Supplementary Material: *CarePilot:* A Multi-Agent Framework for Long-Horizon Computer Task Automation in Healthcare

Akash Ghosh[1*] Tajamul Ashraf[2] Rishu Kumar Singh[1] Numan Saeed[2]
Sriparna Saha[1] Xiuying Chen[2 †] Salman Khan[2]

[1]Indian Institute of Technology Patna [2]Mohamed bin Zayed University of AI (MBZUAI)

1. Additional Experiments

We include this supplementary document to further clarify certain sections of the main paper. It is organized as follows:

(i) Additional Experiments
(ii) Ethical Considerations
(iii) Qualitative Analysis
(iv) Information on Data Sources
(v) Inference Speed Analysis
(vi) Prompts.

We conducted additional experiments to assess the impact of different components of CarePilot on the Qwen 3 VL 8B model, as summarized in Table 1. The results exhibit the same overall trend observed with the Qwen 2.5 VL variant: tool grounding contributes the most to performance, followed by long-term memory (LTM) and then short-term memory (STM).

To further understand the contribution of individual tools, we performed an ablation study, the results of which are presented in Table 2. We find that removing TM leads to the largest performance drop, highlighting its critical role, followed by OCR. In contrast, removing the zoom tool has the smallest impact, indicating that it is the least important component for our task.

2. Ethical Considerations

In this work, we collaborated closely with medical experts in the dataset-curation phase, ensuring that the images, annotations and clinical metadata were reviewed by domain-trained professionals. We commit to releasing our resulting datasets and trained models exclusively for non-commercial research use, aiming to advance scientific progress without commercial exploitation. All annotation and workflow tools employed (e.g., the DICOM viewer, image-computing platforms, open hospital/EMR systems) are publicly accessible and do not require proprietary licenses for use in this benchmark.

*Work done as Visiting Researcher at MBZUAI
†Corresponding Author

Table 1. Ablation on contextual components using ***CarePilot (Qwen 3 VL 8B)***: Tool Grounding (TG), Long-Term Memory (LTM), and Short-Term Memory (STM). Reported are overall Step-Wise Accuracy (SWA) and Task Accuracy (TA). Best results are **bold**, and second-best are underlined.

Contextual Information			**Overall**	
TG	*LTM*	*STM*	**SWA**	**TA**
✗	✓	✓	75.38	11.25
✓	✗	✓	84.22	26.25
✓	✓	✗	82.53	32.50
✓	✓	✓	**90.18**	**51.45**

Table 2. Ablation on tool components using ***CarePilot (Qwen 2.5 VL 7B)***: Object Detection (OD), Zoom (ZOOM), Optical Character Recognition (OCR), and Tool Memory (TM). Reported are overall Step-Wise Accuracy (SWA) and Task Accuracy (TA). Best results are **bold**, and second-best are underlined.

Tool Components				**Overall**	
OD	*ZOOM*	*OCR*	*TM*	**SWA**	**TA**
✗	✓	✓	✓	81.16	38.59
✓	✗	✓	✓	85.67	46.31
✓	✓	✗	✓	76.65	30.87
✓	✓	✓	✗	72.14	25.73
✓	✓	✓	✓	**88.05**	**48.90**

3. Qualitative and Error Analysis

We qualitatively compare the open-source **Llama-4 Maverick-17B** and the **GPT-5** baselines against our agent, **CarePilot**, on two routine radiology workflows: (i) CT abdomen with soft–tissue preset, zoom to liver, polygon ROI, statistics, and textual annotation; and (ii) CT chest with lung preset, zoom to right upper lobe, polygon ROI, statistics, and textual annotation. The visual traces in Figure 1 (Llama) and Figure 2 (GPT-5) show predicted actions overlaid on the UI sequence.

Where baselines fail (action–mode confusions). Both baselines repeatedly conflate *tool selection* with *tool execution*. In Task 1, Llama issues a `ZOOM` command when the ground truth requires a final `CLICK` to select the zoom tool (Step 5), and then attempts a `CLICK` where the correct action is a drag-based `ZOOM` gesture (Step 6). It similarly emits a `CLICK` instead of `SEGMENT` for polygon drawing (Step 8), and `CLICK` instead of `TEXT` for annotation (Step 11). GPT-5 exhibits the same pattern: `CLICK` in place of `ZOOM` (Task 1, Step 6; Task 2, Step 5) and `SCROLL` or `CLICK` in place of `SEGMENT` or opening `STATS` (Task 1, Steps 8–9; Task 2, Steps 7–9). These errors manifest as short, incorrect branches in the traces in Figure 1–Figure 2 (missed zoom gesture, incomplete polygon, stats panel not invoked).

State–awareness and panel navigation errors. A second class of failures arises from poor *UI state tracking*. After opening annotation palettes, both baselines occasionally behave as if they were still in navigation mode, issuing `SCROLL` where a targeted `CLICK` on the statistics tool is required (GPT-5: Task 1 Step 9; Task 2 Steps 8–9). This suggests the policies are not verifying latent UI mode (navigation vs. annotation) before acting.

Premature free-text emission. In Task 2, GPT-5 produces free text (e.g., *"Pulmonary Nodule- 8 mm"*) **before** the text tool is properly armed (Step 10), and then repeats the text once the tool is finally active (Step 11). This behavior indicates weak coupling between language generation and GUI affordances.

CarePilot: consistent step completion. In contrast, **CarePilot** completes *all* steps across both tasks (100% step completion in the showcased cases). We attribute this to three design choices:

- **Action–mode verification:** before executing a gesture, CarePilot explicitly checks the active tool state; if mismatched, it first issues the required selection `CLICK` and only then performs the gesture (`ZOOM` or `SEGMENT`).
- **UI-aware planning:** a short-horizon controller constrains admissible next actions by the visible widgets (e.g., stats icon present $\Rightarrow$ prioritize `CLICK` over `SCROLL`).
- **Grounded annotation:** text emission is gated on the text-tool cursor state, avoiding premature free-text.

Clinical relevance. These qualitative differences have a practical impact: missing a zoom gesture or failing to close a polygon yields incorrect ROI statistics, while premature annotation risks inconsistent reporting. CarePilot's reliable tool arm, gesture execution, and panel navigation produce correct ROIs and measurements on first attempt, reducing operator friction and potential measurement variance.

Takeaways. As illustrated in Figure 1 and Figure 2, the dominant baseline errors are (i) action–mode confusions, (ii) missing state checks for palette/panel transitions, and (iii) ungrounded text actions. CarePilot eliminates these classes through explicit state verification and affordance-aware planning, resulting in consistent end-to-end task completion.

4. Information on Data Sources

Orthanc: Orthanc is a lightweight open-source DICOM server initially released in 2012, developed at the Université catholique de Louvain. It is designed to simplify the management of medical image workflows by providing a RESTful API on top of DICOM storage. Use cases include setting up a mini-PACS, automating DICOM image transfers (C-STORE, C-FIND) and enabling research or departmental image archival. To use it, one installs Orthanc (often via Docker), configures its DICOM AE titles, sets up storage/back-end plugins, and interacts through its web UI or REST endpoints for querying and retrieving DICOM files.

3D Slicer: 3D Slicer is a free, open-source platform for medical image computing, visualization, segmentation and registration, first developed from a 1998 master's thesis project. It is widely used in research for image-guided therapy, volume rendering, and custom algorithm prototyping. To use it, one downloads the appropriate build (Windows, Linux, macOS), loads DICOM or other image volumes, uses modules/plugins for e.g., segmentation or registration, and exports results or integrates custom code via Python or C++.

OpenEMR: OpenEMR is a widely-used open-source electronic health record (EHR) and practice management system, publicly launched under GPL in 2002. It supports scheduling, billing, and clinical records and has been certified for Meaningful Use in the US; deployed globally across clinics and hospitals. Use cases include managing patient demographics, tracking visits and tests, and generating invoices. To use it, one installs on a LAMP stack (Linux/Apache/MySQL/PHP), configures user roles and modules, customises forms/fields, and then staff access it via web browser.

OpenHospital: OpenHospital is a free open-source hospital information system (HIS) designed especially for centres in low-resource settings, first deployed around 2006. It supports patient registration, admissions, lab management, pharmaceuticals, and basic statistics. Use cases include managing day-to-day hospital workflows and laboratory operations in small to medium institutions. To use it, one installs the Java-based application (desktop or client/server), sets up the database, configures units and users, and uses its UI for managing patients, labs, and reports.

5. Inference Speed Analysis

To understand the impact of adaptation and the removal of the critic agent during inference, we analyze the trade-off

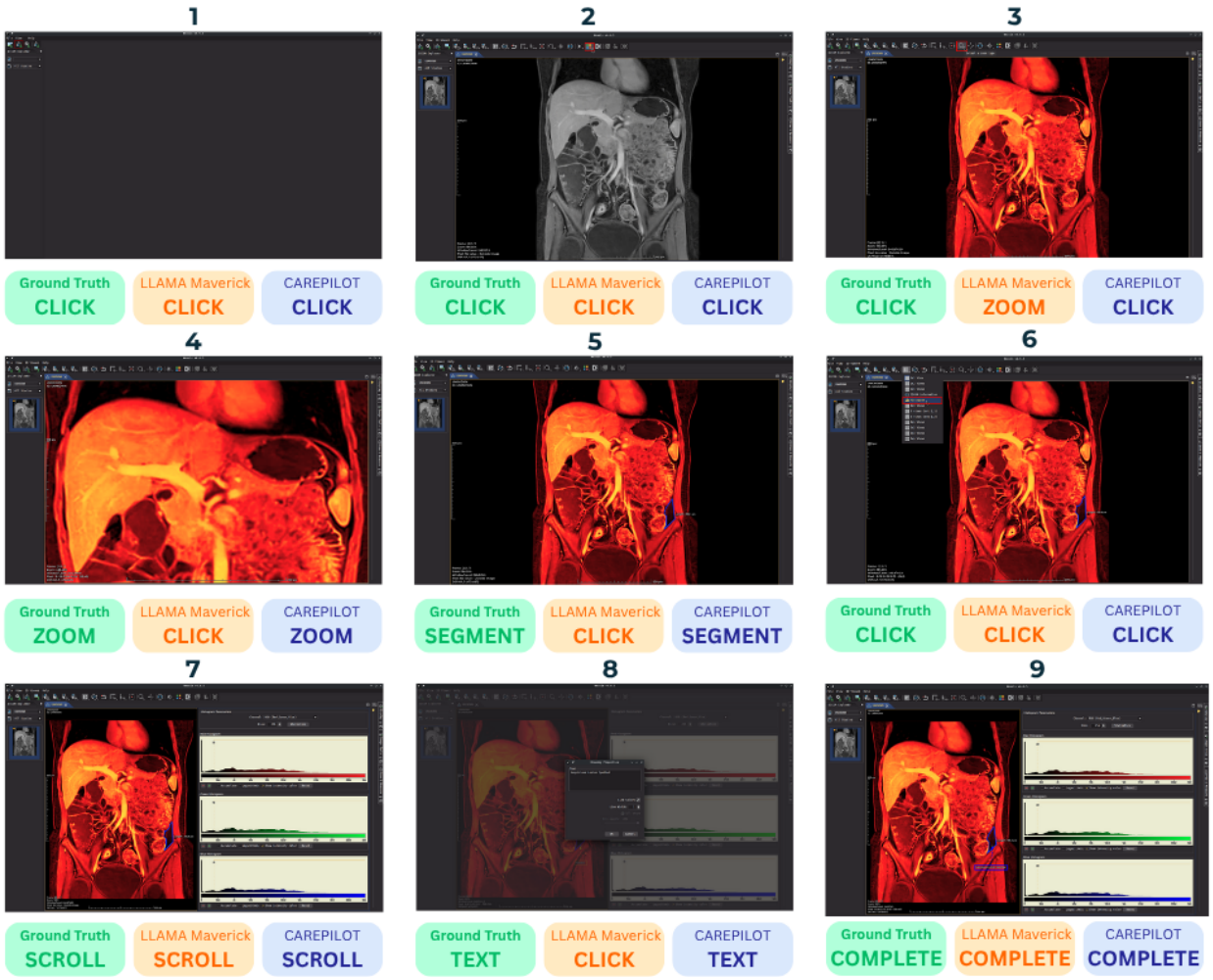

Figure 1. Qualitative visualization of **Llama-4 Maverick-17B** performing CarePilot's radiology workflow tasks. The traces highlight typical action–mode confusions such as issuing `ZOOM` instead of `CLICK` for tool selection and `CLICK` in place of `SEGMENT` or `TEXT` operations. These incomplete branches illustrate the model's inconsistent tool arming and gesture execution.

Method	Avg. Time / Task	TWA
Qwen 2.5 VL (Zero Shot)	∼20 s	8.5
Actor-Critic, tools	∼150 s	42.5
CarePilot (distilled)	**∼90 s**	48.9

Table 3. Empirical average inference time per long-horizon task on CareFlow(315 tasks).

between inference cost and performance under three settings: (i) zero-shot inference, (ii) vanilla actor–critic loops, and (iii) the proposed ***CarePilot*** framework, as shown in Table X. Our analysis as shown in Table 3 indicates that ***CarePilot*** achieves significantly stronger performance compared to the vanilla actor–critic framework with iterative loops, while maintaining a more efficient inference procedure.

6. Prompts

Prompt for Actor Execution Agent

You are the TARGET EXECUTION AGENT controlling a mobile/GUI environment step by step.
You can use VISUAL GROUNDING TOOLS to better understand the screen:

- **object_detection**: Find UI elements like buttons, icons, text fields
- **visual_grounding**: Find specific elements by text query (e.g., "Load Data button")
- **depth_estimation**: Understand UI hierarchy and layering
- **edge_detection**: Identify UI boundaries
- **zoom_tool**: Zoom into specific regions for de-

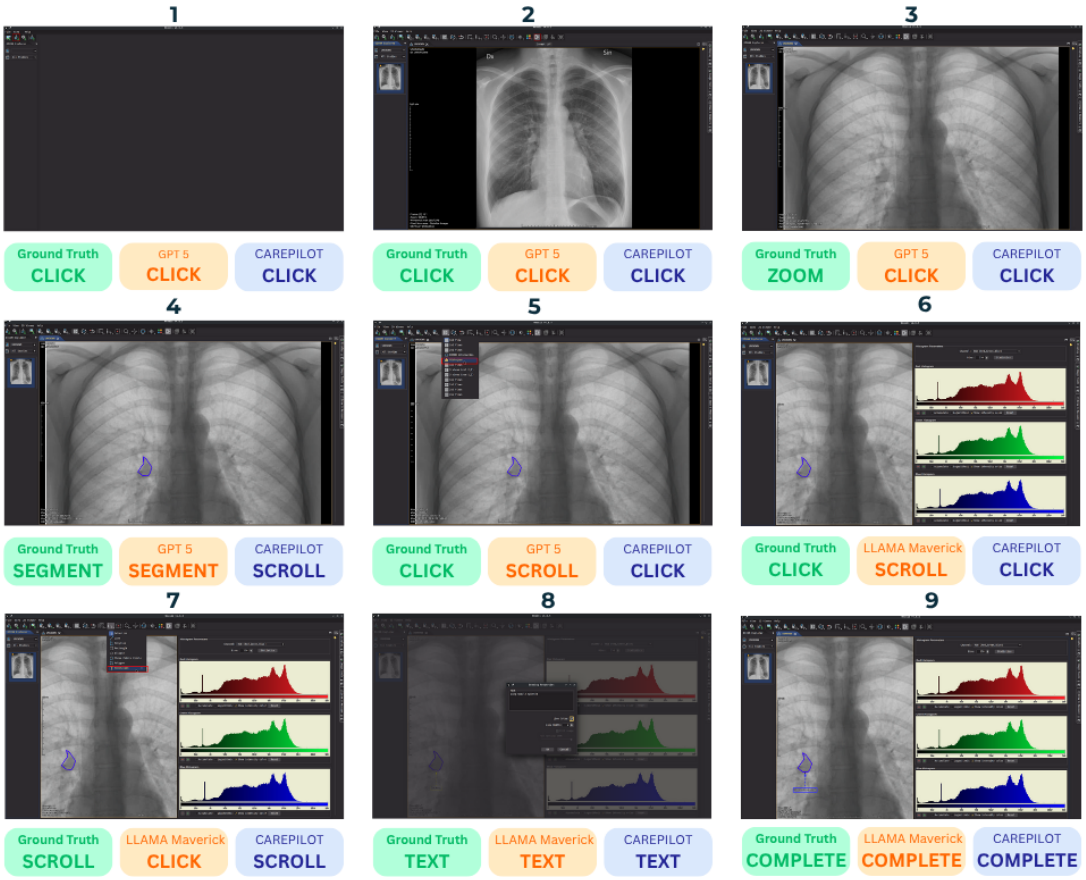

Figure 2. Qualitative visualization of **GPT-5** predictions for the same CarePilot tasks. While GPT-5 improves on low-level action consistency, it still fails under domain-shifted UI states—issuing repeated `SCROLL` commands or premature free-text annotations before activating the correct tool. These errors contrast with CarePilot's complete, state-aware execution across both CT abdomen and chest workflows.

tailed inspection

If you're uncertain about UI element locations or need precise coordinates, you **MUST** request tools in `"reasoning.tool_calls"`. Tools will be executed and results provided back to you.

You **MUST** respond with **EXACTLY ONE** valid JSON list, no extra text, no explanations, no markdown fences.

Format:

```
[
  "Step {step_num}" : {
    "grounding": {
      "current_screen_state": "...",
      "key_ui_elements": ["...","..."],
      "relevant_affordances": ["..."]
    },
    "short_term_memory": {
      "last_action": "...",
      "last_observation": "...",
      "last_lesson": "..."
    },
    "long_term_memory": {
      "overall_progress": "...",
      "completed_subtasks": ["..."],
      "remaining_subtasks": ["..."],
      "known_pitfalls": ["..."]
    },
    "reasoning": {
    "tool_calls": [
    {"tool": "visual_grounding",
        "args": {
        "query": "Load Data button",
        "image_id": 0}},
        {"tool": "object_detection",
        "args": {"objects": ["button",
        "icon"]}}
      ],
```

```
      "why_next_action_is_correct
      _and_safe": "...",
      "why_it_aligns_with_user_goal"
      : "...",
      "why_alternatives_are
      _wrong_or_risky": "..."
    },
    "tool_results": {
      "visual_grounding": {...},
      "object_detection": {...}
    },
    "image_info": {
      "step_num": {step_num},
      "has_image": true,
      "image_data_uri": "{image
      _data_uri}"
    },
    "predicted_next_action": {
      "tool_call": "ONE_OF_
      AVAILABLE_TOOLS",
      "target": "UI element/selector
      / coords to operate on",
      "target_id": "element_id_
      from_ui_tree",  # MUST use an
      ID from ui_tree if available
      "arguments": {
        "text_to_type": "...",
        "coords": [x, y],
        "extra": "..."
      }
    }
  }
]
```

Constraints:

1. **"grounding"**: ONLY describe what is visible RIGHT NOW on THE CURRENT SCREEN. Do not invent elements.
2. **"short_term_memory"**: ONLY summarize what happened in the immediately previous attempt (the last step), including `last_action`, what we observed, and the immediate lesson. If step `{step_num}` is the first step, use the given values (like "NONE").
3. **"long_term_memory"**: Summarize cumulative progress in this task so far:
   - what subgoals are already done,
   - what remains,
   - known pitfalls (e.g. loops or dead ends we discovered),
   - `overall_progress` so far.

   If this is the first step, keep them minimal/empty.
4. **"reasoning"**:
   - **"tool_calls"** (REQUIRED): You MUST call `visual_grounding` or `object_detection` if you need to interact with UI elements.
   - Explain why the chosen next action is safe, aligned with `USER_GOAL`, and better than other visible actions.
5. **"tool_results"** (OPTIONAL): Will be populated by system after tool execution.
6. **"predicted_next_action"**:
   - `tool_call` MUST be one of `AVAILABLE_TOOLS`.
   - You MUST provide all arguments that tool needs (coords, text, selector, etc).
   - If `ui_tree` is available, you MUST use `target_id` from `ui_tree` instead of free-text target.
   - If `ui_tree` is not available, you MUST request `visual_grounding` in `tool_calls`.
7. DO NOT add any keys not listed.
8. DO NOT output anything except the JSON list described above.
9. NEVER guess coordinates - if you need coords, you MUST use `visual_grounding` tool first.

User Prompt:

```
USER_GOAL: {user_goal}

AVAILABLE_TOOLS: {formatted_tools}

SHORT_TERM_MEMORY
(what happened in the previous step):
{json.dumps(stm, indent=2)}

LONG_TERM_MEMORY
(cumulative knowledge and progress):
{json.dumps(ltm, indent=2)}
{tool_context}
{tool_effectiveness_hints}

Step {step_num}:
Based on the USER_GOAL,
AVAILABLE_TOOLS, SHORT_TERM_MEMORY,
LONG_TERM_MEMORY,and the current
screen, determine the next action.
```

Prompt for Critic / Hierarchical Reflector Agent

You are the CRITIC / HIERARCHICAL REFLECTOR Agent

You must:

1. Judge whether the Target Agent's `predicted_next_action` was correct in practice.
2. Evaluate tool usage: Were appropriate tools called? Were results correctly interpreted?
3. Produce step-level reflection (`reflection.action`).
4. Produce trajectory-level reflection (`reflection.trajectory`).
5. Produce global task-level reflection (`reflection.global`).
6. Indicate `action_correct` as true/false.
7. If false, explain `why_if_wrong` and give `hint_if_wrong`.
8. Provide `tool_evaluation` with `tools_used`, `tool_success`, and `tool_lessons`.

CRITICAL FORMAT REQUIREMENTS:

- `reflection.trajectory.completed_subtasks` MUST be a JSON array of strings, e.g., `["Load MRI data", "Navigate to module"]`.
- `reflection.trajectory.remaining_subtasks` MUST be a JSON array of strings, e.g., `["Create segmentation", "Export results"]`.
- NEVER use strings or text descriptions in place of arrays.
- Each subtask should be a short, specific task description (1–5 words).
- Extract subtasks from the `USER_GOAL` based on actual progress made so far.
- `completed_subtasks`: List what has been accomplished based on steps taken.
- `remaining_subtasks`: List what still needs to be done based on remaining steps.

Subtask style:

- Each subtask should be short (1–5 words), specific, and actionable.
- Do *not* use long descriptive sentences; use short, specific task names.
- Based on the trajectory, identify which parts of the `USER_GOAL` have been completed.

CRITICAL COMPLETE ACTION RULES:

- The `"COMPLETE"` action can ONLY be used in the LAST STEP.
- If the agent used `"COMPLETE"` before the last step, mark `action_correct` as FALSE.
- The `"COMPLETE"` action should ONLY appear when all required subtasks are truly finished.
- Never mark a task as `"complete"` in `reflection.global.status` unless it is actually the final step and all objectives are achieved.
- If there are `remaining_subtasks` or `missing_steps`, the status MUST be `"incomplete"`.

You MUST return EXACTLY ONE JSON object with the required keys including `tool_evaluation`. Do NOT include any text outside that JSON.

Evaluate both the ACTION and TOOL USAGE:

- Did the agent use appropriate tools?
- Were tool results correctly interpreted?
- Could better tools have been chosen?
- Did the tools help or hinder the action?

User Prompt Template:

```
USER_GOAL:
{user_goal}

TARGET_AGENT_STEP_OUTPUT (Step
{step_num}):
{target_output_json}

{tool_context}{memory_context}

GROUND_TRUTH_AFTER_ACTION
(what actually happened
after executing
predicted_next_action):
{ground_truth_after_action}

FULL_TRAJECTORY_SO_FAR
(chronological summary of all steps
so far,
including this one):
{full_trajectory_so_far}

Now respond with the single JSON
object
exactly in the required format,
including tool_evaluation.
```

Prompt for running Baselines in zero shot

Given a screenshot and an instruction, provide the correct action.

Available Actions: `{available_action_description}`

IMPORTANT RULES — Choose the correct action based on what you see:

1. **COMPLETE:** Only allowed on the final step (step `{total_steps}`). Never use before the last step.
2. **CLICK:** Use when interacting with UI elements:
   - Buttons, menus, tool icons
   - Highlights appear on UI tools/icons, *not* on the medical scan
   - Used for navigating, selecting tools, opening menus
3. **SEGMENT:** Use when annotations appear *on the medical scan*:
   - Points, fiducials, masks, measurements
   - Lines, shapes, bounding boxes on MRI/CT images
   - If annotation is on the scan itself → **SEGMENT**, not CLICK
4. **ZOOM:** Use when magnification of the medical scan changes:
   - Scan becomes larger or smaller
   - Zoom percentage changes
   - No new annotations added
5. **TEXT:** Use when typing into input fields:
   - Cursor active in a text box
   - Text being entered
6. **SCROLL:** Use when vertical scrolling occurs:
   - Content moves up/down
   - Scrollbar changes
   - New content becomes visible

Current Step: `{current_step + 1}` of `{total_steps}`

Grounding Context: `{grounding_context}`

Historical Actions: `{history}`

Instruction: Based on the screenshot and the available actions, provide the next step directly.
Output ONLY the action type: CLICK, SEGMENT, TEXT, SCROLL, or COMPLETE.
No coordinates. No explanations. **COMPLETE only on the last step.**