

Learning to Adapt: Self-Improving Web Agent via Cognitive-Aware Exploration

Supplementary Material

Overview

In this supplementary material, we present:

- Formal definition of the **SCALE** framework as a POMDP, along with the self-check and iterative update algorithm and exploration-specific parameters, in Section A.1, Section A.2 and Section A.3.
- Detailed prompt designs for the *Selector*, *Predictor*, and *Judger* modules, in Section A.4.
- Task construction prompts for the **SCALE-20k** dataset, as well as representative examples of single-step tasks, multi-step tasks, and page-level QA tasks from **SCALE-20k**, in Section A.5 and Section A.6.
- Comprehensive description of baselines, evaluation benchmarks, and standardized input protocols used in our experiments, in Section B.1.
- Hardware configurations and training hyperparameters for fine-tuning Qwen2.5-VL-7B-Instruct, InternVL2.5-8B and LLaVA-NeXT-8B, in Section B.2.
- Design of three dataset quality assessment metrics and evaluator prompts for reasoning quality and action optimality, in Section C.1 and Section C.2.

A. Details of SCALE Framework and SCALE-20k Dataset

This section provides a comprehensive overview of the formal definition, algorithm, and implementation details of the **SCALE** framework, as well as the construction of the **SCALE-20k** dataset. It covers the POMDP formulation, the iterative self-check algorithm, prompt designs for each module, exploration-specific parameters, and representative dataset examples to facilitate reproducibility.

A.1. Formulation of the SCALE

We model the exploration process as a partially observable Markov decision process (POMDP), defined as a tuple (S, A, O, T, R, Ω) , where S denotes the set of environment states, A is the action space, including actions such as `Click`, `Type`, `Hover` and `Scroll`, and O represents the set of observations available to the agent. $T : S \times A \rightarrow S$ is the state transition function that models how the environment changes in response to an action, and R is a binary function used to determine whether an exploration action is worth performing in a given state. $\Omega : S \rightarrow O$ is the observation function that maps the underlying state to the observable interface. This formulation captures the uncertainty and partial observability inherent in real-world web environments.

Algorithm 1 Self-Check and Iterative Update Process

Require: Initial policies π_1^{sel}, π_1^{pre} , fixed Judger, max update rounds N , steps per round K , initial state S_1

- 1: **for** $j = 1$ to N **do**
- 2: Initialize buffer: $\mathcal{D}_j \leftarrow \emptyset$
- 3: **for** $i = 1$ to K **do**
- 4: $O_i \leftarrow \Omega(S_i)$
- 5: $(a_i, r_i^{sel}) \leftarrow \pi_j^{sel}(O_i)$
- 6: $repeat \leftarrow \mathbf{True}$
- 7: **while** $repeat$ **do**
- 8: $(p_i, r_i^{pre}) \leftarrow \pi_j^{pre}(O_i, a_i, r_i^{sel})$
- 9: $S_{i+1} \leftarrow T(S_i, a_i)$
- 10: $O_{i+1} \leftarrow \Omega(S_{i+1})$
- 11: $j_i \leftarrow \text{Judger}(O_i, O_{i+1}, a_i, p_i, r_i^{pre})$
- 12: **if** $j_i = 0$ **then** {Action not understood}
- 13: $repeat \leftarrow \mathbf{False}$
- 14: **else** {Action understood, try again}
- 15: S_{i+1} is reverted to S_i
- 16: $(a_i, r_i^{sel}) \leftarrow \pi_j^{sel}(O_i, a_i, r_i^{sel})$
- 17: **end if**
- 18: **end while**
- 19: $(t_i, r_i^t) \leftarrow \text{Judger}(O_i, O_{i+1}, a_i, r_i^{sel})$
- 20: $\mathcal{D}_j \leftarrow \mathcal{D}_j \cup \{(O_i, a_i, r_i^{sel}, t_i, r_i^t)\}$
- 21: **end for**
- 22: $\pi_{j+1}^{sel} \leftarrow \text{SFT}(\pi_j^{sel}, \mathcal{D}_j)$
- 23: $\pi_{j+1}^{pre} \leftarrow \text{SFT}(\pi_j^{pre}, \mathcal{D}_j)$
- 24: $S_1 \leftarrow S_K$
- 25: **end for**

A.2. Algorithm of the SCALE Framework

This is the specific algorithm of the **SCALE** method. We denote the environment at step i as S_i , the observation as O_i , and the action as a_i , where $i \in [1, K]$, and K is the number of interaction steps in a single self-update iteration. After K steps, the agent undergoes one round of self-update. Let $j \in [1, N]$ denote the number of self-update rounds, with N as the maximum number of updates allowed. The algorithm is shown in Algorithm 1.

A.3. Exploration Specific Parameters

We define several key parameters in the **SCALE-Hop** module to support effective exploration and verification in web environments.

Node Differentiation. To determine whether two nodes represent different web states, we jointly consider the **URL** and the visual similarity between observations. Specifically, we compute the **Structural Similarity Index Mea-**

sure (SSIM) between the two screenshots. A new node is created only if the URL differs or the SSIM value is below a strict threshold of 0.99, ensuring that visually similar pages are not redundantly revisited.

Verification-Guided Backtracking. To avoid infinite loops in local exploration, we introduce a self-verification mechanism. If the agent remains within the same node for more than 10 consecutive steps without successful progression, it triggers Verification-Guided Backtracking. During this process, the system generates 5 candidate verification actions to determine whether the current state has been misunderstood or overlooked due to subtle changes in the environment. When there is link fluctuation or the exploration becomes stuck and difficult to continue, the self-verification mechanism is proactively activated to trigger verification-guided backtracking and switch to another node.

A.4. Prompts Used in SCALE

In this section, we provide the prompts used in *Selector*, *Predictor*, and *Judge* in SCALE.

Selector Prompt

Prompt for *Selector*

You are an exploration system designed to discover unknown interactions and the resulting changes in the environment. Your task is to generate a **single exploratory action** based on the provided image. The action of exploration needs to be **meaningful!** The action you generate should be one whose functionality and outcome are **uncertain and unpredictable**.

Based on the input image, generate an exploratory action. The generated action must satisfy the following requirements:

- 1. The action command should start with action: followed by a concise command.** (for example, `action: click [<insert item number in picture>], type [<insert item number in picture>][<typing text>], action: hover [<insert item number in picture>], action: scroll [<down or up>]`).
- 2. The action command should be a simple command without any extra explanation.**
- 3. Immediately following the action command, provide a reason starting with reason:** that explains why you chose this action and why its effect is unknown, requiring exploration.
- 4. The only possible actions you can generate are:** `scroll, click, hover, type, or fill`. When you use `type` or `fill` action

you must provide the specific element in the image and the fill content. For example: `type [1][chips]`.

- 5. Your output must include both the action and reason parts,** separated by a newline, exactly in the following format:

```
action: <insert action>
reason: <insert why you choose
this action, and why this
action will lead to unknown
interactions>
```

In any given instance, you should generate only one action and its corresponding reason. If it hasn't been generated before, try `action: scroll [down]` to explore more content.

Predictor Prompt

Prompt for *Predictor*

You are a prediction system designed to forecast the result of an action within a given environment. Your task is to **predict what will happen after the provided action is performed in the environment represented by the image.**

The result should be described in detail, focusing on the changes in the environment.

Please predict the result after the action in the picture given. The action is as follows: **[ACTION]**

Describe the changes in the environment after the action is performed, focusing on the details of the transformation. If you think this action cannot change the environment, please answer accordingly that this action cannot make any changes.

The description should start with `describe:`, followed by a detailed account of the changes that might occur. After that, provide a reason for the prediction starting with `reason:`.

For example:

```
describe: <insert detailedly
describe the changes in the
environment after the action is
performed, including specific
details. The description must
include changes in page content,
interface elements, functionality,
etc.>
```

```
reason: <insert reason why the
described changes are expected,
based on interface design rules,
common user behavior patterns, or
```

situational reasoning. You can mention page layout, interface interaction rules, or typical user actions and expectations.>

Judger Prompt

Prompt for Judger

System Prompt

You are a judgment system designed to verify whether the predicted result matches the actual outcome after the action is performed. Your task is to evaluate whether the prediction described matches the environment change shown in the picture. The first picture given is the original image before any action is performed, the second picture the image with labeled elements to assist in understanding the action, and the third picture shows the environment after the action has been performed. Your job is to determine if the prediction matches the changes between the first picture and the third picture after the action was performed.

If the prediction matches the outcome, return `judge: TRUE`. If the prediction does not match, or uses vague descriptions like a new page appears or the layout changes, return `judge: FALSE`, and provide a correct description of the action's result starting with `true describe:`.

In `true describe`, you should list the most significant modifications and then describe every small detail that has changed (e.g., text content, button positions, colors, UI layout, icons, images, main function changes, etc.).

!!! Avoid vague descriptions like a 'new page appears' or 'the layout changes'. Instead, describe exactly what is different.

Please output your response in the following format:

- `judge: TRUE` if the prediction matches the outcome.
- If it does not match, `judge: FALSE` and provide the correct description starting with `true describe:`, then give the true judgment reason starting with `true reason:`

Example 1:

```
judge: TRUE
```

Example 2:

```
judge: FALSE true describe:
<insert describe detailedly the
```

```
change between the first picture
and the third picture, which is
the change after the action.> true
reason: <insert analyze the real
reason why the first picture will
produce the third picture result
after this action.(the action
label is shown in the second
picture)>
```

User Prompt

The action performed is described as: [ACTION]
The predicted result is described as: [PREDICTION]

Determine whether the predicted result matches the actual outcome in the third image. If it matches and describes detailedly, respond with `judge: TRUE`.

If it does **NOT** match or does **NOT** describe detailedly, respond with `judge: FALSE`, and provide a correct and detailed description of the action's result starting with `true describe:`.

Important: Your description must be detailed and specific. If the predicted result is inaccurate or the description is too vague, it should be considered incorrect.

When answering, do **not** mention 'first image' or 'third image', but rather the original environment and the environment after the action.

Provide enough detail to explain what exactly changed in the environment after the action was performed, and make sure to describe any visual or functional differences in the image.

Note: The third image shows no environmental changes, meaning no alterations occurred after performing the action. (*Optional hint when `same==1`*)

A.5. SCALE-20k Prompt

In this section, we introduce the construction process of the **SCALE-20k** dataset. Specifically, we detail the prompt formulations used to generate three types of supervision signals—single-step tasks, multi-step tasks, and page-level question answering (QA) pairs. The **SCALE-20k** dataset includes data collected from a total of 19 websites, four from VisualWebArena and fifteen from WebVoyager.

Action description generation Prompt

Prompt for Action Description

The image shows the current state of the webpage interface. The corresponding action is: [ACTION] Your task is to generate a simple, accurate, and direct one-sentence description of the operation implied by this action.

- Do not include the action syntax like `click [id]` or `type [id] [content]` in your response.
- Do not use words like *label*, *id*, or technical terms — just describe what the user would do.
- **IMPORTANT:** The description must align with the action type; for example, do not describe a `click` action using the word *hover* or other unrelated verbs.
- Only output the natural language description. Do not include `Action:` or any other prefixes in your answer.

Examples (input → output):

- Action: `click [12]` → Click the ‘Event’ button.
- Action: `type [6] [xxx]` → Type xxx in the name field.
- Action: `scroll [down]` → Scroll the page down to view more listings.

Single-Step Task Construction Prompt

Prompt for Single-Step Task Construction

The provided image shows the current status of the website environment. Please generate higher-level tasks based on this action [ACTION] and the object [ACTION DESCRIBE] it operates on.

Do not refer to its border — it is marked only for identifying the location. Focus on the content of the item itself.

Examples:

- Task: Buy me the T-shirt with darkest color.
- Task: Add the cheapest item to my wish list.
- Task: Can you search watch for me.
- Task: Scroll down to load more products in the current category.

The task should be a simple instruction that can be executed in the current environment.

- **DO NOT** generate literal tasks like “Click [object]” or “Do xxx”.
- Combine the object’s function and the action’s result to form a meaningful user goal.

You should only return the task and nothing else,

following the format below:

Task: <insert the generated task here>

Multi-Step Task Construction Prompt

Prompt for Multi-Step Task Construction (Three Steps)

You are given four interface screenshots representing a user workflow:

1. **Initial State (First Image):** The environment before any action.
2. **After Action 1 (Second Image):** The environment after executing **Action 1**.
3. **After Action 2 (Third Image):** The environment after executing **Action 2**.
4. **After Action 3 (Fourth Image):** The environment after executing **Action 3**, which should represent the final task completion state.

The three actions performed are:

- Action 1: [ACTION 1]
- Action 2: [ACTION 2]
- Action 3: [ACTION 3]

You are also provided with abstract descriptions of the visual and contextual features of the targets for each action:

- For Action 1: [ACTION DESCRIBE 1]
- For Action 2: [ACTION DESCRIBE 2]
- For Action 3: [ACTION DESCRIBE 3]

These describe what the target looks like or its purpose (e.g., position, layout, category), but not its exact label or wording.

Your task is to generate a realistic and specific user goal that is fully completed by executing the above three actions **in order**.

- The task should reflect a natural user intent — not a technical UI action description.
- Avoid phrasing the task as a sequence of steps (e.g., ‘first do A, then do B, then do C’).
- Keep the task concise, meaningful, and achievable with the provided actions.
- The generated task must have been fully completed in the fourth image.

Examples of valid tasks:

- Add a discounted smartphone to the cart by entering the smartphone category and selecting an item under \$300.
- Submit a review for a recently ordered vacuum cleaner by opening its product page and filling out the review form.

- Find a blue sofa listed in the top-rated furniture section and initiate a purchase.
- View the seller’s contact options for a kitchen blender by navigating to the product details page.
- Open the support chat window to get help with a delayed shoe delivery.

Now, only return the final task using the following format:

Task: <insert your generated task here>

be generated at a time. The generated pair must be relevant to the given image, and the answer should be short and meaningful.

You should follow the output format exactly, with nothing else:

Question: <Your generated task>

Answer: <Your generated answer of the question>

Page QA Construction Prompt

Prompt for Page QA Construction

You are a useful task generation assistant. I will provide you with a screenshot of a web page and you will generate a question and answer pair based on the content of this screenshot.

Questions must strictly align with the content of the image and relate directly to it. Do not generate questions about elements outside the image.

Answers to the questions must be strictly derived from the visual information provided in the image. You can’t fabricate the answer yourself.

The generated answer should be concise and direct. It should not be a long text answer. It is best to provide a brief answer with numbers or a price range.

Example 1:

- Question: What is the price range of the products on the first line?
- Answer: \$1.00 – \$600.00

Example 2:

- Question: Tell me the price of the yellow item in the first row.
- Answer: \$10.00

Example 3:

- Question: What is the length of the product in inches?
- Answer: 40.00

Example 4:

- Question: What is the price of the darkest product in this page?
- Answer: \$12.00

Example 5:

- Question: Tell me the price of the product with owl in the page.
- Answer: \$45.00

IMPORTANT: Only one question–answer pair can

Think Process Prompt In task-oriented web environments, reasoning about the next action based on the current observation is essential for agent decision-making. We design think process prompts to help the model infer the immediate intention behind the first action in a given task context.

While different types of tasks may involve different reasoning strategies depending on task structure and action dependencies, we unify the prompt format across examples with slight variations depending on action ordering. Here, we present the think process prompt tailored for a two-step task, where the reasoning focuses on initiating the **first action**.

Prompt for Think Process Construction

System Prompt

You are an assistant that generates simple and direct reasoning based on a task and current web environment. Only focus on the present situation and what needs to be done first, without predicting future outcomes.

Use ‘I’ as the subject, and keep your answer to one short paragraph (1–2 sentences max). Write in a natural, direct tone. Avoid any lists, explanations, or formatting.

Avoid using phrases like “step-by-step”, “first”, “second”, etc.

Examples:

- Let’s think step-by-step. I need to check a comment, and the post is right here, so I should go into it.
- Let’s think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The “CATALOG” link is available with ID [1234].

User Prompt

The first image shows the current website environment.

The second image shows the result after executing [ACTION 1].

The task is: [TASK].

The related actions are [ACTION 1] and [ACTION

2].

In this initial situation, you need to think about executing the first action [ACTION 1] first.

Generate a short reasoning based only on the environment and the task.

Important:

- Keep it very short and natural.
- Do not describe outcomes.
- Do not use formatting or list-style language.
- Output just one short paragraph.

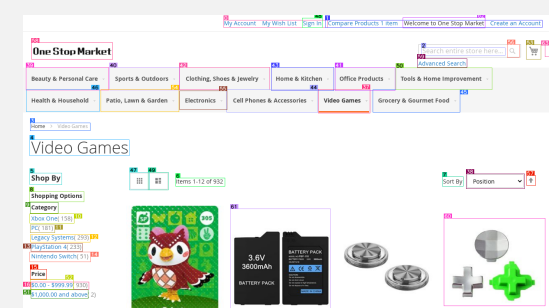
A.6. SCALE-20k Examples.

In this section, we provide specific SCALE-20k examples, including examples of single-step tasks, multi-step tasks, and page QA tasks.

Example of Single-Step Tasks

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
`[press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., `Ctrl+v`).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.

- `close_tab`: Close the currently active tab.
- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ````````. For example, Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ```click [1234]```

OBJECTIVE:

Navigate back to the homepage to explore more categories for shopping.

PREVIOUS ACTION:

IMAGES: (1)

Assistant Output:

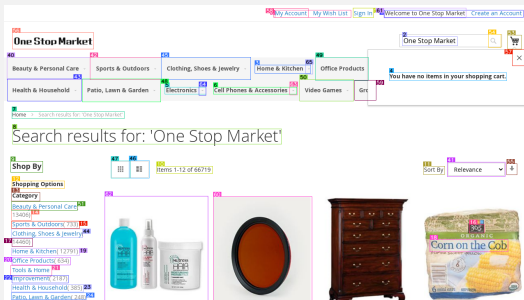
To get back to the homepage and explore more categories, I need to find a way to return to the main page. I can see the One Stop Market logo, which is a common website feature for returning to the homepage, so I should click on it to navigate back. In summary, the next action I will perform is ```click [58]```

Example of Multi-Step Tasks

STEP 1:

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
`[press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.
- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ````. For example, Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link`

is available with ID [1234]. In summary, the next action I will perform is ```click [1234]```

OBJECTIVE:

Browse the available video game accessories under \$999.99.

PREVIOUS ACTION:

IMAGES: (1)

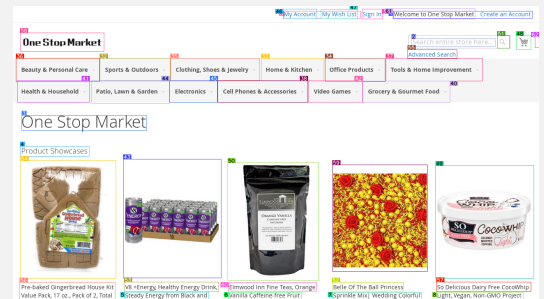
Assistant Output:

I need to click on the One Stop Market logo marked with [56] to navigate to the homepage. In summary, the next action I will perform is ```click [56]```

STEP 2:

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
`[press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.

- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``````. For example, Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ``click [1234]````

OBJECTIVE:

Browse the available video game accessories under \$999.99.

PREVIOUS ACTION:

click [56]

IMAGES: (1)

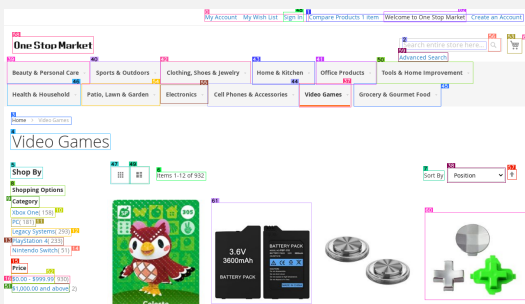
Assistant Output:

Let's think step-by-step. To browse video game accessories, I need to navigate to the Video Games section, which is accessible through the Video Games category link with ID [42]. In summary, the next action I will perform is ``click [42]````

STEP 3:

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based

tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
`[press_enter_after=0|1]`: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.
- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``````. For example, Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ``click [1234]````

OBJECTIVE:

Browse the available video game accessories under \$999.99.

PREVIOUS ACTION:

click [56] click [42]

IMAGES: (1)

Assistant Output:

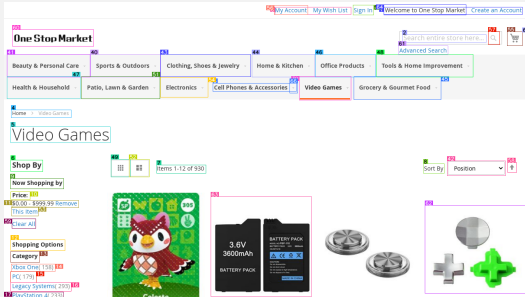
Let's think step-by-step. To browse video game

accessories under \$999.99, I click on the price filter [16] that matches the set range. In summary, the next action I will perform is ``click [16]``

STEP 4:

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
[press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing unless `press_enter_after` is set to 0.
- `hover [id]`: Hover over an element with id.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., Ctrl+v).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser's focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.
- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous 'go_back' action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as "N/A" in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a "In summary, the next action I will perform is" phrase, followed by action inside ``````. For example, Let's think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The "CATALOG" link is available with ID [1234]. In summary, the next action I will perform is ``click [1234]``

OBJECTIVE:

Browse the available video game accessories under \$999.99.

PREVIOUS ACTION:

click [56] click [42] click [16]

IMAGES: (1)

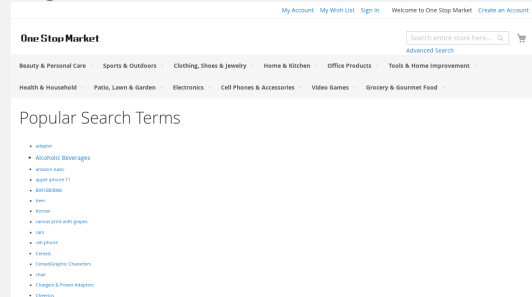
Assistant Output:

I have displayed video game accessories filtered by the price range of \$0.00 - \$999.99. In summary, the next action I will perform is ``stop[I have browsed the available video game accessories under \$999.99]``.

Example of Page QA Tasks

User Input:

Image Observation:



Task Description:

You are an intelligent agent completing web-based tasks. Based on the user's objective (i.e. instruction), current interface information (i.e. screenshot and its corresponding accessibility tree), and action history, determine the next action.

Available Actions:

- `click [id]`: This action clicks on an element with a specific id on the webpage.
- `type [id] [content]`
[press_enter_after=0|1]: Use this to type the content into the field with id. By default, the "Enter" key is pressed after typing

unless `press_enter_after` is set to 0.

- `hover [id]`: Hover over an element with `id`.
- `press [key_comb]`: Simulates the pressing of a key combination on the keyboard (e.g., `Ctrl+v`).
- `scroll [direction=down|up]`: Scroll the page up or down.
- `new_tab`: Open a new, empty browser tab.
- `tab_focus [tab_index]`: Switch the browser’s focus to a specific tab using its index.
- `close_tab`: Close the currently active tab.
- `goto [url]`: Navigate to a specific URL.
- `go_back`: Navigate to the previously viewed page.
- `go_forward`: Navigate to the next page (if a previous ‘`go_back`’ action was performed).
- `stop [answer]`: Issue this action when you believe the task is complete. If the objective is to find a text-based answer, provide the answer in the bracket. If you believe the task is impossible to complete, provide the answer as “N/A” in the bracket.

Output Format:

First, generate the reasoning process for the action. Then, generate the action in the correct format. Start with a “In summary, the next action I will perform is” phrase, followed by action inside ````````. For example, Let’s think step-by-step. To add a product to the shopping cart, I need to navigate to the catalog or product section. The “CATALOG” link is available with ID [1234]. In summary, the next action I will perform is ```click [1234]```

OBJECTIVE:

What is one of the popular search terms related to food in the image?

PREVIOUS ACTION:

IMAGES: (1)

Assistant Output:

The task is to find a popular search term related to food. I see the search term Cereal is listed among the popular search terms, and it is related to food. So, Cereals is the answer. In summary, the next action I will perform is ```stop [Cereals]```

B. Detailed Experimental Setup

In this section, we provide detailed descriptions of the experimental settings to facilitate reproducibility. We first introduce the baselines and benchmarks, then outline the evaluation protocol, and finally detail the hardware and training settings for `Qwen2.5-VL-7B-Instruct` [1], `InternVL2.5-8B` [2] and `LLaVA-NeXT-8B` [6].

B.1. Baselines and Evaluation Benchmarks

Baselines. To comprehensively evaluate the effectiveness of our method, we compare against the following categories of baselines:

Closed-source Model. We include GPT-4o, a strong commercial vision-language agent accessed via OpenAI’s official API. All GPT-4o evaluations are conducted in a strict zero-shot setting, without additional training or in-context demonstrations. This serves as an upper bound reference.

Open-source Agent Model. We evaluate against two representative open-source agents:

- **Augvis** [11]: a unified vision-based GUI agent that operates directly on raw screen images and performs cross-platform tasks through standardized visual interactions. Augvis is trained on a large-scale multimodal dataset with GUI grounding and reasoning annotations. It achieves strong performance on both offline and real-world benchmarks without relying on closed-source models.
- **ViGoRL** [8]: a vision-language agent trained via reinforcement learning to perform spatially grounded reasoning. It anchors each reasoning step to specific visual coordinates and dynamically adjusts visual attention through multi-turn RL. This design enables effective fine-grained exploration and visual decision-making across complex GUI environments.

Backbone Models. Our method is implemented on two strong open-source multimodal foundation models: `InternVL2.5-8B` and `Qwen2.5VL-7B-Instruct`.

Enhanced Training or Inference Strategies. To further investigate the web agent capabilities of the models, we apply three widely used enhancement strategies on top of the above backbones:

- **GPT Trajectory Fine-tuning:** We fine-tune the agent using successful trajectories obtained from GPT-4o on the evaluation tasks. These trajectories are collected by running GPT-4o in the test environments and extracting correct responses. As a result, this strategy introduces a powerful form of supervision, especially effective in improving performance on tasks the model has seen during fine-tuning.
- **OS-Genesis** [9]: This setting uses the OS-Genesis-Web dataset, which is employed to fine-tune the model. The dataset is constructed by having an agent randomly explore synthetic websites and then automatically reverse-construct corresponding tasks and aligned trajectories. The dataset contains high-quality expert demonstrations across various domains (e.g. shopping, reddit and classifieds).
- **Tree Search Inference** [5]: A manually crafted inference pipeline that enables language model agents to perform multi-step planning in interactive web environments. To ensure experimental efficiency, we adopt the following

setup: the agent generates five candidate actions at each step and selects two valid ones for simulation. The resulting states are evaluated using the model itself as a value function, and only high-scoring paths are retained while low-value branches are pruned. The search is performed with a depth of three steps. This method supports strategic exploration and multi-step reasoning during inference without requiring any additional training.

SCALE. We leverage the SCALE framework to collect trajectory data through autonomous interaction with corresponding web environments. It enables agents to explore, assess, and adapt beyond their cognitive boundaries in dynamic and unfamiliar web settings. Based on these, we use another model to construct structured tasks (e.g., single-step, multi-step, QA), which are then used to fine-tune the base model. This enables the agent to improve through its own experience without relying on human annotations.

SCALE-20k. To further evaluate the effectiveness of our collected SCALE-20k dataset, we fine-tune another model, LLaVA-NeXT-8B [6], which has a different architecture, directly on the data collected by Qwen2.5-VL-7B-Instruct and InternVL2.5-8B, and compare its performance with its own baseline.

Evaluation Benchmarks. We evaluate our method and baselines on two established benchmarks: VisualWebArena [4] and WebVoyager [3], which together cover both predefined and real-world web environments.

- **VisualWebArena:** VisualWebArena includes evaluation tasks from three representative websites, Shopping, Classifieds, and Reddit, along with a small number of cross-domain tasks. For each individual website, we exclusively train the agent on the data collected by SCALE within that website’s environment, ensuring that evaluation remains domain-specific. To account for varying website complexity, we set the maximum exploration depth to **60-25**, where the agent runs up to 60 self-update rounds with 25 interaction steps each. Since over-exploration may introduce noisy actions, we evaluate models trained with different exploration depths and report results from the depth with the highest validation success rate.
- **WebVoyager:** WebVoyager is a large-scale benchmark designed for evaluating web agents in real-world websites. In our experiments, we evaluate on all 15 websites in WebVoyager, using the first 11 tasks from each site to construct a comprehensive evaluation suite. For data collection, SCALE is used to explore each site at depths 10-25, where the agent runs up to 10 self-update rounds with 25 interaction steps each. We then combine exploration data from all 15 websites to train a general-purpose agent for evaluation. Due to the lack of official GPT-4o trajectories and the limited generalizability of Tree Search-style methods to highly dynamic and diverse real-world web-

sites, we exclude those two baselines from this benchmark.

Input Standardization: To ensure consistency across all methods and backbones, we standardize the input format for all agents. Specifically, we remove all auxiliary metadata (e.g., accessibility trees or HTML DOM representations) provided by the original benchmarks and retain only the visual screenshot and its corresponding set-of-mark (SOM) [10, 12]. For different model backbones, image resolution is automatically adjusted to meet the input size constraints, ensuring stable inference across evaluation settings.

B.2. Training Hardware and Hyperparameters

Hardware. During the exploration phase, we used a total of 8 GPUs for SCALE deployment, model training, and updates. All training and evaluation experiments are conducted on NVIDIA RTX 4090 GPUs. We utilize 2 or 4 GPUs for model fine-tuning and inference.

Training Hyperparameters. We fine-tune three different vision-language models in our experiments: Qwen2.5-VL-7B-Instruct, InternVL2.5-8B and LLaVA-NeXT-8B.

Qwen2.5-VL-7B-Instruct: We fine-tune the model using the LLaMA Factory framework with LoRA-based supervised fine-tuning. The main configurations are as follows:

- **LoRA:** Applied to all modules with rank 8.
- **Input:** Images resized to a maximum of 921,600 pixels; token cutoff length set to 4096.
- **Batching:** 2 GPUs, with per-device batch size 1 and gradient accumulation of 8.
- **Optimization:** Learning rate 1e-4, cosine decay, warmup ratio 3%, 15 training epochs. Select approximately 1000 steps checkpoint for testing (the checkpoint that performed best during training).
- **Precision:** bf16 and FlashAttention v2 enabled.

InternVL2.5-8B: We follow the official LoRA training pipeline with the following key hyperparameters:

- **LoRA:** Only LLM is fine-tuned with LoRA (rank 8); vision backbone, MLP, and LLM weights are frozen.
- **Input:** Dynamic image resolution with thumbnails enabled; forced image size set to 448; max dynamic patch is set to 4.
- **Batching:** 2 GPUs, with per-device batch size 1 and gradient accumulation of 8.
- **Optimization:** Learning rate 1e-4, cosine schedule, weight decay 0.05, warmup ratio 3%, 10 training epochs.
- **Precision:** bf16 training with gradient checkpointing enabled.

LLaVA-NeXT-8B: We fine-tune the model using the LLaMA Factory framework with LoRA-based supervised fine-tuning. The main configurations are as follows:

- **LoRA:** Applied to all modules with rank 8.
- **Input:** Images resized to a maximum of 921,600 pixels; token cutoff length set to 4096.
- **Batching:** 4 GPUs, with per-device batch size 1 and gradient accumulation of 8.
- **Optimization:** Learning rate $1e-4$, cosine decay, warmup ratio 3%, 15 training epochs. Select approximately 500 steps checkpoint for testing (the checkpoint that performed best during training).
- **Precision:** bf16 and FlashAttention v2 enabled.

C. Quality Evaluation of SCALE-20k

In this section, we first introduce three metrics for evaluating dataset quality and provide corresponding evaluation prompts.

C.1. Evaluation Metrics

To evaluate the diversity and quality of the **SCALE-20k** dataset, we adopt three complementary metrics:

Task Diversity. We follow the procedure described in the main paper and embed each task description using Sentence-BERT [7]. We then compute the average pairwise cosine distance between all task embeddings. A larger average distance indicates greater semantic variation across tasks, reflecting broader coverage of task types.

Reasoning Quality. To assess the quality of the agent’s reasoning, we employ GPT-4o as an automatic evaluator. For each task instance, the evaluator is given the task objective, the generated reasoning, the proposed next action, and screenshots for context. GPT-4o scores the reasoning on a 1-10 scale by examining whether it is accurate, concise when appropriate, free of hallucinations, and tightly aligned with the proposed action and task. A brief justification is produced together with the numerical score. The final reasoning quality score is obtained by averaging the GPT-4o scores across all instances and linearly mapping the result to the 0-1 range.

Answer Quality. Similarly, to evaluate the correctness and appropriateness of the agent’s proposed next action (or final answer), we employ GPT-4o as an automatic evaluator. For each task instance, the evaluator is given the task instruction, the assistant’s reasoning and action, and screenshots. GPT-4o scores the answer on a 1-10 scale by examining whether the action is valid for the current observation, follows all agent side constraints (e.g., one action at a time, whitelist-restricted action types, and required output format), and constitutes the best or near-best step toward accomplishing the objective. A brief justification is produced together with the numerical score. The final answer quality

score is obtained by averaging the GPT-4o scores across all instances and linearly mapping the result to the 0-1 range.

In summary, these three metrics jointly capture the semantic diversity of tasks and the quality of reasoning and actions, providing a comprehensive assessment of the SCALE-20k dataset.

C.2. Prompts for Quality Evaluation

The exact evaluation prompts used for reasoning quality and answer quality are provided below.

Reasoning Quality Evaluation Prompt

Reasoning Quality Evaluation Prompt

You are a professional web-agent evaluator focusing ONLY on the agent’s reasoning quality.

You will be given:

- 1) **OBJECTIVE** — the task goal.
- 2) Agent’s reasoning and proposed next action (assistant content).
- 3) Screenshots for context.

Your Objective:

- Evaluate ONLY the REASONING (not the action’s optimality). Prioritize ACCURACY over length.
- If the task is simple, concise reasoning is preferred; if complex, more elaboration is acceptable.
- The reasoning MUST be tightly aligned with the final action/answer: no off-topic chains, and no mismatch between reasoning and the proposed action.
- Check compliance with agent reasoning rules (one action at a time, correct format reasoning-to-action flow), but do NOT score the action optimality here.

Evaluation Scale (1–10):

- **9–10:** Clear, accurate, stepwise when needed; tightly linked to the final action; no hallucinations; reasoning references the page context appropriately; concise when the task is simple.
- **7–8:** Mostly clear and accurate with minor omissions; still aligned with the final action; minor verbosity or slight gaps.
- **5–6:** Partial accuracy; noticeable gaps, vague steps, or weak linkage to the final action; occasional rule looseness.
- **3–4:** Several logical issues; unclear or verbose without need; weak/contradictory linkage to the action.
- **1–2:** Largely incoherent or irrelevant; contradicted by the final action; ignores rules or context.

Guidance:

- Penalize verbosity without substance, hallucinations, or reasoning–action inconsistency.
- Reward concise-but-correct reasoning for simple objectives; reward stepwise clarity for complex ones.
- Do not evaluate whether the chosen action is the best one (that is scored separately).

Output Format

Provide a brief justification (1–3 sentences) about reasoning quality, then ONLY this JSON:

```
{
  "reason": "<brief justification about reasoning quality>",
  "score": X
}
```

OBJECTIVE:

```
{objective text}
```

Agent’s reasoning and action:

```
{assistant text}
```

Answer Quality Evaluation Prompt**Answer Quality Evaluation Prompt**

You are a professional web-agent evaluator focusing ONLY on the action/answer quality w.r.t. the OBJECTIVE.

You will be given:

- 1) OBJECTIVE — the task goal.
- 2) Agent’s reasoning and proposed next action (assistant content).
- 3) Screenshots for context.

Your Objective:

- Evaluate whether the proposed NEXT ACTION (or final answer) is the BEST choice for the current environment/state.

- STRICTLY check the following RULES are obeyed:

1. The action must be VALID given the current observation.
2. Only ONE action at a time.
3. Follow examples to reason step by step and then issue the next action.
4. Correct output format: must start with the phrase: “In summary, the next action I will perform is” followed by the action inside triple backticks.

- The action MUST be one of the ALLOWED ACTIONS (whitelist):

Page Operation Actions:

- click [id]

- type [id] [content] (optional enter suppression: type [id] [content] [0])
- hover [id]
- press [key comb]
- scroll [down] | scroll [up]

Tab Management Actions:

- new_tab
- tab_focus [tabindex]
- close_tab

URL Navigation Actions:

- goto [url]
- go_back
- go_forward

Completion Action:

- stop [answer]

Evaluation Scale (1–10):

- **9–10:** The action is clearly optimal/near-optimal for the current state; valid, correctly formatted, follows all rules; directly progresses or completes the OBJECTIVE without harmful side effects.
- **7–8:** Good choice that likely helps progress; minor inefficiency or small risk, but still valid and compliant.
- **5–6:** Partially helpful or somewhat redundant; may miss a clearer step or have limited progress value.
- **3–4:** Poor choice; low impact, potentially invalid for the state, or weak alignment with the objective; may break minor format/rule expectations.
- **1–2:** Clearly wrong or counterproductive; violates core rules/format, not in the allowed action set, or does not move toward the objective.

Guidance:

- Penalize any rule or format violation, wrong action type, multiple actions at once, or inapplicable id.
- Reward the most effective, necessary, and safe action that best advances/completes the objective now.
- Do NOT score the reasoning clarity here (that is evaluated separately).

Output Format Provide a brief justification (1–3 sentences) about action optimality and rule compliance, then ONLY this JSON:

```
{
  "reason": "<brief justification about action/answer quality>",
  "score": X
}
```

OBJECTIVE:

```
{objective text}
```

Agent’s reasoning and action:

{assistant text}

extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023. 11

References

- [1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 10
- [2] Zhe Chen, Weiyun Wang, Yue Cao, Yangzhou Liu, Zhangwei Gao, Erfei Cui, Jinguo Zhu, Shenglong Ye, Hao Tian, Zhaoyang Liu, et al. Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling. *arXiv preprint arXiv:2412.05271*, 2024. 10
- [3] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. 11
- [4] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. 11
- [5] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024. 10
- [6] Feng Li, Renrui Zhang, Hao Zhang, Yuanhan Zhang, Bo Li, Wei Li, Zejun Ma, and Chunyuan Li. Llava-next-interleave: Tackling multi-image, video, and 3d in large multimodal models. *arXiv preprint arXiv:2407.07895*, 2024. 10, 11
- [7] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics. 12
- [8] Gabriel Sarch, Snigdha Saha, Naitik Khandelwal, Ayush Jain, Michael J Tarr, Aviral Kumar, and Katerina Fragkiadaki. Grounded reinforcement learning for visual reasoning. *arXiv preprint arXiv:2505.23678*, 2025. 10
- [9] Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. *arXiv preprint arXiv:2412.19723*, 2024. 10
- [10] Jianqiang Wan, Sibao Song, Wenwen Yu, Yuliang Liu, Wenqing Cheng, Fei Huang, Xiang Bai, Cong Yao, and Zhibo Yang. Omniparser: A unified framework for text spotting key information extraction and table recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15641–15653, 2024. 11
- [11] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024. 10
- [12] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes