
Self-Guided Hierarchical Exploration for Generalist Foundation Model Web Agents

Supplementary Materials

Anonymous Author(s)

Affiliation

Address

email

1 A Algorithm Details

2 In this section, we provide additional details on the core algorithmic components of SAGE. We first
3 present the Monte Carlo Tree Search (MCTS) rollout procedure used during low-level exploration,
4 followed by the evaluator training setup that enables robust supervision without relying on ground-
5 truth rewards.

6 A.1 MCTS Rollout Procedure

7 Our MCTS-based rollout strategy is inspired by prior work on planning and structured decision-
8 making [1, 2], with key adaptations for the web-based setting and multimodal agent interaction. The
9 search tree is initialized with a single root node representing the starting state of the task, typically
10 corresponding to the homepage URL.

11 Each MCTS iteration proceeds in four phases: *selection*, *expansion*, *simulation*, and *backpropagation*:

12 **Selection.** Starting from the root, the agent traverses the tree by selecting child nodes based on the
13 Upper Confidence Bound (UCB) score [3]:

$$\text{UCB}(u) = v_u + \epsilon \sqrt{\frac{\ln n_{\text{parent}}}{n_u}},$$

14 where v_u is the current value estimate of node u , n_u is the number of times node u has been visited,
15 and n_{parent} is the visit count of its parent node. The exploration coefficient ϵ encourages selection of
16 less frequently visited nodes to balance exploration and exploitation.

17 **Expansion.** Upon reaching a leaf node, we expand it using the multi-action agent policy. Specifi-
18 cally, the agent generates a set of candidate actions conditioned on the current state, and we create
19 child nodes for each resulting next state. The value v_u for each new node is initialized by querying
20 the proprietary model Claude Sonnet 3.5 [4] to estimate its likelihood of task success, providing a
21 strong prior for downstream planning.

22 **Simulation.** From the selected best-valued child node, we perform a standard linear rollout using
23 the remaining environment steps. During this simulation, the agent greedily selects the top-confidence
24 action at each step from its multi-action output, continuing until the task either succeeds or terminates.

25 **Backpropagation.** After completing the rollout, we receive a binary reward indicating task success
26 or failure. This reward is used to update the value estimate v_u of the expanded node. We then
27 backpropagate both the value and visit count updates recursively from the leaf node up to the root.

28 The MCTS process continues until a fixed number of iterations is reached. This planning strategy
29 allows the agent to explore promising trajectories in a structured manner, significantly improving
30 sample efficiency and task success in complex web environments.

31 A.2 Evaluator Training

32 To enable reliable supervision without access to ground-truth reward functions, we train a task
33 outcome evaluator using the open-source model Qwen2.5VL-7B [5]. The evaluator is designed
34 to predict task success or failure based on evidence collected during agent interaction. Its input
35 consists of: (1) the task instruction, (2) the final answer generated by the agent, (3) the full interaction
36 trajectory, and (4) multimodal evidence composed of the final three screenshots and their associated
37 accessibility tree. The output is a binary success label, optionally accompanied by a natural language
38 explanation.

39 We construct the training dataset in two stages. First, we bootstrap the evaluator using over 10,000
40 evaluation traces generated by Claude 3.5 Sonnet [4], treating its responses as pseudo-labels. These
41 include both reasoning and binary outcome labels. This initial phase enables the evaluator to inherit
42 strong generalization capabilities from the proprietary model.

43 In the second stage, we refine the evaluator using a smaller, high-quality human-annotated dataset.
44 Specifically, we randomly sample 1,200 trajectories from the initial dataset and manually annotate
45 their ground-truth success labels. Among these, we identify approximately 400 instances where
46 Claude’s predictions disagree with human judgment. For each of these error cases, we provide a
47 corrected binary label and, when relevant, a brief explanation describing the evaluation mistake. The
48 format remains consistent with the initial dataset, enabling seamless fine-tuning. This two-stage
49 process enhances the evaluator’s precision and robustness, particularly for nuanced or failure-prone
50 cases.

51 B Environment Details

52 In this section, we describe the multimodal web environments used in our experiments, including
53 implementation details, observation formatting, and action space definitions.

54 **Environment Overview.** Our environment is implemented using a combination of Selenium¹
55 and Playwright², and wrapped into a standardized Gym interface to support reinforcement learning
56 workflows. In addition to basic functionalities required for agent-environment interaction, we extend
57 the framework with support for saving and restoring environment states at arbitrary time steps. This
58 feature is critical for enabling tree-based search strategies such as MCTS, which require repeated
59 rollouts from intermediate checkpoints.

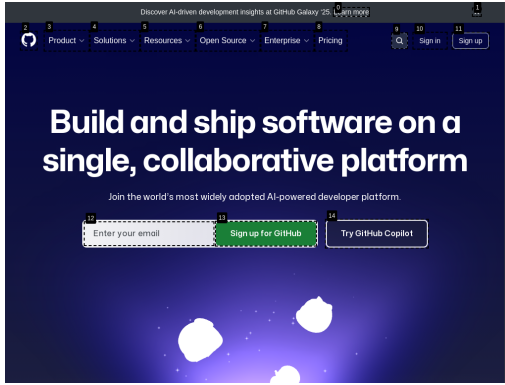
60 **Observation Details.** As illustrated in Figure 1, each observation consists of two aligned modal-
61 ities: a rendered screenshot with overlaid set-of-marks, and a structured accessibility tree. In our
62 implementation, we first extract the set-of-marks and the accessibility tree independently. We then
63 compute a matching between elements in the two modalities and align the numeric labels in the
64 set-of-marks with the corresponding nodes in the accessibility tree. This alignment ensures that
65 agents can refer to web elements consistently across visual and structural representations.

66 **Action Space.** Following the setup in PAE [6], we define a discrete action space that includes the
67 following interaction types:

- 68 • **Click:** Click on a labeled element such as a button or link.
- 69 • **Type:** Enter text into an input field.

¹<https://pypi.org/project/selenium/>

²<https://playwright.dev/>



```
[333] link 'Skip to content'
[360] alert 'Announcement'
[0] link 'Learn more'
[1] button 'Close'
[387] heading 'Navigation Menu'
[2] link 'Homepage'
[434] navigation 'Global'
[3] button 'Product'
[4] button 'Solutions'
[5] button 'Resources'
[6] button 'Open Source'
[7] button 'Enterprise'
[8] link 'Pricing'
[9] button 'Search or jump to...'
[10] link 'Sign in'
[11] link 'Sign up'
```

Figure 1: **Multimodal environment input.** Left: Screenshot of a rendered webpage with set-of-marks. Right: Accessibility tree snippet showing the hierarchical structure of interactive elements. This multimodal input enables agents to understand both visual and structural content for accurate interaction. The agent can find buttons both from the vision input and the text input.

- **Scroll:** Scroll a scrollable container or the entire page.
- **Return:** Navigate back to the previous page in browser history.
- **Answer:** Return the final answer.

C Additional Experiments

In this section, we present supplementary experimental results to further analyze the effectiveness of individual components in our framework.

C.1 Ablation Study: Pre-Exploration Strategy

To evaluate the impact of the pre-exploration strategy, we conduct an ablation experiment where this phase is removed. In this variant, the task generator only receives the base URL of the website along with trajectories collected from previous training iterations. Without the structured understanding provided by pre-exploration, the generator lacks information about the page layout and interaction flow.

As shown in Table 1, removing pre-exploration leads to a noticeable degradation in performance. The task generator in this setting frequently proposes irrelevant or infeasible tasks due to its limited knowledge of website structure, demonstrating the importance of pre-exploration in grounding curriculum design.

Model	Map	Reddit	OSS	Gitlab	CMS	Avg
Qwen 2.5VL-7B SAGE w/o pre	20.3	12.4	26.6	9.3	8.8	15.3
Qwen 2.5VL-7B SAGE	28.1	29.7	43.2	18.9	15.6	27.1
Qwen 2.5VL-32B SAGE w/o pre	40.3	24.1	32.0	18.3	20.8	27.1
Qwen 2.5VL-32B SAGE	42.3	43.7	46.1	31.4	35.2	39.7

Table 1: **Ablation study on the pre-exploration strategy.** We evaluate the impact of removing the pre-exploration phase, where the task generator receives no structural information about the website beforehand. Without this step, the task generator becomes less grounded and often infeasible, resulting in a significant performance drop across all categories. These results demonstrate the importance of pre-exploration for understanding environment structure and supporting meaningful task generation.

C.2 Ablation Study: Evaluator

We conduct an ablation study to assess the importance of the training-based evaluator in our framework. Specifically, we replace the learned evaluator with alternative models—Claude 3.5 Sonnet and Qwen2.5VL-32B—and use their raw outputs to supervise agent training. As shown in Table 2, our training-based evaluator leads to better task success rates than both baselines, *despite being based*

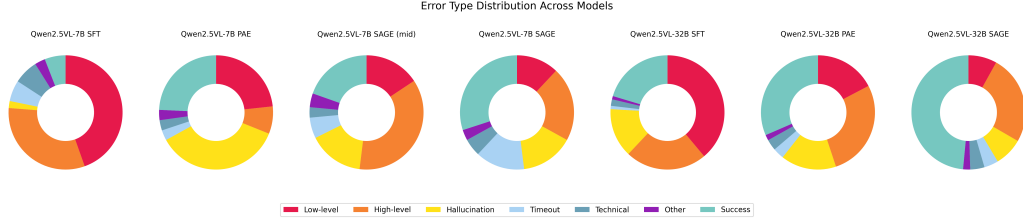


Figure 2: **Error type distribution across models.** Each donut chart illustrates the proportions of different error types and success cases on WebVoyager tasks, annotated for each model variant. SAGE effectively reduces both low-level and high-level error rates across model sizes. For Qwen2.5VL-7B, we also include an intermediate checkpoint (“mid”) which shows a significant reduction in low-level errors early in training. As training progresses, high-level reasoning errors also diminish, suggesting that *SAGE enables agents to first master simple skills before tackling harder reasoning challenges*. On Qwen2.5VL-32B, SAGE achieves the lowest error rates and highest success rate, demonstrating its robustness at scale.

98 on a smaller open-source model. This highlights the benefit of targeted evaluator fine-tuning on
 99 trajectory outcomes, which enhances the quality and reliability of training supervision.

100 C.3 Error Analysis

101 To better understand how SAGE facilitates skill acquisition from easy to hard tasks, we conduct a
 102 detailed error analysis on the WebVoyager [7] benchmark. We randomly sample 200 tasks from the
 103 test set and evaluate multiple models, and then manually annotate the resulting trajectories based on
 104 failure modes.

105 Following the error types introduced in prior work [6], we categorize each failure into six distinct
 106 types:

- 107 • **Low-level execution errors:** The agent has a plausible plan but fails to interact correctly
 108 with the interface, such as clicking the wrong element or failing to navigate to the intended
 109 page.
- 110 • **High-level reasoning or planning errors:** The agent fails to formulate an effective strategy
 111 or to reason correctly through the interface in order to solve the task.
- 112 • **Visual hallucinations:** The agent fabricates answers that are not supported by the visible
 113 content—e.g., referencing information that is not present or misidentifying page content.
- 114 • **Timeouts:** The agent is on the correct path but exceeds the maximum number of interaction
 115 steps before completing the task.
- 116 • **Technical issues:** Errors resulting from environment-level problems such as broken links,
 117 loading failures, or website outages, not attributable to the agent itself.
- 118 • **Other:** Miscellaneous cases that do not fall into the above categories, including fundamen-
 119 tally infeasible tasks.

120 We present a detailed error breakdown in
 121 Figure 2 to examine how SAGE facilitates
 122 progressive skill acquisition across training
 123 stages and model scales. Across all mod-
 124 els, we find that SAGE significantly im-
 125 proves task performance by reducing two
 126 major sources of failure: low-level execu-
 127 tion errors and high-level planning errors.
 128 Notably, in the Qwen2.5VL-7B model, we
 129 analyze an intermediate checkpoint after
 130 5 training iterations and observe a marked
 131 decrease in low-level failures—indicating
 132 that the agent learns precise actions early
 133 on in the training process, largely through
 134 practicing easier tasks. As training progresses, the proportion of high-level planning errors decreases

Model	Map	Reddit	OSS	Gitlab	CMS	Avg
Qwen 2.5VL-7B SAGE Claude	24.3	25.3	38.4	16.3	14.2	23.0
Qwen 2.5VL-7B SAGE Qwen	23.2	26.3	36.2	16.0	14.6	22.7
Qwen 2.5VL-7B SAGE	28.1	29.7	43.2	18.9	15.6	27.1
Qwen 2.5VL-32B SAGE Claude	41.7	40.8	41.3	29.6	32.4	36.4
Qwen 2.5VL-32B SAGE Qwen	39.9	40.7	41.6	25.8	30.2	34.5
Qwen 2.5VL-32B SAGE	42.3	43.7	46.1	31.4	35.2	39.7

Table 2: **Ablation study on the evaluator module.** We compare our training-based evaluator against two alternatives: Claude 3.5 Sonnet and Qwen2.5VL-32B, both used as fixed evaluators without additional fine-tuning. Despite its smaller scale, our trained evaluator consistently yields higher task success rates, demonstrating the importance of domain-adapted supervision for reliable reward estimation.

Hyperparameter	Value
number of actions	10
maximum exploration step	100
number of tasks pre-generated	10,000
number of tasks generated in iteration	2,000
number of iterations in MCTS	25
number of trajectories	2,048
actor update epochs per iteration	4
batch size	8
DPO β	0.45

Table 3: Hyperparameters used for SAGE.

as well, reflecting the agent’s transition to mastering more complex tasks. This learning trajectory aligns with the curriculum induced by our top-level exploration strategy. Furthermore, on the larger Qwen2.5VL-32B model, SAGE consistently reduces both types of errors and achieves the highest success rate among all variants, highlighting its scalability and effectiveness even with stronger backbones.

D Hyperparameters

We include the hyperparameters that we have used in Table 3.

E Prompts

We include all prompting templates used throughout our framework. In particular, Figures 4 and 5 show the prompt used during the *pre-exploration phase*, where the agent is instructed to explore and describe the structure and functionality of a given website starting from its homepage. Figures 6 and 7 provide the prompt template used in the *top-level exploration phase*, where the task generator proposes new instructional tasks based on previous trajectories and structural understanding of the environment. All prompts are designed to be general and domain-agnostic, ensuring compatibility across both open-source and proprietary models.

F A Case Study of SAGE

We provide a case study that illustrates how SAGE enables an agent to acquire complex skills through structured, hierarchical exploration. We examine a challenging task from the GitHub website: “Find the most recently updated Web Agent project and list its contributors.” As shown in Figure 3, prior to training, the agent consistently fails to initiate the correct actions, struggling even to identify how to search for relevant projects.

Through the *top-level exploration* module, SAGE’s self-guided task generator decomposes the difficult instruction into three simpler subtasks: (1) locating a Web Agent project, (2) identifying the most recently updated entry, and (3) retrieving its contributors. These tasks are short-horizon and require only a few well-defined interactions, allowing the agent to explore and acquire each skill individually via *low-level exploration*.

Once the agent demonstrates competence on the subtasks, the task generator reintroduces the original complex task, allowing the agent to practice solving it end-to-end. Empowered by prior experience and learned sub-skills, the agent successfully completes the original instruction—demonstrating the effectiveness of SAGE in enabling curriculum-driven skill composition and long-horizon reasoning.

G Limitations and Future Work

While SAGE achieves strong performance across diverse web environments, it is not without limitations. First, our framework assumes access to a stable and scriptable browser environment, which

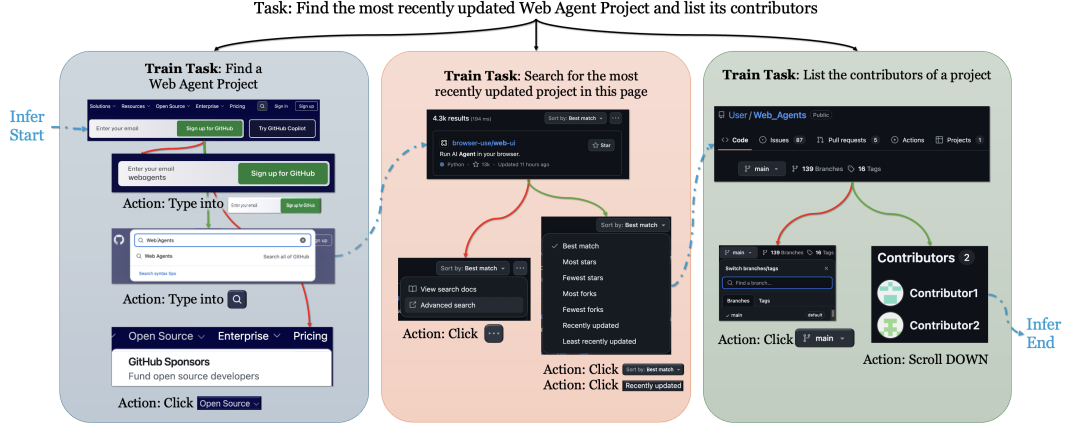


Figure 3: An example showing how SAGE facilitates skill acquisition through hierarchical exploration. The agent initially fails to complete the task “Find the most recently updated Web Agent project and list its contributors.” In subsequent iterations, the task generator offers simpler subtasks with the *top-level exploration*. The agent learns to solve each subtask individually through the *low-level exploration*, ultimately enabling it to succeed on the original, more complex task. The blue line shows a trajectory the agent generated during the inference phase.

may not hold in the presence of real-world challenges such as CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart), login requirements, or dynamically changing web content. These factors can degrade the quality of exploration and evaluation and may require additional engineering to handle robustly. Second, although our method is designed for research and sandboxed environments, applying it directly to live websites may raise privacy concerns, as agents could unintentionally access or leak user-sensitive information during open-ended exploration. Ensuring privacy-safe deployment remains an important direction for future work. Finally, while our method demonstrates strong generalization across web-based domains, future work will explore its application to broader human-computer interaction settings, including desktop and mobile environments, to extend its utility beyond web navigation.

H Broader Impact

This work aims to advance the development of generalist web agents that can interact with real-world websites using vision-language foundation models. Such agents have the potential to automate a broad range of digital tasks—ranging from customer support to online form completion and data retrieval—improving efficiency and accessibility for users. By removing the reliance on handcrafted task datasets and enabling agents to self-improve through hierarchical exploration, our method also lowers the barrier to developing robust web automation systems in low-resource or rapidly changing environments.

However, the deployment of web agents trained with real-world interfaces introduces several societal considerations. First, these agents might unintentionally access or process sensitive personal information, especially on open websites. Without explicit safeguards, this could raise concerns around privacy, data misuse. Second, autonomous agents acting on behalf of users or organizations must be designed with care to avoid unintended actions, biases, or misrepresentations, especially in domains like finance, healthcare, or content moderation.

Task Generator Prompt for SAGE (Part A)

```
{ "web_name": "Amazon",  
  "web": "https://www.amazon.com/",  
  "structure_info": ["Homepage includes sections such as 'Electronics', 'Books', and 'Deals'.", "Product pages contain price, reviews, delivery info, and variants.", "Search results are filterable by price, rating, and brand."],  
  "previous_trajectories": [  
    { "instruction": "Search for headphones under $50 with at least 4 stars.", "success rate": 0.32 },  
    { "instruction": "Compare features between AirPods Pro and Galaxy Buds.", "success rate": 0.00 },  
    { "instruction": "Find a gift card worth exactly $100.", "success rate": 0.90 }  
  ] }
```

We are training a web agent that learns to complete real-world tasks by interacting with web interfaces. Your job is to generate high-quality, instructional tasks that help the agent improve over time. The agent has already explored this website and attempted previous tasks. Based on this prior experience and structural understanding of the site, you must generate a new batch of 25 training instructions.

Step 1: Reasoning Instructions

Before listing your output, write your reasoning on how you adapted your tasks. Your strategy must consider the following:

- **Simplify low-success tasks:** For instructions with low success (e.g., below 40%), generate 1–2 simpler variants by relaxing filters or breaking down subtasks.
- **Harder from high-success tasks:** For instructions with high success (e.g., above 80%), generate harder variants by adding filters or composing with other subtasks.

Figure 4: Task Generator Prompt Part A.

Task Generator Prompt for SAGE (Part B)

- **Structure-aware task design:** Use the provided `structure_info` to ensure realism.
- **Curriculum balance:** Ensure a mix of easy, medium, and hard tasks.

Step 2: Output Format

- Output exactly **25 instructions** in **JSONL format**.
- Each line must be a valid JSON object with a single field `"instruction"`.

Rules and Constraints

- Do not require login, user data, or current time.
- Ensure task is verifiable via screenshots.
- Completion should require 2–8 steps.

In-Context Examples:

- *Simplified:* { "instruction": "Search for headphones under \$50." }
- *Harder:* { "instruction": "Compare Galaxy Buds and AirPods, then find a cheaper rated alternative." }

Figure 5: Task Generator Prompt Part B.

Pre-Exploration Prompt (Part A)

You are an intelligent web browsing agent designed to explore and understand new websites before learning specific tasks. In this phase, you will only receive the homepage of a new website, and your goal is to discover and record meaningful pages and their functions. This phase is unguided—you are not solving any tasks.

In each step:

- You will observe a screenshot, accessibility tree, and numerical element labels.
- Recall visited pages to avoid redundant navigation.
- Propose up to **10 distinct actions** that could lead to semantically different states.
- Record what the page reveals (e.g., filters, product listings, FAQs).

Figure 6: Pre-Exploration Prompt Part A: Instructions for agent behavior and high-level goals.

Pre-Exploration Prompt (Part B)

Output Format per step:

- **Thought:** Reason about the state and exploration plan.
- **Top-10 Actions:** Propose 10 diverse actions (e.g., Click [3], Scroll WINDOW; down).
- **Summary:** Describe the page’s purpose or findings.

Exploration Guidelines:

- Avoid login, signup, or non-navigational elements.
- Maximize diversity—explore filters, categories, products, help pages, etc.
- Use accessibility tree to guide semantic understanding.
- Avoid repeated or redundant actions.

Example Output:

Thought: The homepage includes categories and a search bar. I haven’t explored “Books” or filtered search results yet.

Top-k Actions:

- Click [3]
- Click [4]
- Scroll WINDOW; down
- Type [10]; "usb-c hub"
- ...

Summary: This is the homepage with links to major categories and a search bar. I begin mapping product pages and filter paths.

Figure 7: Pre-Exploration Prompt Part B: Format, examples, and behavior guidelines for site exploration.

References

- [1] Dan Zhang, Sining Zhou, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*, 2024. 1
- [2] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models, 2023. 1
- [3] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, Peter Auer, and András Antos. Upper-confidence-bound algorithms for active learning in multi-armed bandits, 2015. URL <https://arxiv.org/abs/1507.04523>. 1
- [4] Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf. 1, 2
- [5] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL <https://arxiv.org/abs/2407.10671>. 2
- [6] Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levione, and Erran Li. Proposer-Agent-Evaluator (PAE): Autonomous skill discovery for foundation model internet agents. In *ICML*, 2025. URL <https://arxiv.org/abs/2412.13194>. 2, 4
- [7] 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, 2024. URL <https://arxiv.org/abs/2401.13919>. 4