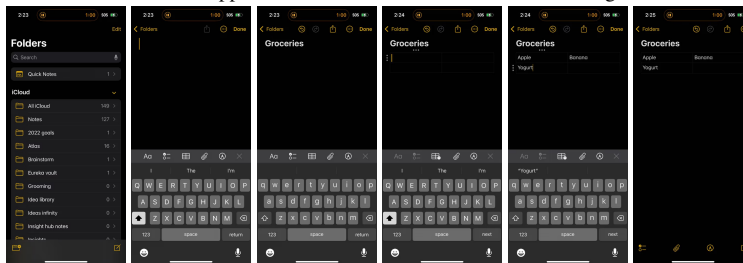
Note
 - Use English in `Thought` part.
 - Write a small plan and finally summarize your next action (with its target element) in one sentence in `Thought` part.
 - When the task is completed, call `finished()` action.

User Instruction

Figure 12. Prompt for the UI-Tars agent

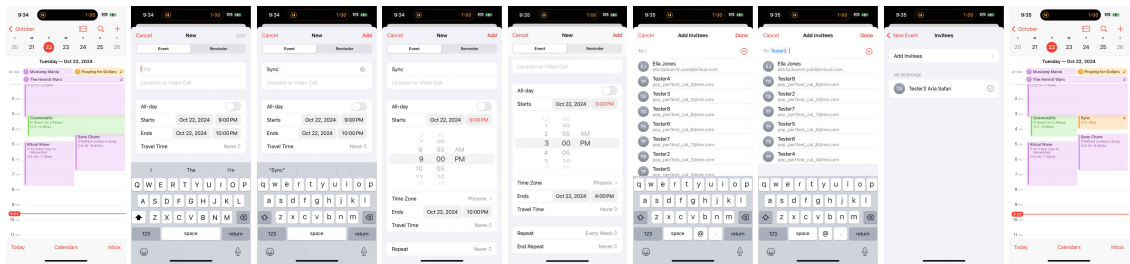


(a) Successful execution of the task for Notes app: "Create a new note 'Groceries', containing a bulleted list of apple, banana and yogurt".

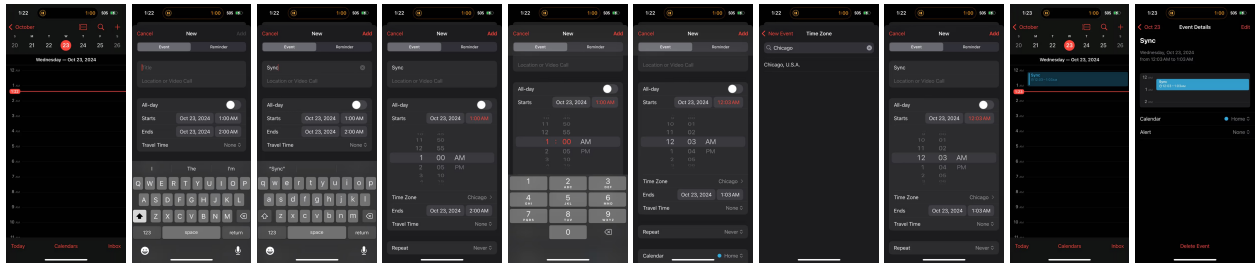


(b) Failure execution of the task for Notes app: "Create a new note 'Groceries', containing a bulleted list of apple, banana and yogurt". The content is presented in the form of a table, so it is considered a failure.

Figure 13. Execution outcomes for the Notes app task.



(a) Successful execution of the task for Calendar app: Create a recurring weekly event called 'Sync' starting today with Tester3 at 3:00 PM.



(b) Failure execution of the task for Calendar app: Create a recurring weekly event called 'Sync' starting today with Tester3 at 3:00 PM. This fails because it does not meet the task requirement of scheduling a weekly meeting with Tester3 at 3:00 PM.

Figure 14. Execution outcomes for the Calendar app task.

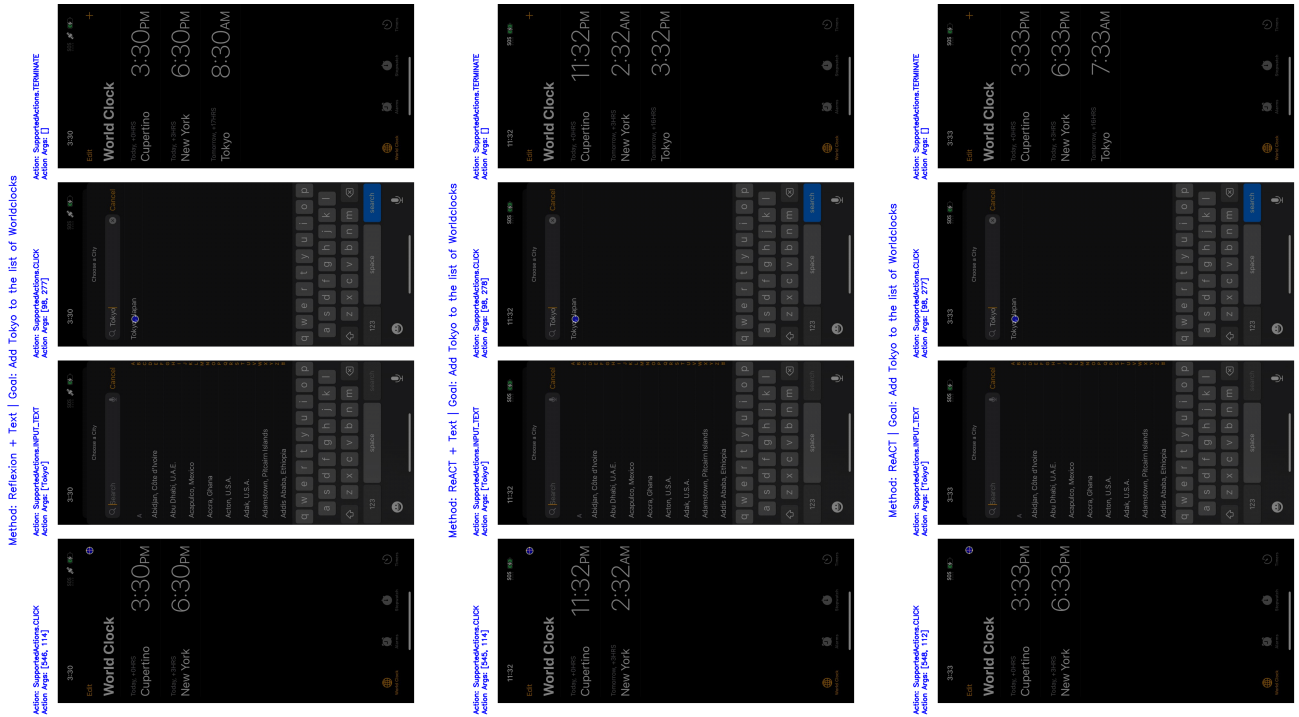
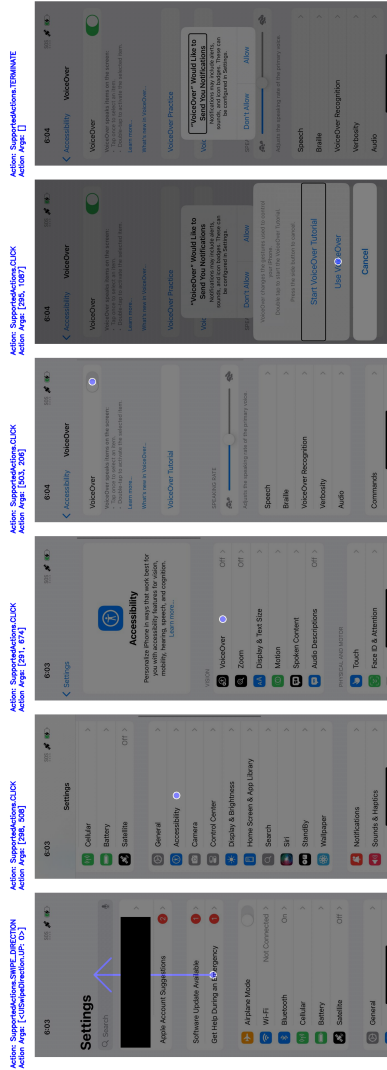
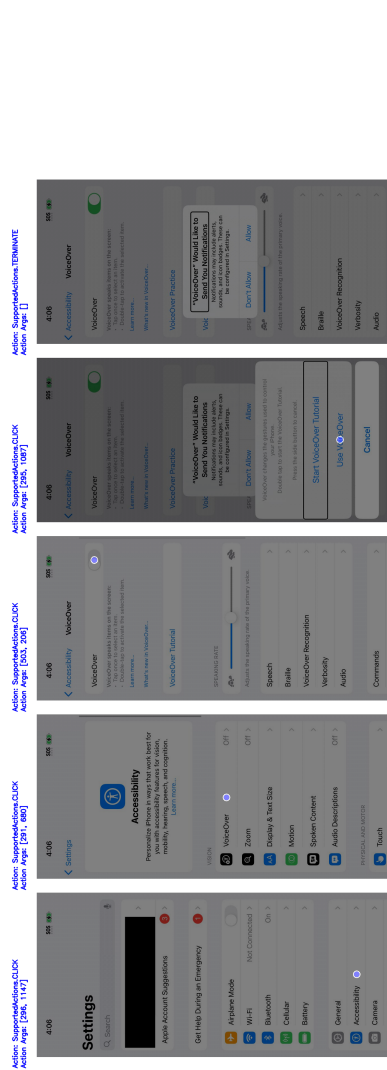


Figure 15. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, all agents successfully complete the task with the smallest number of steps possible.

Method: Reflexion + Text | Goal: Enable Voiceover in Settings



Method: ReACT + Text | Goal: Enable Voiceover in Settings



Method: ReACT | Goal: Enable Voiceover in Settings

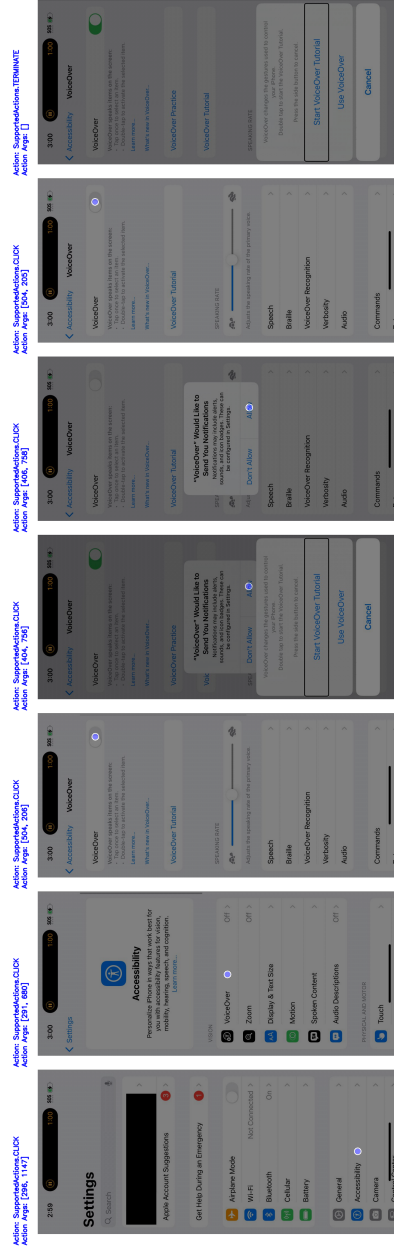


Figure 16. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. Here we show an example in which all the agents complete the task. The ReACT agent takes longer to finish the task because it tries to dismiss the popup first even though it is behind the three buttons at the bottom of the screen.

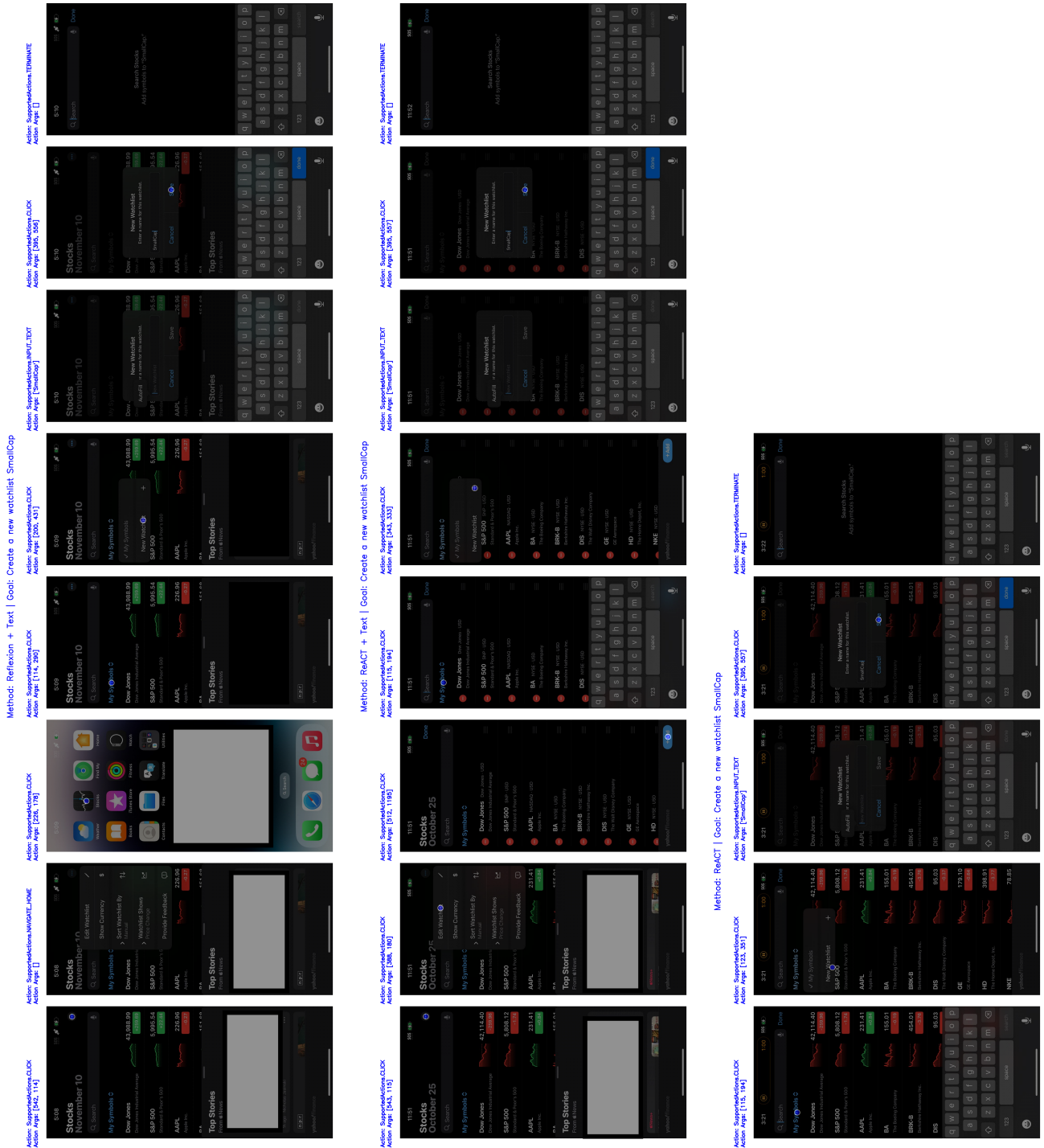


Figure 17. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. Here we show a counterexample in which having text-based representation hurts path efficiency even though all agents are successful. Text-based representation lacks spatial context, and therefore both Reflexion + Text and ReACT + Text agent taps the options icon, instead of tapping the watchlist dropdown element.

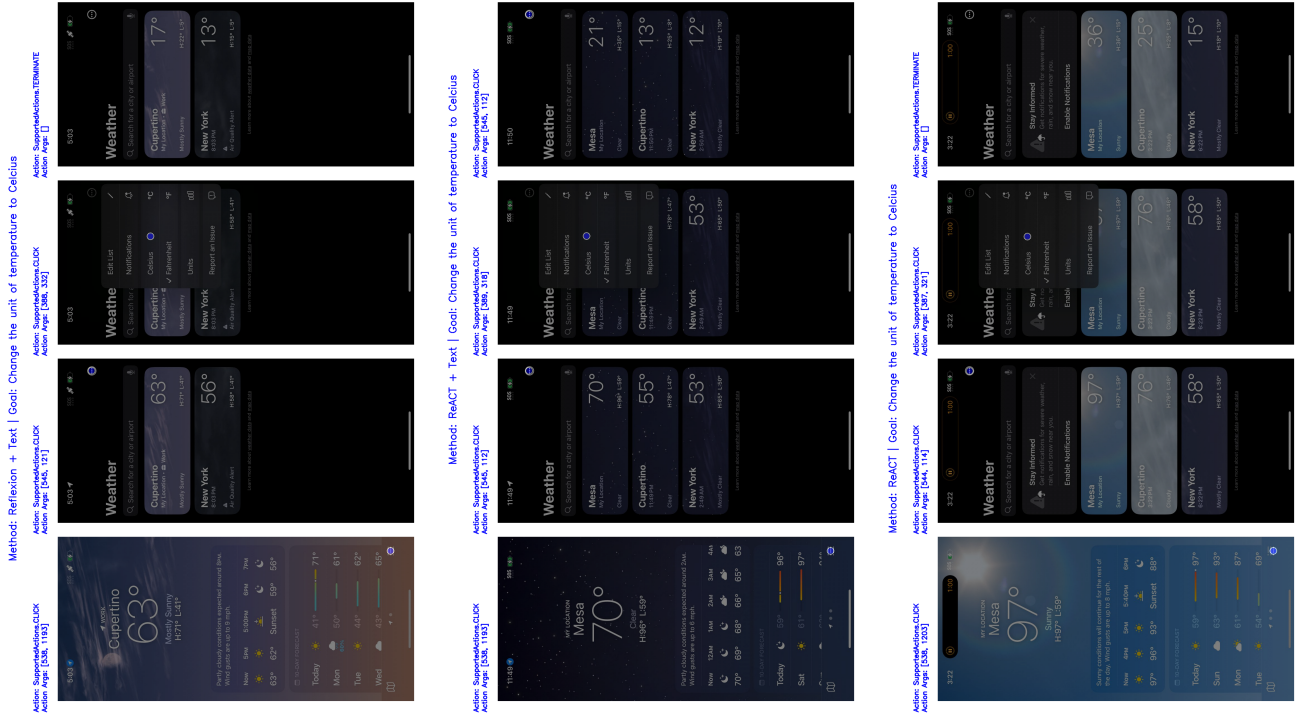


Figure 18. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, all agents successfully complete the task.

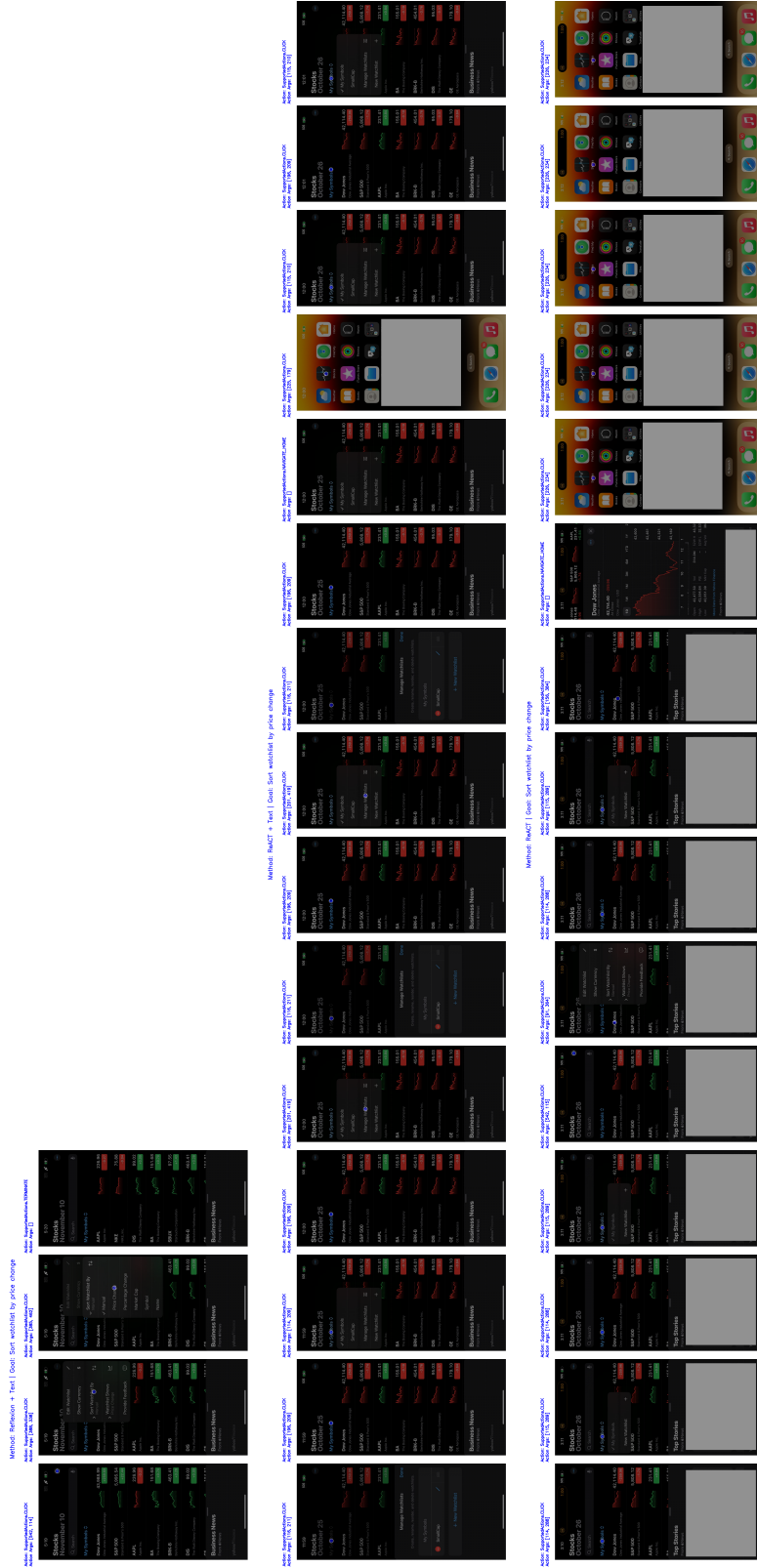


Figure 19. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, the goal is “Sort the watchlist by price change”. Reflexion agent (Row 1) correctly taps on the ‘options’ icon to complete the task in the shortest amount of time. However, both ReACT-based agents (Row 2 and 3) fail to finish the task. They commit to the wrong path at the very beginning of the trajectory and fail to recover from that mistake.

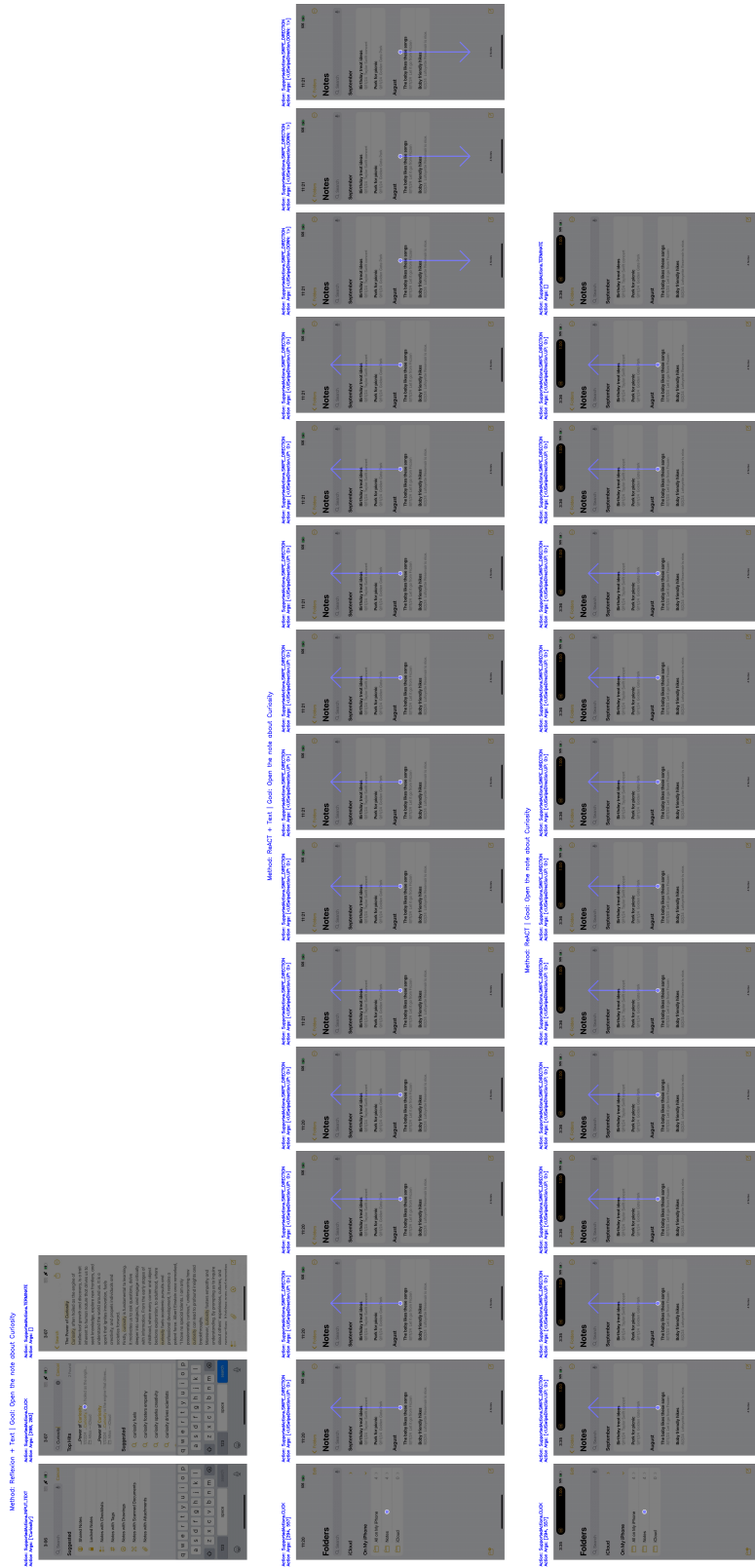
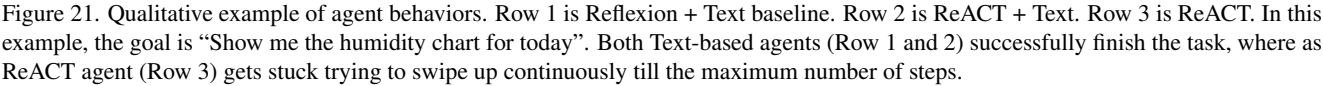


Figure 20. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, the goal is “Open the note about Curiosity”. Reflexion agent (Row 1) correctly uses the search bar to complete the task in the shortest amount of time. However, both ReACT-based agents (Row 2 and 3) fail to finish the task. They commit to the wrong path at the very beginning of the trajectory by trying to scroll down to find the note, and fail to recover from that mistake.



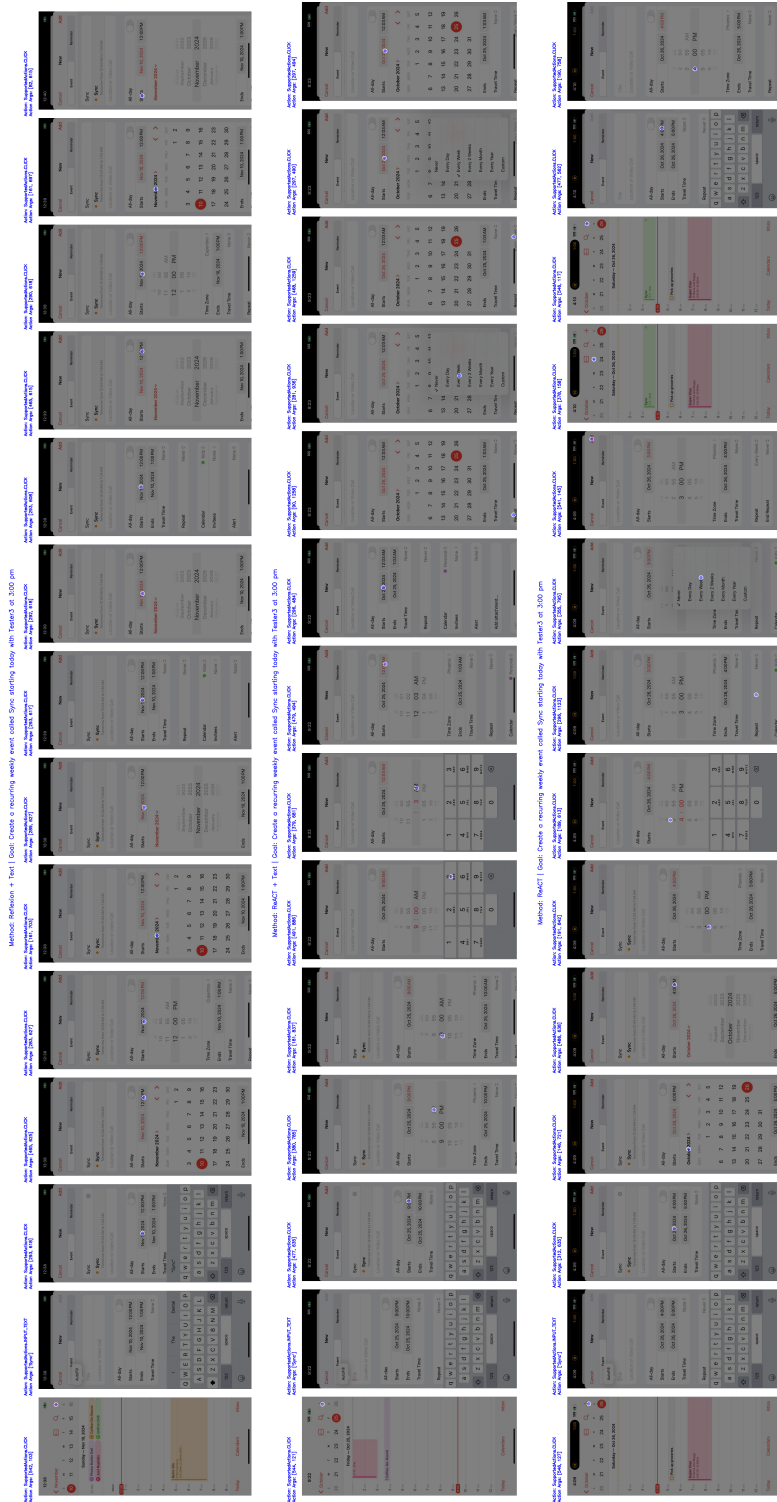


Figure 22. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, all agents fail to successfully complete the task. Calendar application is especially tricky for all our agents as it requires fine-grained control to interact with the picker-wheel element to set the time. We can observe in all rows that the agent struggles to correctly set the time and fails to execute the task.

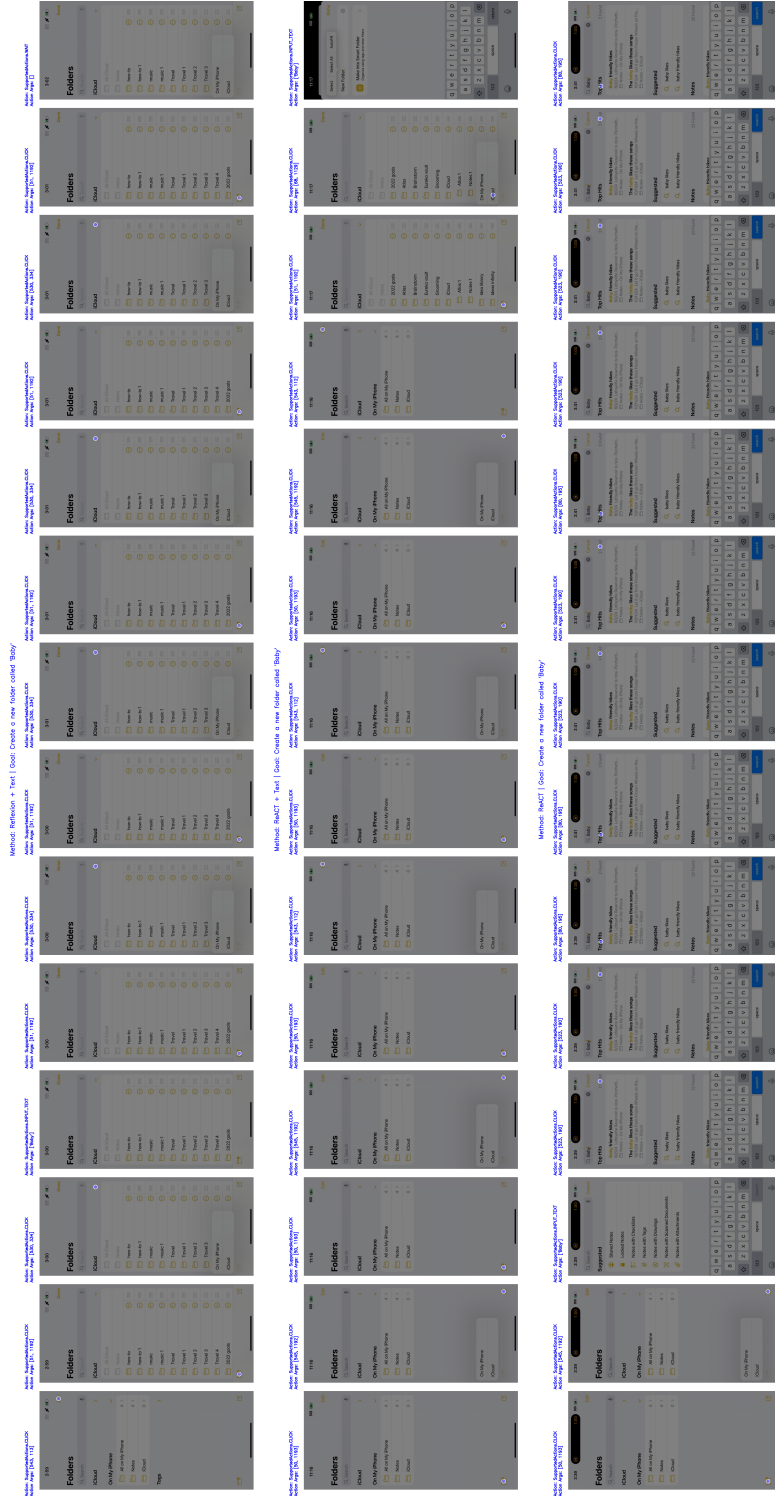


Figure 23. Qualitative example of agent behaviors. Row 1 is Reflexion + Text baseline. Row 2 is ReACT + Text. Row 3 is ReACT. In this example, all agents fail to successfully complete the task. This is related to perceptual difficulty and app knowledge. The agent is required to identify the new note icon "visually" as there is no supporting text next to the icon. Additionally, the agent has to rely on app knowledge to understand that on clicking the new note icon, a tooltip appears next to the icon asking for the location of the new folder. All agent fails to make the spatial connection, and taps on incorrect buttons instead.