ProSoftArena: Benchmarking Hierarchical Capabilities of Multi-modal Agents in Professional Software Environments

Supplementary Material

1. ProSoftArena Platform: Additional Details

1.1. Executable Professional Software Environment

ProSoftArena is constructed upon a realistic and interactive computing framework, utilizing virtual machine (VM) technology to host a fully functional Windows 11 operating system managed via Docker containers. We selected Windows as the primary platform given its dominance in the professional software ecosystem, where complex, industry-standard tools often require deep system integration and are frequently exclusive to or most stable on this OS. The VM-based architecture ensures an isolated and secure execution space, preventing agents from causing irreversible damage to the host system. Crucially, the environment leverages snapshot functionality to enable efficient, deterministic resets to a pristine state before each task, thereby guaranteeing consistent initial conditions for every evaluation.

The environment is pre-configured with 13 core professional applications, including Adobe Illustrator, Photoshop, ImageJ, ChemDraw, R, Excel, VSCode, NVivo, ArcGIS, ANSYS, MultiSim, AutoCAD, and SolidWorks. These applications were rigorously selected to cover representative workflows across 6 major disciplines and 20 specialized subfields. To ensure a consistent and reproducible evaluation baseline, all software is fixed to specific versions. Detailed specifications, including software versions and release builds, are provided in Table 6.

Beyond evaluation, this environment also serves as a scalable training platform, enabling agents to learn and master complex professional software usage.

1.1.1. Observation Space

The observation space O denotes the subset of the computer environment state $s_t \in S$ that can be perceived by the agent at each time step. Our environment provides multimodal observations comprising screen representations and additional computer-state signals. We evaluate three types of screen representations: (i) pixel screenshot image that captures the current screen content and window layout; (ii) XML-format accessibility (a11y) tree with structured attributes (control names, bounding boxes, interactable states); and (iii) Set-of-Marks (SoM) overlay that annotates actionable elements with unique identifiers. Visual examples of these observation types are illustrated in Table 7. Beyond the screen, we supply indirect observations including clipboard contents and session metadata (titles of all open windows and the current foreground window).

1.1.2. Action Space

We implement a unified action space A that encompasses 13 core mouse and keyboard actions covering the full spectrum of human-computer interactions, including mouse movement, clicks, dragging, keystrokes, and hotkeys.

To support strategic task management, we add three control actions: WAIT (pause to accommodate interface latency), DONE (declare successful completion), and FAIL (indicate infeasibility/abort). Together, these actions enable agents to execute precise low-level manipulations while managing high-level task flow, providing comprehensive control over professional software environments. Complete action definitions are listed in Table 8.

1.2. Agent Execution Trajectory

Given a policy π, the agent predicts action $a_t \in A$ at each time step t based on the goal g and current observation $o_t \in O$. The professional application then executes a_t inside the VM and updates the state s_t to s_{t+1}. Subsequently, the agent obtains an updated observation o_{t+1} reflecting the new environment state s_{t+1}. The process repeats until the agent outputs a termination action (DONE or FAIL) or when the maximum number of steps is reached ($t > t_{max}$). Figure 6 visualizes representative execution trajectories of an agent completing tasks in ProSoftArena environment.

1.3. Automated Evaluation Framework

Initial Task Environment Setup. As shown in the red block of Figure 7, the initialization script ("config") first downloads the specific task file "blank.svg" from the cloud storage to the VM path "C:/Users/Docker/Downloads/". It then executes the launch command to open Adobe Illustrator and load this file. To standardize the initial context, the script executes a series of preprocessing steps: it activates the target window "blank.svg", uses Python scripts to simulate keyboard shortcuts: 'v' to switch to the Selection Tool and 'ctrl+alt+0' to fit the artboard to the window. This ensures agents start in a consistent, ready-to-work state.

Execution-based Evaluation. Upon task completion, the system enters the evaluation phase shown in the green block in Figure 7. It executes the postconfig sequence to activate the Illustrator window, inject a JavaScript command via mshta to force-save the current canvas content back to the local disk. The system then retrieves the modified file from the VM for verification. Finally, the execution-based evaluation function "check_ai_l1_001" (shown in the blue block in Figure 7) is called. This function parses the XML struc-

Table 6. List of Professional Software, Versions, Disciplines, and Usage Descriptions in ProSoftArena.

Software	Version	Discipline	Subfield	Usage
Adobe Photoshop	26.7	Art & Design	Art	Digital image editing and photo retouching.
Adobe Illustrator	29.5.1	Art & Design	Design	Vector graphic design and layout print.
Microsoft Excel	2024	Business, Science	Accounting, Management, Math	Financial modeling, business intelligence, and numerical analysis.
R (RGui)	4.5.0	Business, Science, Health & Medicine	Finance, Math, Public Health	Statistical computing, econometric modeling, and health policy modeling.
ImageJ	15.4	Science, Health & Medicine	Biology, Diagnostics & Laboratory Medicine	Microscopic/medical image analysis, cellular quantification, and diagnostics.
ChemDraw	20.0	Science, Health & Medicine	Chemistry, Pharmacy	Molecular structure design, chemical property analysis, and drug design.
ArcGIS	10.2	Science	Geography	Spatial analysis, cartography, and geographical studies.
ANSYS	2024 R1	Science, Tech & Engineering	Physics, Energy Power, Materials	Finite element analysis (structural mechanics, thermal dynamics) and multiphysics simulation.
Multisim	14.3	Science, Tech & Engineering	Physics, Electronics	Electronic circuit simulation and PCB design.
NVivo	20 v1.7.2	Humanities	Social Science & Sociology	Systematic coding, thematic organization, and theoretical analysis of qualitative data (e.g., interviews).
VSCode	1.99.3	Technology & Engineering	Computer Science	Integrated development environment (IDE) for programming and software development.
AutoCAD	2026	Technology & Engineering	Architecture & Engineering	Technical drafting and construction documentation.
SolidWorks	2025 SP1.2	Technology & Engineering	Mechanical Engineering	Parametric modeling and assembly design.

ture of the retrieved SVG file and check whether a triangle exists as the task instruction requests.

2. ProSoftArena Benchmark: Construction Details and Data Samples

2.1. Domain and Software Coverage

We systematically investigate 6 core disciplines spanning 20 subfields and select 13 representative professional software applications. This selection prioritizes software that is most widely used and fundamental to executing core workflows within their respective domains. Figure 8 presents representative task samples from each domain, displaying the task instruction, initial state, and key metadata such as difficulty and human execution baselines.

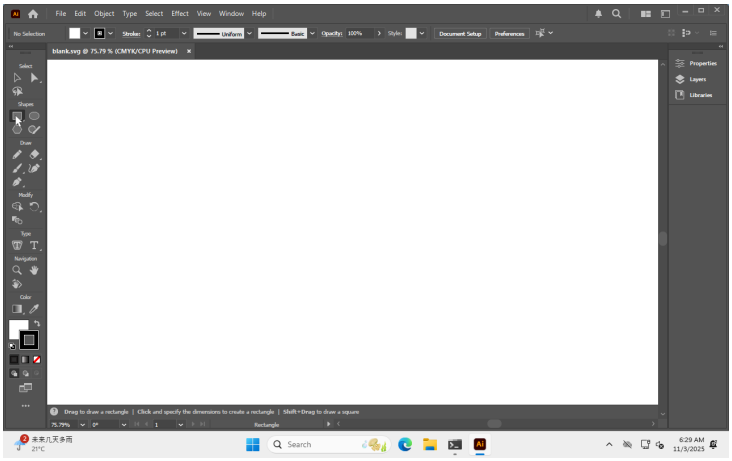
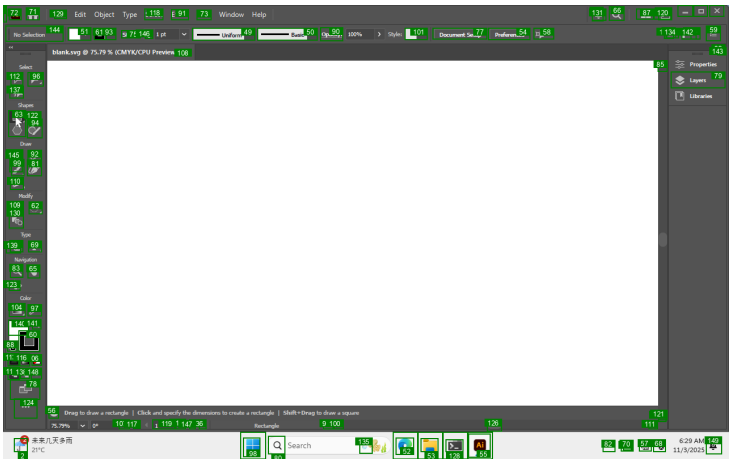
Art & Design. We consider two key subfields in this domain: artistic creation and visual design. For artistic creation, we include Adobe Photoshop, which professionals use for digital image editing and photo retouching. For visual design, we incorporate Adobe Illustrator, dedicated to vector graphic design and print layout. A sample task from this domain is: "Remove all the birds from the opened image in Photoshop, then adjust the brightness of the image to 20 and the vibrance to 100."

Business. We investigate three critical subfields: accounting, management, and finance. In accounting and management workflows, Microsoft Excel serves as the primary tool for financial modeling and business intelligence tasks. For advanced financial analysis, we include R, which supports statistical computing and econometric modeling. A representative task requires: "Add Slicers for 'Region' and 'Sales' to the data in range 'A1:B3' in the opened sheet to enable regional sales analysis."

Science. This domain covers five core subfields: biology, chemistry, geography, mathematics, and physics. For biology, we include ImageJ, which is widely used for microscopic image analysis and cellular quantification. In chemistry, we incorporate ChemDraw, dedicated to molecular structure design and chemical property analysis. Geographical studies are represented by ArcGIS, which serves as the standard platform for spatial analysis and cartography. Mathematical computing relies on Excel, employed for numerical analysis, and R, specialized in statistical modeling. For physics applications, we include ANSYS for multiphysics simulation and Multisim for electronic circuit design. Representative tasks include: "Measure the angles of four leaf veins from top to bottom in ImageJ and export the results to results.csv," and "Classify the service radius of parks based on park levels using spatial analysis tools in ArcGIS."

Health & Medicine. This domain encompasses the subfields of diagnostics & laboratory medicine, pharmacy, and public health. For the first, we include ImageJ, which supports medical image analysis and diagnostic quantification.

Table 7. Visual Examples of Different Observation Types Evaluated in ProSoftArena.

Observation Type	Visual Example
Screenshot	
Accessibility Tree (A11y)	2. Accessibility tree: tag name text class description position (top-left x&y) size (w&h) togglebutton Widgets 未来几天多雨 21°C "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (6, 752) (152, 48) textblock 未来几天多雨 "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (51, 759) (71, 16) textblock 21°C "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (51, 775) (23, 16) textblock 2 "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (36, 758) (5, 15) textblock 未来几天多雨 "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (51, 759) (71, 16) textblock 21°C "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (51, 775) (23, 16) textblock 2 "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (36, 758) (5, 15) togglebutton Start "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (418, 752) (45, 48) togglebutton Search "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (465, 760) (220, 32) textblock Search "" <class 'pywinauto.controls.uia_controls.StaticWrapper'> (502, 766) (42, 19) button Search - 世界水母日 "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (629, 760) (56, 32) systemtray-normalbutton Show Hidden Icons "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (1053, 752) (32, 48) systemtray-normalbutton OneDrive - Personal Not signed in "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (1085, 752) (32, 48) systemtray-accentbutton Network Network 46 Internet access "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (1117, 752) (28, 48) systemtray-showdesktopbutton Show Desktop "" <class 'pywinauto.controls.uia_controls.ButtonWrapper'> (1268, 752) (12, 48) progman Program Manager "" <class 'pywinauto.controls.uiawrapper.UIAWrapper'> (0, 0) (1280, 800) unknown Recycle Bin "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (0, 5) (74, 70) unknown Adobe Illustrator 2025 "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (0, 108) (74, 85) unknown ImageJ "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (0, 211) (74, 70) unknown Excel "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (0, 314) (74, 70) unknown ArcMap 10.8.1 "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (0, 417) (74, 85) unknown hello.txt "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (75, 57) (74, 70) unknown Adobe Photoshop 2025 "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (75, 108) (74, 85) unknown R 4.5.0 "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (75, 211) (74, 70) unknown ChemDraw 20.0 "" <class 'pywinauto.controls.uia_controls.ListItemWrapper'> (75, 314) (74, 85)
Set-of-Marks (SoM)	

In pharmacy, we incorporate ChemDraw, widely used for pharmaceutical molecule analysis and drug design. Public health research is represented by R, utilized for statistical analysis in epidemiological studies and health policy modeling. A sample task in this domain is "Analyze the molecular structure information of Acetylsalicylic acid (Aspirin) using ChemDraw."

Humanities & Social Science. This domain centers on qualitative research methodologies in sociology and related subfields. We incorporate NVivo, the premier software platform in this field, which enables systematic coding, thematic organization, and theoretical analysis of qualitative data such as interview transcripts and survey responses. A representative task requires: "Conduct Matrix Coding query in NVivo to compare perspectives of different respondents across various aspects of 'mental-emotional wellbeing.'"

Technology & Engineering. This domain spans six core subfields with their respective specialized software. For architecture and engineering, we include AutoCAD, employed for technical drafting and construction documenta-

Table 8. Definition of the Unified Action Space, encompassing 13 core primitive actions for low-level mouse and keyboard interaction and 3 high-level control actions for strategic task management.

Action	Function	Description
Mouse Movement	`move_id(id)`	Move cursor to a specific UI element via SoM ID.
	`move_abs(x, y)`	Move cursor to absolute coordinates (x, y).
Clicking	`single_click()`	Perform a single left click.
	`double_click()`	Perform a double left click.
	`right_click()`	Perform a right click (context menu).
Scrolling	`scroll(direction)`	Scroll the screen in the specified direction.
Keyboard	`write(text)`	Input a text string into the active field.
	`press(key)`	Press a specific key or hotkey combination.
Clipboard	`copy_text(text)`	Copy specified text to the system clipboard.
	`copy_image(image)`	Copy an image to the system clipboard.
	`paste()`	Paste content from the clipboard.
System Control	`open_program(prog)`	Launch a specific professional application.
	`switch_to_app(win)`	Switch focus to a target application window.
Task Control	`WAIT()`	Pause execution to accommodate interface latency.
	`DONE()`	Declare successful task completion.
	`FAIL()`	Abort task due to infeasibility or error.

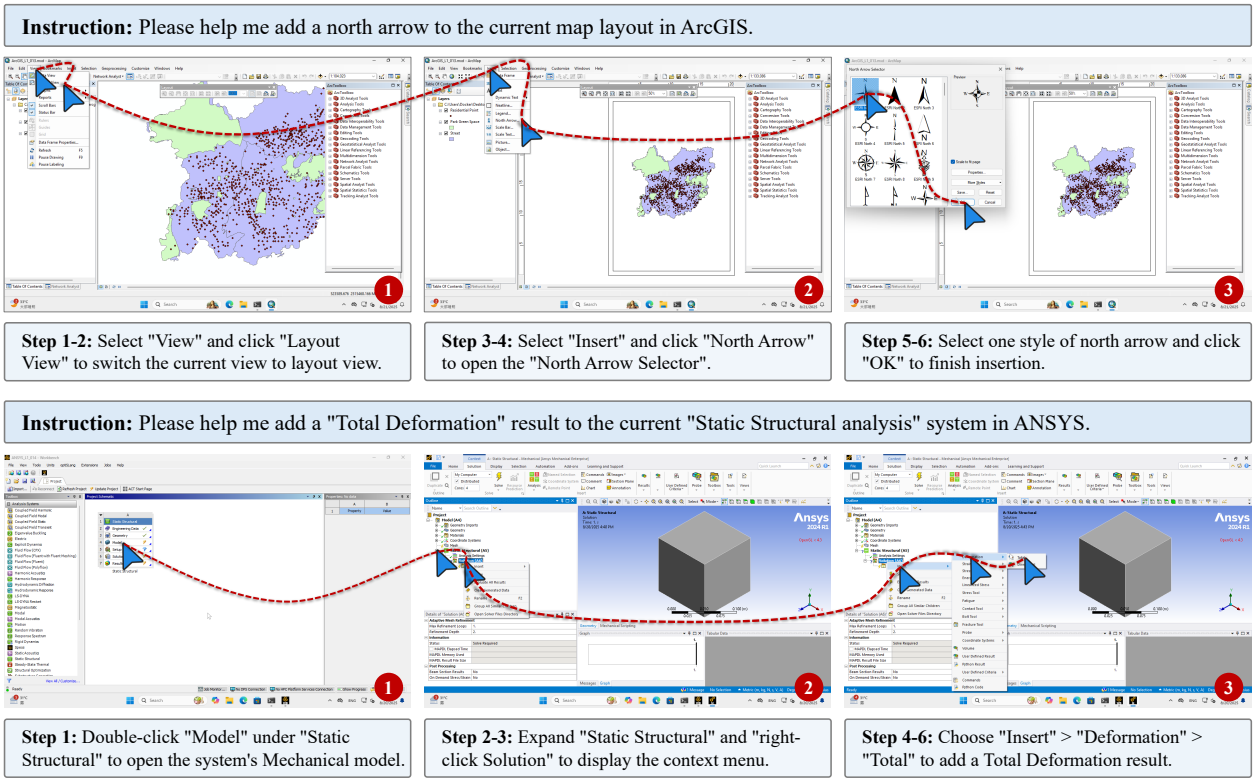


Figure 6. **Qualitative Examples of Agent Trajectories.** The top row shows the process of adding a north arrow in **ArcGIS**, involving view switching and dialog interaction. The bottom row illustrates adding a simulation result in **ANSYS** through tree navigation and context menus. Red dashed lines and blue cursors indicate mouse movements and clicks, respectively.

tion. Computer science utilizes VSCode, which provides the integrated environment for programming and software development. For electronics engineering, we incorporate Multisim, dedicated to circuit simulation and PCB design. For energy power and materials science, we include ANSYS, widely used for finite element analysis including

```
{
    "id": "AI_L1_001",
    "snapshot": "ai",
    "instruction": "Please help me draw a triangle in the currently opened file in Adobe Illustrator.",
    "source": "https://helpx.adobe.com/illustrator/user-guide.html",
    "config": [
      {"type": "download", "parameters": {"files": [{"url": "https://github.com/.../blank.svg", "path":
                                                     "C:/Users/Docker/Downloads\\blank.svg"}]}},
      {"type": "launch", "parameters": {"command": ["C:/Program Files/.../Illustrator.exe","C:/Users/Docker/Downloads/blank.svg"]}},
      {"type": "sleep", "parameters": {"seconds": 35}},
      {"type": "activate_window","parameters": {"window_name": "blank.svg"}},
      {"type": "execute" ,"parameters": {"command": ["python -c import pyautogui; pyautogui.press('v');"]}},
      {"type": "execute","parameters": {"command": ["python -c import pyautogui; pyautogui.hotkey('ctrl','alt','0');"]}}
    ],
    "evaluator": {
      "postconfig": [
        {"type": "activate_window","parameters": {"window_name": "blank.svg"}},
        {"type": "execute","parameters": {"command": ["mshta", "javascript:var ai=new ActiveXObject('Illustrator.Application');
                                                       ai.DoJavaScriptFile('C:\\\\save_ai.jsx');close()"]}},
        {"type": "sleep", "parameters": {"seconds": 2}},
      ],
      "func": "check_ai_l1_001",
      "result": {"type": "vm_file","path": "C:\\Users\\Docker\\Downloads\\blank.svg","dest": "AI_L1_001_output.svg"}
    }
  }
```

```python
def check_ai_l1_001(svg_path):
    tree = ET.parse(svg_path)
    root = tree.getroot()
    ns = {'svg': 'http://www.w3.org/2000/svg'}
    for polygon in root.findall('.//svg:polygon', ns):
        points = polygon.get('points', '')
        coords = [float(p) for p in re.findall(r'[-+]?\d*\.\d+|\d+', points)]
        if len(coords) == 6 or (len(coords) == 8 and
           abs(coords[0]-coords[-2]) < 0.1 and abs(coords[1]-coords[-1]) < 0.1):
            return 1.0
    return 0.0
```

Figure 7. Initialization and Evaluation Script Example for Task "AI_L1_001" (Adobe Illustrator).

structural mechanics and thermal dynamics. Mechanical engineering employs SolidWorks for parametric modeling and assembly design. Representative tasks include: "Draw a standard outdoor badminton court and annotate the dimensions in AutoCAD," and "Model three components of a syringe in SolidWorks: a barrel, a plunger, and a needle."

See Table 6 for detailed software specifications, versions, and domain mappings.

2.2. Metadata Annotation

All annotations are conducted within a standardized software environment with fixed application versions to ensure consistency in task execution. For each task, domain experts curate a comprehensive metadata profile comprising:

- **Task ID:** A unique identifier for each task.
- **Instruction:** A precise natural language directive specifying the user goal. Instructions are rigorously formulated to ensure the target output is unambiguous and objective.
- **Context Resources:** The necessary input assets to complete the task (e.g., files, datasets or images).
- **Task Source:** The task's provenance (e.g., manual section, tutorial URL, or expert-authored).
- **Core Operations:** For higher-level tasks (L2/L3), annotators explicitly map the workflow back to the atomic operations (L1) required (e.g., "Shape Drawing"), enabling fine-grained capability analysis.
- **Expected Output:** Defines the tangible artifacts (e.g., saved files, exported charts) or system states used for objective evaluation.

Table 9. Metadata Annotation Example.

Metadata Field	Content / Value
Task ID	`AI_L1_001`
Instruction	Please help me draw a triangle in the currently opened file in Adobe Illustrator.
Context Resources	`blank.svg` (Input file loaded at start)
Task Source	Official User Guide (`https://helpx.adobe.com/illustrator/user-guide.html`)
Core Operations	Shape Drawing / Polygon Tool Usage
Expected Output	`AI_L1_001_output.svg` (Must contain a valid 3-vertex polygon)
Difficulty	Easy
Human Reference	• *Demo:* `AI_L1_001.mp4` • *Steps:* 3 • *Time:* 10s

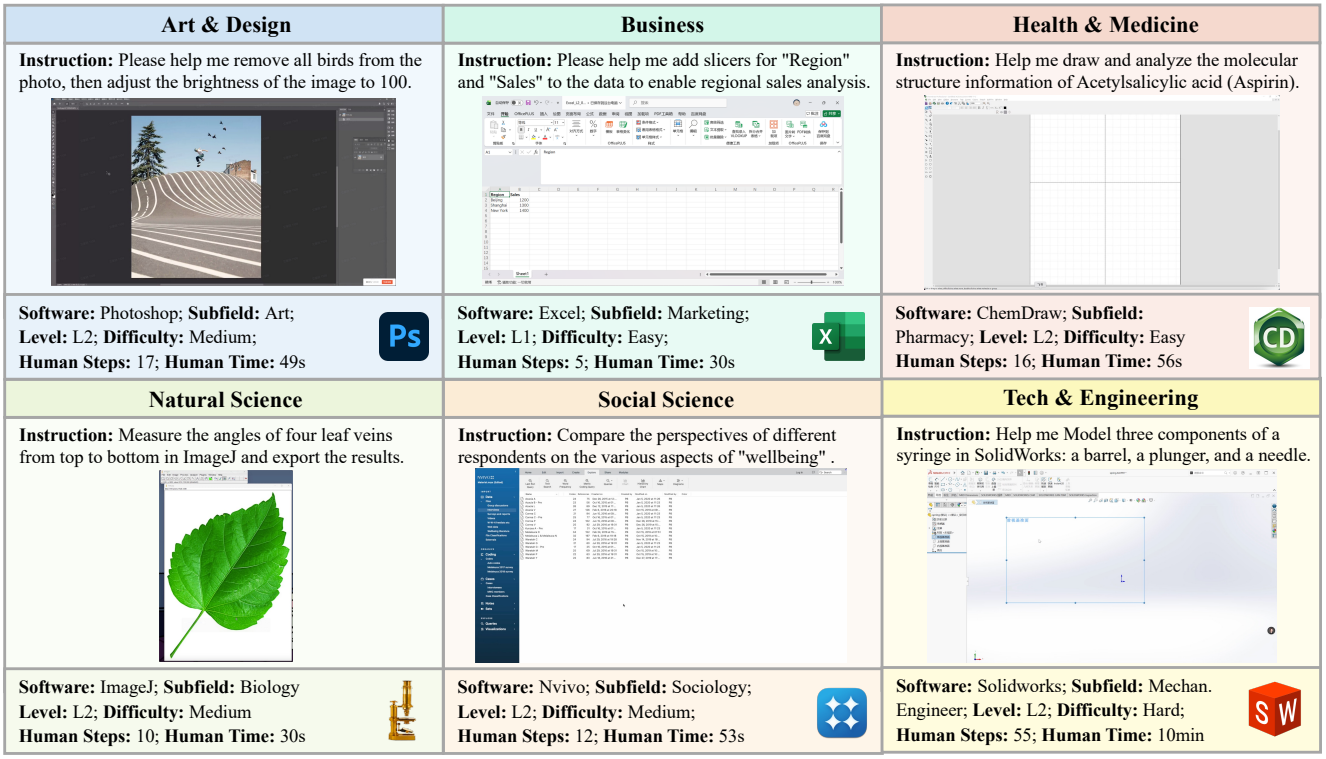


Figure 8. **Representative Task Samples across Six Core Domains in ProSoftArena.** For each domain, we showcase a typical task scenario including the specific natural language instruction, the initial computer state, and associated metadata.

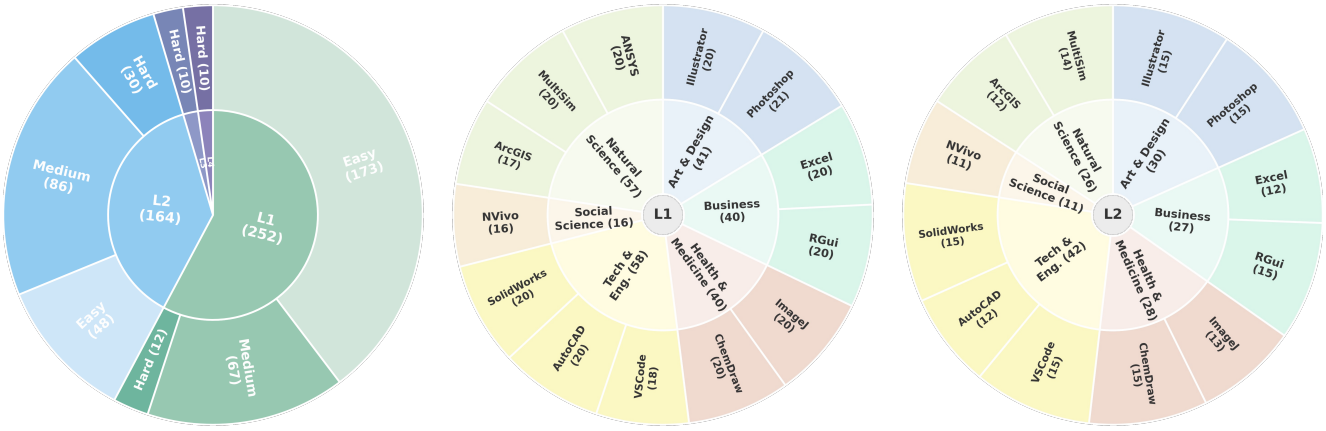


Figure 9. **Statistical Breakdown of ProSoftArena Benchmark. Left:** Task distribution across Capability Levels (L1-L4) and Difficulty ratings (Easy, Medium, Hard). **Middle & Right:** Detailed breakdowns of domain and software coverage for L1 (Operation Level) and L2 (Software Level) tasks, respectively.

- **Difficulty & Human Baseline:** Experts provide a subjective difficulty rating (Easy, Medium, Hard) and record their own execution metrics, including a standard demonstration video, total operation steps, and completion time, serving as a "Human Expert" baseline.

Table 9 presents a concrete example of these metadata fields for an Operation Level task in Adobe Illustrator.

2.3. Quality Control

Following the data annotation phase, we implement a rigorous quality control protocol to ensure the reliability and validity of the benchmark. Each task is subjected to independent cross-validation by at least two domain experts. The review process strictly assesses the following dimensions:

- **Feasibility:** Verifying that the task is technically com-

pletable within the standardized software environment.

- **Instructional Clarity:** Ensuring the natural language instruction is detailed, specific, and unambiguous, guaranteeing that it leads to a unique, valid solution rather than open-ended interpretations.
- **Output Alignment:** Confirming that the annotated expected output (e.g., specific files or system states) accurately reflects the fulfillment of the task objective.
- **Evaluability:** Verifying that the task is objectively evaluable using our execution-based scoring criteria.

Tasks that failed to meet these standards—particularly those with ambiguous instructions, ill-defined outputs, or unreliable evaluation metrics—were explicitly excluded from the final benchmark.

2.4. Statistics Breakdowns

Figure 9 visualizes the benchmark composition. The left chart displays the distribution of tasks across capability levels and difficulty ratings. The middle and right charts break down the specific domain and software coverage for L1 and L2 tasks, respectively. Additionally, Table 10 details a representative Pipeline Level (L3) workflow to illustrate cross-application complexity.

3. Experiments Setup and Extended Results

3.1. Subjective Evaluation for Creative Tasks (L4)

Unlike the deterministic tasks in L1-L3, Creative Level (L4) tasks focus on open-ended generation where binary success metrics are insufficient. Figure 10 presents a subjective comparison between the outputs generated by state-of-the-art multimodal agents and those created by human experts.

Results Analysis. The comparison reveals a profound capability gap in open-ended creative workflows. While human experts utilize these tools to produce polished and structurally sound designs, current agents struggle significantly to generate coherent results. As observed, the agent-generated outputs are often structurally incomplete or even shapeless, failing to meet basic design standards. This critical limitation stems from two primary factors:

- **GUI Interaction Bottlenecks:** Creative tasks require continuous, fine-grained mouse manipulations (e.g., dragging to draw curves, precise positioning of elements). Current agents lack the high-frequency, pixel-perfect motor control required for such ”analog” interactions.
- **Abstract Intent Grounding:** Agents struggle to translate abstract creative intentions (e.g., ”design a warm and inviting logo”) into the concrete, multi-step execution plans required by professional software, resulting in outputs that are functionally disjointed from the user's prompt.

3.2. Agent Configurations and System Prompts

For MLLM-based agents, we adopt a standardized configuration with a temperature of 1.0 and top-p of 0.8 to balance generation diversity with instruction adherence. To accommodate varying task complexities while preventing infinite execution loops, we dynamically adjust the maximum step budget (MAX _STEPS): 35 steps for Operation Level (L1), 55 steps for Software and Creative Levels (L2, L4), and 100 steps for the long-horizon Pipeline Level (L3) workflows. The agent's observation space integrates raw screenshots (with optional Set-of-Marks overlay) and the accessibility trees. Table 11 presents a complete system prompt used for the observation setting ”SoM+Screenshot+A11y”, defining the agent's role, operational constraints, and success criteria, while outlining the available observation and action spaces alongside representative examples.

3.3. Failure Case Studies

We visualize representative examples of common error patterns in Figures 11-14, including (i) **Domain & Tool Knowledge Gaps** (Figure 11) where agents lack specific tool or domain expertise; (ii) **Task Planning Errors & Invalid Actions** (Figure 12) where agents omits crucial procedures or predicts actions outside the predefined action space; (iii) **Visual Grounding Inaccuracies** (Figure 13) where agents misidentify UI elements despite generating correct action plans; and (iv) **Repeating Operations** (Figure 14).

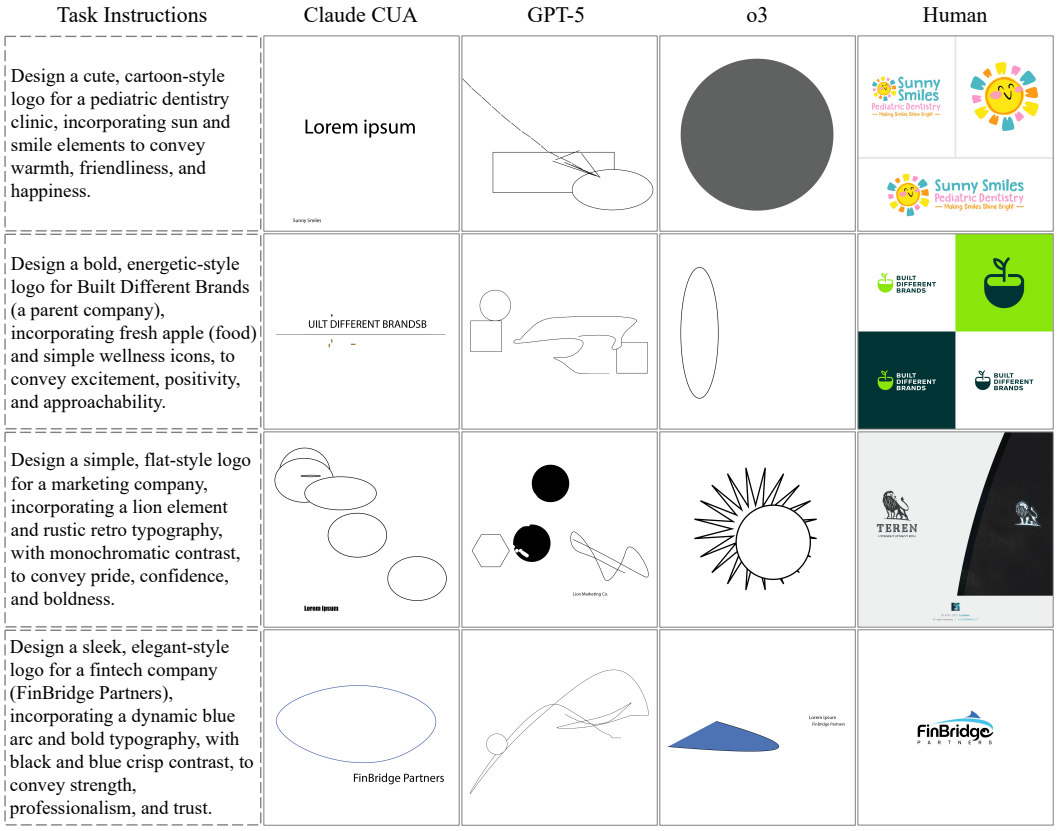
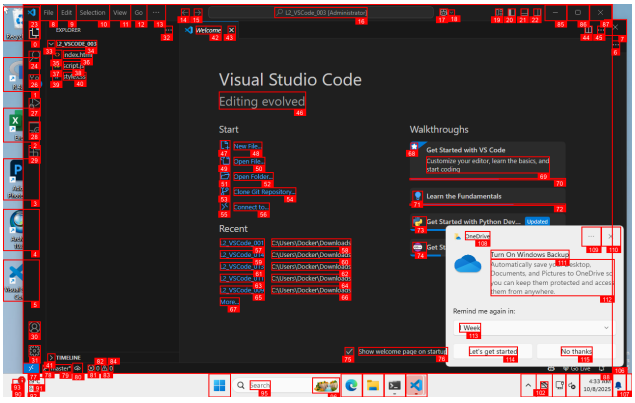


Figure 10. Subjective Evaluation Results on Creative Level (L4) Tasks.

Task: Please help me set up a complete web development environment for the currently opened project "L2_VSCode_003" in VSCode. **Model:** Qwen2.5-VL-72B-Instruct

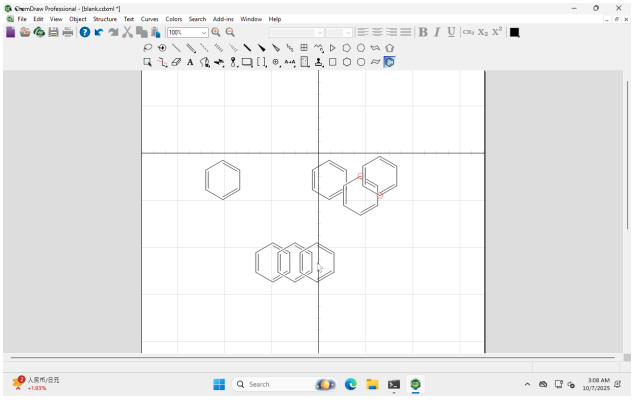
Step: # Move the mouse to the "View" menu and click to open the dropdown menu…
# Move the mouse to the "Extensions" option in the dropdown menu…
# Use the search bar in the Extensions view to search for "Live Server"…
computer.keyboard.write("Live Server") computer.keyboard.press("enter")



(a) **Tool Usage Gap (VSCode)**

Task: Please help me draw an anthracene molecule in ChemDraw.
Model: Qwen2.5-VL-72B-Instruct

Step: ```decision
Anthracene is already drawn on the canvas; task is complete.
DONE

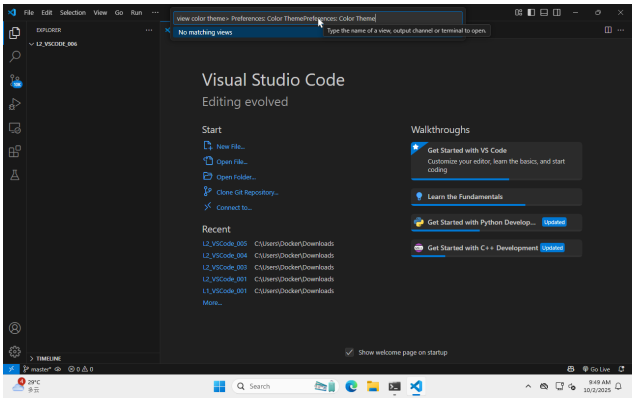


(b) **Domain Knowledge Gap (ChemDraw)**

Figure 11. **Domain and Tool Knowledge Gaps.** (a) The agent fails to install and launch the "Live Server" extension in VSCode due to a lack of specific tool knowledge. (b) The agent incorrectly constructs the anthracene molecule but hallucinates success, demonstrating a failure to verify the chemical structure based on domain principles.

Task: Help me use the Command Palette in VSCode to change the color theme to “Red”.
Model: Qwen2.5-VL-72B-Instruct

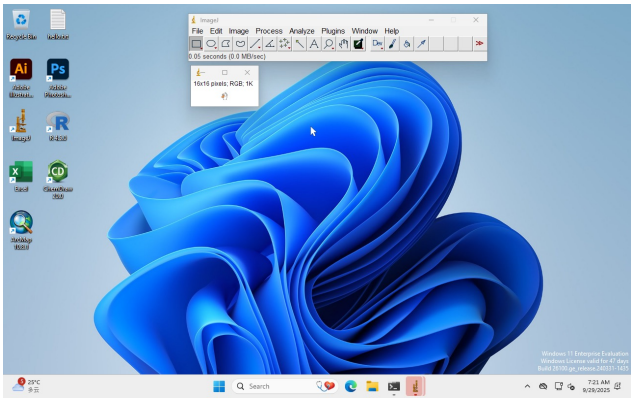
Step: # Focus the Command Palette input and run the Color Theme command
computer.mouse.move_id(id=78) computer.mouse.single_click()
computer.keyboard.write("Preferences: Color Theme")



(a) **Task Planning Error**

Task: Please help me rotate the current image 90 degrees clockwise in ImageJ.
Model: GLM-4.5V

Step: For the current step, I need to click on the "Image" menu in the ImageJ window. The final answer is
<|begin_of_box|>CLICK(box='[194, 66]')<|end_of_box|>



(b) **Invalid Action (GLM-4.5V)**

Figure 12. **Task Planning Error & Predicting Invalid Actions.** (a) The agent fails to clear the previously entered command before inputting the next one due to task planning error. (b) The agent predicts an action outside the predefined action space, indicating a limitation in instruction following capabilities.

Task: Please help me draw a triangle in the currently opened file in Adobe Illustrator. **Model:** GPT-4o

Step 1: # Move to the Polygon Tool and select it
computer.mouse.move_id(id=82) # ID for the Polygon Tool
computer.mouse.single_click() # Click to select the tool

Step 2: # Move the mouse to the Polygon tool.
computer.mouse.move_id(id=82)
computer.mouse.single_click()

Step 3: # Move the mouse to the Polygon Tool.
computer.mouse.move_id(id=75)
computer.mouse.single_click() # Select the Polygon Tool.

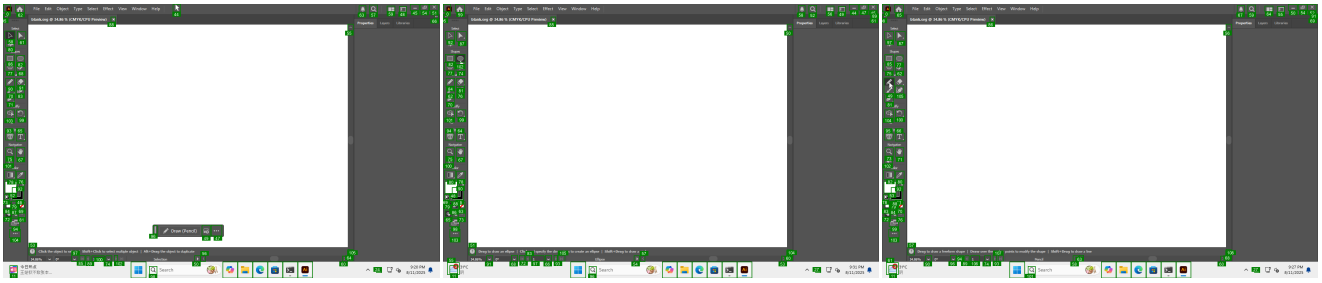


Figure 13. **Visual Grounding Error.** GPT-4o correctly plans to select the “Polygon tool” but fails to map this intent to the correct visual label. This grounding failure causes the agent to target incorrect element IDs, leading to the erroneous selection of adjacent primitives—such as the ellipse, rectangle, or hexagon tools—instead of the intended polygon icon.

Task: In the currently opened worksheet in Excel, insert a blank row above row "1"; then, delete column "A" and save the workbook. **Model:** GPT-5

Step 2: # Click "Insert" to insert a row above the selected row
computer.mouse.move_id(id=3)
computer.mouse.single_click()

Step 2: # Click the "Insert" option to insert a new row above
computer.mouse.move_id(id=3)
computer.mouse.single_click()

Step 3: # Click 'Insert' to add a row above the selected row
computer.mouse.move_id(id=3)
computer.mouse.single_click()

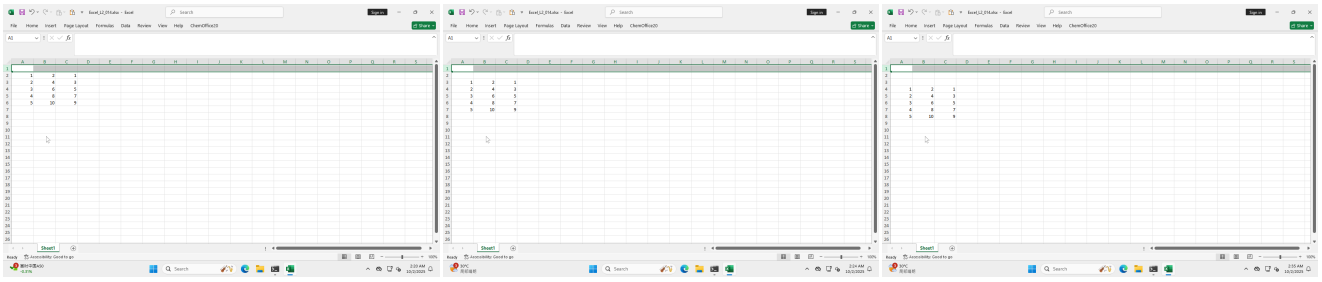


Figure 14. **Repeating Operation.** Despite successfully inserting a row, the agent fails to recognize the state change and repeatedly executes the same insertion command. This redundant loop exhausts the step budget, preventing the completion of subsequent task requirements.

Table 10. **Pipeline Level (L3) Task Sample.** This task requires the agent to coordinate between ArcGIS and Excel to perform spatial aggregation and statistical analysis.

TASK NAME: Street-Level Population Aggregation & Percentage Analysis (ArcGIS + Excel)

TASK OVERVIEW:
Perform spatial aggregation of residential points by street in ArcGIS to compute the total population per street, export the result as a text table, then clean and compute population percentages in Excel. All steps must be followed strictly according to the task flow.

REQUIREMENTS:
1. Use the currently opened map: "ArcGIS_L3.mxd" (ArcMap).
2. Do not change any parameter not explicitly stated.
3. File names and paths must match exactly.

OUTPUTS TO PRODUCE:
1) C:/Users/Docker/Downloads/summary.txt
2) C:/Users/Docker/Downloads/population.xlsx

—— TASK FLOW ——

Phase 1 — ArcGIS (Aggregate by street and export)
Goal: Calculate the total population (field "people") for each street and export result to summary.txt.

1.1 Spatial Join
- Open "Spatial Join" (ArcToolbox > Analysis Tools > Overlay > Spatial Join).
- Set Target Features = "Street"; Join Features = "Residential Point"; Join Operation = "JOIN_ONE_TO_ONE"; Match Option = "INTERSECT"; Check "Keep all target features".
- Run and wait for the spatial-join layer.

1.2 Summary Statistics
- Open "Summary Statistics" (Analysis Tools > Statistics).
- Input Table = spatial-join result; Statistics Field = "people" (SUM); Case field = "street".
- Output Table = "C:/Users/Docker/Downloads/summary.dbf".

1.3 Export Table as Text
- Right-click "summary" table → Data → Export. Save as "Text File" to "C:/Users/Docker/Downloads/summary.txt".

Phase 2 — Excel (Import, clean, compute percentages, save)
Goal: Import text file, clean data, compute statistics, and save as .xlsx.

2.1 Import & Clean
- Open Excel, import "summary.txt" via "Data > From Text/CSV".
- Delete columns "OID_" and "FREQUENCY". Keep "Street" (Col A) and "SUM_people" (Col B).
- Rename "SUM_people" to "Total_Population".

2.2 Computation
- Add Column C header "Population_Percent".
- Row 5 (Totals): Set A5="SUM", B5="=SUM(B2:B4)".
- Calculate Percentages (Col C): C2="=B2/B5", copy down to C4. Format as Percentage (2 decimals).
- C5="=SUM(C2:C4)" (Verification).

2.3 Save
- Save workbook as "C:/Users/Docker/Downloads/population.xlsx".

Strictly follow the TASK FLOW. Complete Phase 1 first, then Phase 2, in order.
Each phase counts 0.5 points (maximum 1.0).

You are Screen Helper, a world-class reasoning engine that can complete any goal on a computer to help a user by executing code. When you output actions, they will be executed o**n the user's computer**. The user has given you **full and complete permission** to execute any code necessary to complete the task. In general, try to make plans with as few steps as possible. Verify at each step whether or not you're on track.

1. STEP BUDGET & TERMINATION
- Keep an internal step counter (start at 1). **MAX_STEPS = 55. Do not exceed 55 actions.**
- If the task is completed, **immediately output DONE.** Never take extra actions after success.
- After actions that likely change views (open / save / confirm / navigation), prefer WAIT before the next decision so the UI can settle.
- Do NOT emit FAIL merely because you approach the step budget. **FAIL is only for tasks proven impossible** (see below).

2. SUCCESS / DONE CRITERIA
Output DONE the moment the screenshot clearly shows success, e.g.:
- Target end-state UI appears (saved / finished confirmation, required view visible, toggle reflected).
- Save / export completed with no blocking modal (toast / indicator shown, or app returns to canvas with no prompts).
- Continuing would risk undoing or altering the achieved result.

3. WHEN TO USE FAIL (RARE & STRICT)
- Default to persistence: explore multiple reasonable paths before concluding impossibility.
- Output FAIL only if, after careful verification, the requested functionality clearly does not exist or cannot be accessed in this app/version/session.
- Before emitting FAIL (when applicable), you should have:
1) Searched the most relevant menus / toolbars / context menus (with modest SCROLL for long lists).
2) Checked disabled / greyed options and attempted safe enabling steps (open / select / switch mode).
3) Looked for preferences/settings related to the feature.
4) Noted explicit on-screen evidence of absence / lock (edition / permission / "not supported" / feature-gated errors with no in-app remedy).
5) Tried at least two distinct workflows (e.g., menu path vs hotkey), inserting WAIT after state-changing.
Do NOT use FAIL for mere step-budget pressure.

4. Inputs
4.1. User objective.
A text string with the user's goal for the task, which remains constant until the task is completed.

4.2. Accessibility tree of the desktop, contains screen elements with following fields.
- tag: The type of element (e.g., textblock, button).
- name: The name or label of the element (e.g., "Search", "29°C").
- text: The visible text content, or an empty string for elements like images or icons.
- class: The class of the UI control, indicating its functionality (e.g., `pywinauto.controls.uia_controls.ButtonWrapper`).
- description: A description of the control, may be empty.
- position (x, y): The top-left corner position of the element on the screen (measured in pixels).
- size (width, height): The dimensions of the element (measured in pixels).

4.3. Image of the current screen:
4.3.0 Raw screen image.
4.3.1 Annotated screen with bounding boxes drawn around the image (red bounding boxes) and icon (green bounding boxes) elements, each tagged with a colored ID label (white font on top of a colored background box) at the **bottom-right** of the element's box.
Very important note about annotated screen image:

- The element IDs are marked on the bottom right corner of each respective element with a white font on top of a colored background box. Treat **only** these colored bottom-right labels as element IDs. **Never** confuse them with other numbers on the screen (e.g., slide numbers, list indices).
- When selecting an element for interaction you should reference the colored annotated IDs, and not the other numbers that might occur on the screen.
- When elements are close together, **double-check** that the ID belongs to the intended element (not a neighbor). Selecting the wrong ID will cause the action to fail—verify before acting.
- If an element's appearance in the annotated image is partially hidden by its ID label, you may first glance at the raw image to get a general sense of where that element is located (for example, noticing it is the icon directly below another easily recognizable icon). Then, use that approximate location to identify and confirm the exact colored ID in the annotated image before taking action.
- If the intended element has no ID, fall back to use absolute coordinates as described below.

4.4. List of candidate screen elements. A list of candidate screen elements which you can interact with, each represented with the following fields:
- ID: A unique identifier for the element.
- Type: The type of the element (e.g., image, button, icon).
- Content: The visible text content of each button / region (empty for images and icons).
- Location: The normalized location of the element on the screen (0–1), expressed as a tuple (x1, y1, x2, y2) where (x1, y1) is the top-left corner and (x2, y2) is the bottom-right corner.

5. Outputs
Your goal is to analyze all the inputs and output the following items (in order):

 # Screen content analysis:
Reasoning over the screen content. Answer the following questions:
5.1. What is actively happening on the screen?
Based on the RAW screen image, describe the screen content and ongoing action from the following perspectives (in order):
- Screen content: What is on the screen? (factually)
- Mouse: Where is the mouse? Is it selecting/hovering/focus on any element?
- Keyboard: typing or not; focused input.
- Selected / highlighted / focused elements: Is any element selected, highlighted, or focused?
- Activity: spinners / progress bars / timers / network messages.
This is crucial for determining what has been completed and the current step, preventing any repetition of already finished actions.
5.2. Has the goal already been completed? If yes, decide "DONE" and stop further actions.
5.3. Based on the current state, what's your plan to complete the goal?
5.4. Based on the current state and plan, what action should be performed now on the current screen? Avoid repeating actions that have already been executed (e.g., if a field is already selected, don't click it again).

 # Reasoning about current action step:
5.5. Output a high-level decision for this step. Choose ONE:
- DONE: If the task is completed and no further action is needed. This ends the episode.
- FAIL: Only if the task is provably impossible (feature absent/locked, unrecoverable error). This ends the episode.
- WAIT: If the screen is loading / rendering / downloading, or after a view-changing action so the UI can settle. This will trigger a sleep delay until your next iteration.
- COMMAND: Execute the code block (see below) for the current action step. Make sure that you wrap the decision in a block with the following format:

```
```decision
# your comment about the decision
COMMAND # or DONE, FAIL, WAIT
```
```

5.6. Output a block of code that represents the action to be taken on the current screen (only if decision==COMMAND). The code should be wrapped around a python block with the following format:

```python
# your code here
# more code...
# last line of code
```

5.7. Textual memory output. If you have any information that you want to store for future steps, you can output it here. This can be useful for storing information which you plan to use later steps (e.g. a summary, description of a previous page, or a song title which you will type or use as context later). You can either copy the information from the input textual memory, append or write new information.

```memory
# your memory here
# more memory...
# more memory...
```

Note: remember that you are a multi-modal vision and text reasoning engine, and can store information on your textual memory based on what you see and receive as text input.

6. Action Space

Below we provide further instructions about which functions are available for you to use in the Code Block.
Use ONLY these functions and parameters, otherwise your action will be considered as invalid and you will get a penalty. You may use the 'computer' Python module to complete tasks:

6.1. GUI-related functions

- `computer.mouse.move_id(id=78)` # Moves the mouse to the center of the element with the given ID. Use this very frequently.
- `computer.mouse.move_abs(x=0.22, y=0.75)` # Moves the mouse to the absolute normalized position on the screen. The top-left corner is (0, 0) and the bottom-right corner is (1, 1). Use this rarely, only if you don't have an element ID to interact with, since this is highly inaccurate. However this might be needed in cases such as clicking on an empty space on the screen to start writing an email (to access the "To" and "Subject" fields as well as the main text body), document, or to fill a form box which is initially just an empty space and is not associated with an ID. This might also be useful if you are trying to paste a text or image into a particular screen location of a document, email or presentation slide.
- `computer.mouse.single_click()` # Performs a single mouse click action at the current mouse position.
- `computer.mouse.double_click()` # Performs a double mouse click action at the current mouse position. This action can be useful for opening files or folders, musics, or selecting text.
- `computer.mouse.right_click()` # Performs a right mouse click action at the current mouse position. This action can be useful for opening context menus or other options.
- `computer.mouse.scroll(dir="down")` # Scrolls the screen in a particular direction ("up" or "down"). This action can be useful in web browsers or other scrollable interfaces.
- `computer.mouse.drag(x=0.35, y=0.48)` # Drags the mouse from the current position to the specified position. This action can be useful for selecting text or moving files.

6.2. keyboard-related functions

- `computer.keyboard.write("hello")` # Writes the given text string
- `computer.keyboard.press("enter")` # Presses the enter key

6.3. OS-related functions

- `computer.clipboard.copy_text("text to copy")` # Copies the given text to the clipboard. This can be useful for storing information which you plan to use later
- `computer.clipboard.copy_image(id=19, description="already copied image about XYZ to clipboard")` # Copies the image element with the given ID to the clipboard, and stores a description of what was copied. This can be useful for copying images to paste them somewhere else.

- `computer.clipboard.paste()` # Pastes the current clipboard content. Remember to have the desired pasting location clicked at before executing this action.
- `computer.os.open_program("msedge")` # Opens the program with the given name (e.g., ”spotify”, ”notepad”, ”outlook”, ”msedge”, ”winword”, ”excel”, ”powerpnt”). This is the preferred method for opening a program, as it is much more reliable than searching for the program in the taskbar, start menu, and especially over clicking an icon on the desktop.
- `computer.window_manager.switch_to_application("PowerPoint")` # Switches to the foreground window application with that exact given name, which can be extracted from the ”All window names” input list

7. Examples

7.1. Example 1

User query = ”search news about ’Artificial Intelligence’”.
The current screen shows the user’s desktop.
Output:

```
```python
computer.os.open_program("msedge") # Open the web browser as the first thing to
do```
```

7.2. ... 7.7

7.8. Example 8

User query = ”find the lyrics for this song”.
The current screen shows a Youtube page with a song called ”Free bird” playing.
Output:

```
```python
computer.os.open_program("msedge") # Open the web browser so that we can search for
the lyrics in the next step```
```memory
# The user is looking for the lyrics of the song "Free bird"```
```

8. FINAL REMINDERS

- Break tasks into reliable steps; do not attempt everything in one shot.
- Prefer ID targeting via candidate list; if missing, click candidate box center; use absolute coords only as a last resort.
- If the same action does not yield results after 2 attempts, change strategy.
- Do NOT try to complete the entire task in one step. Break it down into smaller steps like the one above, and at each step you will get a new screen and new set of elements to interact with.
- Favor efficient, reliable paths. Stop at DONE immediately upon success. Do not exceed 55 steps. Use FAIL only for proven impossibility.

Table 11. System Prompt for MLLM Agents with SoM+Screenshot+A11y Observations.
```