Research Article

ScribeAgent: Towards Specialized Web Agents Using Production-Scale Workflow Data

Junhong Shen[1], Atishay Jain[2], Zedian Xiao[2], Ishan Amlekar[2], Mouad Hadji[2], Aaron Podolny[2], Ameet Talwalkar[1]

1. Machine Learning Department, Carnegie Mellon University, United States; 2. AI/ML, Scribe, United States

Large Language Model (LLM) agents are rapidly improving to handle increasingly complex web-based tasks. Most of these agents rely on general-purpose, proprietary models like GPT-4 and focus on designing better prompts to improve their planning abilities. However, general-purpose LLMs are not specifically trained to understand specialized web contexts such as HTML, and they often struggle with long-horizon planning. We explore an alternative approach that fine-tunes open-source LLMs using production-scale workflow data collected from over 250 domains corresponding to 6 billion tokens. This simple yet effective approach shows substantial gains over prompting-based agents on existing benchmarks—ScribeAgent achieves state-of-the-art direct generation performance on Mind2Web and improves the task success rate by 7.3% over the previous best text-only web agents on WebArena. We further perform detailed ablation studies on various fine-tuning design choices and provide insights into LLM selection, training recipes, context window optimization, and effect of dataset sizes.

Junhong Shen led the research, while Atishay Jain and Zedian Xiao led the engineering efforts.

Corresponding author: Junhong Shen, junhongs@cs.cmu.edu

1. Introduction

Large language model (LLM) agents have advanced significantly in web navigation. They can carry out user-specified tasks in multiple steps by reasoning on their own what actions to take and what external resources to interface with. Recent studies[1][2][3][4] have shown that, with better planning and exploration strategies, LLM agents can independently solve various web tasks ranging from simple

navigation, such as locating a specific Wikipedia page, to more complex operations, such as booking flights or restaurants.

Despite these improvements, the performance of existing web agents on research benchmarks remains significantly below human levels[5][6][7]. One possible reason is their dependence on *general-purpose* LLMs. Indeed, all top-performing agents like WebPilot[3], AWM[8], and SteP[9] rely on prompting proprietary models like GPT-4[10]. These general-purpose LLMs are not optimized for interpreting web contexts such as HTML or accessibility trees; their pretraining and alignment processes do not address navigation-related challenges; and their proprietary nature presents a major obstacle in adapting them to web environments via continual training.

In this work, we explore an alternative approach by fine-tuning open-source LLMs with a large set of real-world workflow data[1] to develop *specialized* web agents (Figure 1). Through extensive experiments, we show that this approach not only boosts the web understanding and planning abilities of LLMs, achieving state-of-the-art results on various benchmarks, but also allows us to develop agent models significantly smaller than proprietary LLMs, reducing the serving costs. Our empirical results suggest that large-scale, high-quality, real-world data can be essential to agent development.

Specifically, we collect a set of proprietary workflow data representing action sequences executed by real users in real web environments through Scribe. This dataset encompasses a large and diverse spectrum of websites (over 250 domains and 10,000 subdomains), task objectives, task difficulty, and task length. Each step in the workflow features not only the raw HTML-DOM of the website but also a comprehensive documentation of the action, including action description in natural language, mouse or keyboard operation, and the CSS selector of the target HTML element. We reformat the data into a next-step prediction formulation and fine-tune a set of open-source LLMs via the parameter-efficient LoRA[11]. After preprocessing and reformatting, our training dataset contains more than 6 billion tokens.

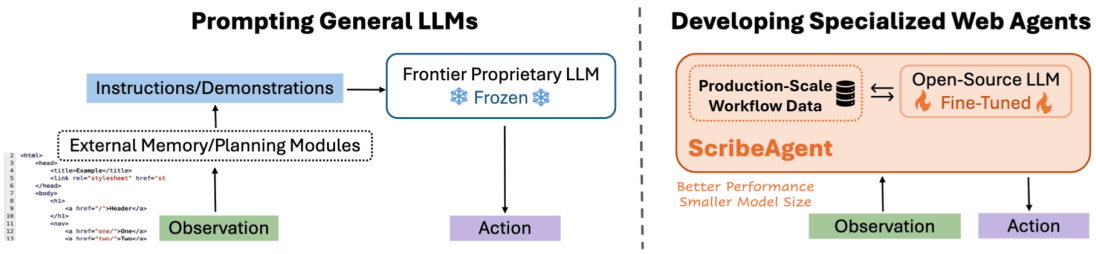

Figure 1. Left: Most existing LLM web agents are built on top of general-purpose, proprietary models like GPT-4 and rely heavily on prompt engineering. Their performance is enhanced by leveraging external planning, reasoning, and memory modules. **Right:** We explore an alternative way to develop specialized agents by fine-tuning open-source LLMs using a large set of high-quality, real-world workflow data. This significantly boosts agent's navigation and planning capacity, enabling it to outperform proprietary models with a smaller LLM backbone, thereby reducing serving costs.

With access to this production-scale dataset, we develop ScribeAgent, the first family of specialized, single-stage LLM agents capable of directly generating the next step based on the website's DOM and action history. This is in contrast with previous fine-tuned agents that require multiple stages to produce an action, e.g., first narrowing down to a set of target element candidates and then selecting one from the candidates[5]. To evaluate the capacity and generalization ability of ScribeAgent, we test it on public benchmarks without any further training. ScribeAgent significantly outperforms existing GPT-4-based and multi-stage agents. Notably, the 32B-parameter ScribeAgent-Large achieves state-of-the-art direct generation performance on Mind2Web[5], with step success rate surpassing the baselines by 5-10% across all test sets. On the end-to-end task execution benchmark WebArena[6], our 7B ScribeAgent-Small improves the previous best task success rate from 45.7% to 51.3%. ScribeAgent-Large achieves an even better task success rate of 53%, marking the highest performance among text-only LLM agents.

Beyond the empirical results, our work also provides several insights valuable for future web agent research: (1) we show that direct fine-tuning on highly structured inputs (HTML-DOM) is feasible and can improve the agent's ability in identifying the correct target; (2) we identify an effective HTML preprocessing strategy that balances between preserving essential information and minimizing context length; (3) we provide a thorough analysis on various design choices in fine-tuning, such as LLM backbone and context window selection; (4) we illustrate how fine-tuning improves agent performance as dataset size increases.

doi.org/10.32388/8VOGoO

Our work highlights the potential of building web agents via specialized fine-tuning with production-scale data. This approach not only improves agents' capabilities relative to prompt-engineered alternatives, but also reduces inference costs due to the smaller sizes of open-source LLMs. While our work focuses on studying the effect of fine-tuning, ScribeAgent can be extended to leverage more sophisticated search or memory modules[12][8], combined with existing planning frameworks[13][14][15], or integrated into multi-modal web agent systems as the text model[16]. We view ScribeAgent as an important step towards developing AI assistants and fully automated agents for real-world web applications.

2. Related Work

Prompting-based agent frameworks.

The majority of web agent works reuse existing LLMs and propose different prompting strategies to improve action prediction. One line of research focuses on exploiting previous experience via self-feedback[17] or in-context demonstrations[18][1][8][19][20]. A separate line of work centers around encouraging exploration by including external evaluators[21], using synthesized instructions[22], or applying more sophisticated search algorithms like stack[9], best-first tree search[12], or Monte Carlo Tree Search[3]. Despite the research efforts, these prompting methods rely heavily on the quality of the LLM used. Open-source models such as LLaMA[23], Code LLaMA[24], and Flan-T5[25] generally underperform proprietary models like GPT-4. However, fine-tuning proprietary LLMs can often be costly and challenging, as it is restricted to being done through APIs. This implies an opportunity for enhancing open-source LLMs to match or outperform proprietary agents.

Fine-tuning-based web agents.

Compared to developing better reasoning and planning frameworks, comparatively less attention has been given to optimizing the LLMs themselves to better handle web environments. Due to the difficulty of directly generating a single target element from the raw HTML, which often contains thousands of elements, existing work mostly focuses on multi-stage prediction. MindAct[5] proposes a two-stage pipeline that first uses a small LM to filter the web elements and then uses a more powerful LM to select from the filtered elements in a multi-choice question answering format. Both LMs can be fine-tuned using the Mind2Web dataset. WebAgent[26] uses HTML-5 to first process the HTML and then fine-tunes a 540B Flan-UPalm to generate code for controlling web pages. More recently, AutoWebGLM[2] trains a

single ChatGLM3 6B[27] using a combination of curriculum learning, reinforcement learning, and rejection sampling fine-tuning. NNetnav[28] leverages synthetic demonstrations collected by zero-shot LLM exploration and hindsight relabeling to fine-tune open-source models into web agents. Despite the complicated training and inference procedures, these methods often underperform agents that prompt GPT-4. In contrast, our work shows that given sufficient high-quality workflow data, fine-tuning a single LLM can achieve strong performance. We note that the newly released OpenAI o1[29] can be viewed as a specialized agent with a complicated planning framework. Nonetheless, we show in Section 4.1 that ScribeAgent outperforms o1-preview by a large margin on our proprietary dataset. Moreover, while none of the training details for o1 have been released, our work provides valuable insights into data preprocessing and fine-tuning.

Beyond the aforementioned work, there is an earlier line of research that fine-tunes LLMs for HTML inputs[30][31][32]. However, their primary application is question-answering tasks, such as answering "could sunflowers really track the sun across the sky", and they cannot be used to generate a sequence of actions based solely on the user objective.

Lastly, we note that an emerging line of research has committed to developing multi-modal web agents that use screenshots along with HTML observations. Examples include CogAgent[33], SeeClick[34], WebGUM[35], WebVoyager[36], and AWA 1.5[37]. However, our current version of ScribeAgent focuses exclusively on text-based inputs due to the lack of effective visual preprocessing schemes. Thus, we do not compare with the aforementioned multi-modal methods in our experiments and leave developing multi-modal ScribeAgent as future work.

3. Method

In this section, we first overview the general setup of solving web-based tasks with LLM agents. Then, we detail our proposed method to develop specialized agents from open-source LLMs.

3.1. General Setup

We consider solving a web-based task as a sequential decision-making process guided by a high-level objective. For each task, the user first specifies an objective and a starting web page. Then, at every step, the agent outputs an action based on the task objective, the current web page, and the history. Formally, denote the user objective as q. The web environment is governed by a transition function T that can evolve over time. The agent is instantiated by a language model L. At each time step t, the agent observes

o_t produced by the environment state s_t and observes the history $h_t = H(o_{1:t-1}, a_{1:t-1})$. It outputs an action $a_t = L(q, o_t, h_t)$, which is executed in the environment, and the state changes correspondingly $s_{t+1} = T(s_t, a_t)$. This iterative process stops when the agent issues a stop signal, or a task termination condition is met, such as we have reached a predefined maximum number of steps.

For single-modal, text-only agents, the observation o_t typically consists of the website's URL, the HTML-DOM (Object Model for HTML, which defines HTML elements and their properties, methods, and events), and potentially the accessibility tree (a representation that can be understood by assistive technologies like screen readers). Since the raw HTML-DOM is often long and contains redundant structural information, most methods employ preprocessing and pruning strategies, which could be as simple as retaining a fixed set of HTML tags and attributes or more complex ones like LLM-based element ranking and filtering[5].

The action a_t emulates the keyboard and mouse operations available on web pages. The most general action space in existing work consists of element operations, such as clicking, typing, and key combination pressing; tab actions, such as opening, closing, and switching between tabs; navigation actions, such as going forward and backward in the browsing history[6].

As discussed earlier, previous web agent work focuses on presenting useful demonstrations through h_t or iteratively revising a_t to improve the quality of the predicted next step. In contrast, we explore whether we can improve the model L itself by learning from a vast amount of data and incorporating more information into o_t, such as the natural language description and HTML representation of a action. We detail our approach in the next section.

3.2. ScribeAgent: Specializing Web Agents Through Fine-Tuning

3.2.1. Collecting Production-Scale Data

We collected a large set of real-world, user-annotated, proprietary data through Scribe, a software that streamlines the creation of step-by-step guides for web-based tasks. Scribe allows users to record their interactions with the web through a browser extension and converts the interactions into well-annotated instructions, which can be then customized to specific business needs. The collected dataset consists of everyday workflows in common web application domains, encompassing customer relationship management (CRM) tools like HubSpot and Salesforce; productivity tools like Notion and Calendley; social platforms like Facebook and LinkedIn; shopping sites like Amazon and Shopify; and many others.

Each workflow features a high-level user objective and a step-by-step documentation of the action sequence to achieve the task. The objective spans a wide range of topics, such as "add a user in a Salesforce" or "invite someone to manage Facebook ad accounts". Each step contains the following information: the current web page's URL, raw HTML-DOM, a natural language description of the action performed, the type of action, and the autogenerated CSS selector to identify the action target. There are three types of actions in the dataset:

- *mouse_click_action*: click at an element
- *keyboard_sequence_action*: type a sequence of characters to an element
- *keyboard_combination_action*: press a set of keys together (e.g., hotkey like ctrl+c)

Note that there is no scroll actions in our action space since all elements are already fully accessible in the captured data. This is because we capture the full DOM from a system perspective, which inherently includes the entire webpage as observed from the backend. This method differs from user-centric data collection, where only the elements within the visible browser viewport are captured.

To ensure the quality of the data, we remove workflows with invalid selectors, i.e., the selector cannot be used to locate a target element in the DOM. We also remove non-English workflows to reduce dataset complexity and enable us to explore English-only LLMs like Mistral 7B[38]. The resulting dataset is at production scale: using raw data collected over a two-month period, we are able to extract workflow data from more than 250 domains and 10,000 subdomains with an average task length of 11 steps, which correspond to about 6 billion training tokens. This large-scale, high-quality, real-world dataset is unmatched in prior web agent research.

Since this dataset is collected from real users and might contain sensitive and confidential information, it will not be released to the public to protect user privacy. The dataset is solely for research purposes and has been anonymized to prevent the identification of any individual.

3.2.2. Preprocessing

For ScribeAgent, we consider an observation space consisting mainly of the URL and HTML-DOM. Specifically, HTML-DOM provides agents with all structural and content information about the web page that are essential for generating the next step and long-term planning. For instance, while a drop-down menu may not be visible on the website before expansion, the agent can detect the menu items from the DOM and determine whether to click and expand it. We do not use accessibility tree to develop

ScribeAgent because it may lose information about the HTML elements, such as the drop-down items, and does not generalize across different browsers and devices.

Given our observation space, a subsequent problem is that the DOM can be quite long and exceed the context window of prevailing open-source LLMs. To reduce the DOM sizes, we propose a pruning algorithm that maintains the essential structure and content while eliminating redundant or disruptive elements that could hinder the LLM's understanding. Specifically, we first use the BeautifulSoup library[39] to remove non-essential components such as metadata, CSS, and JavaScript. Then, we utilize a tag-attribute white list to retain useful tag level information like retaining interactive elements. Since some attribute values can contain random character sequences that do not provide useful information, we propose a novel detection method that removes the attributes with character-to-token-ratio smaller than 2, i.e., $\frac{len(s)}{len(tokenizer(s))} < 2$, where s denotes the value string. Intuitively, if each character in a string is encoded using a separate token, it is highly likely that the string is not semantically meaningful. Lastly, we remove the comments and extra whitespaces to clean up the DOM. After pruning, we assign each tag in the HTML with a unique ID by traversing the HTML tree from bottom to top. More details about preprocessing and analysis on the tokenizer-pruning method can be found in Appendix A.1.

We restrict the action space of ScribeAgent to the three types of operations specified in Section 3.2.1. To preprocess the action sequences, we rewrite each step into five lines as follows:

```
1.
Description: Click the “Menu” button to browse all food options
Action: mouse_click_action
Node: 832
Target: <svg class=“open-hamburger-icon” node=“832” role=“img”>
```

The first line represents the current time step. The second line is the natural language description of the action, which can help LLMs to learn about the rationale behind applying a specific action. The third line is one of the three operations in the action space. The fourth line is the unique ID assigned to the target element. The last line details the HTML tag and attributes, which can be directly obtained from the processed DOM.

For the history, we consider only previous actions, omitting previous observations due to the extensive length of the DOMs. That is, $h_t = a_{1:t-1}$. Therefore, at each step, ScribeAgent will be given the task objective, URL, HTML-DOM, and all previous actions in the aforementioned five-line format. Its goal is to output the next action $a_t = L(q, o_t, a_{1:t-1})$ that helps complete the task. In Appendix A.2, we provide an example of a full workflow.

Lastly, during our inspection, we find that 10% of the action descriptions in the dataset are not informative (e.g., "click here"). In these cases, we use GPT-4o[40] to regenerate the action description from screenshots. We provide the prompt as well as examples of the regenerated action descriptions in Appendix A.3.1.

3.2.3. Fine-Tuning with LoRA

After preprocessing, we divide the dataset into two splits. The test set comprises of 1200 workflows with diverse objectives and domains. We use the remaining workflows as the training data to adapt LLMs via standard supervised fine-tuning. Note that for each fine-tuning example, the label is a single next-step instead of all remaining steps needed to complete the task. The agent is trained to generate all information in the five-line format described above, including the natural language description.

To reduce fine-tuning cost, we opt for the parameter efficient method LoRA[11] instead of full fine-tuning, since we have not observed significant performance gain by updating more parameters. We also follow previous work[41][42] to fine-tune the layernorms in addition to the LoRA adapters. Based on empirical observations, we set the fine-tuning epoch to 2, effective batch size to 32, LoRA rank to 64 and α to 128. We use a cosine scheduler with 30 warmup steps and a learning rate of 1e-4.

3.2.4. Exploring the Design Space

There are multiple design choices for ScribeAgent that might affect the prediction accuracy, fine-tuning cost, and inference latency. We focus on three aspects and perform detailed ablation studies to find out the optimal modeling and training configurations.

Model	# Params	Before Fine-Tuning		After Fine-Tuning	
		EM (%)	Calibrated EM (%)	EM (%)	Calibrated EM (%)
Mistral-7B-Instruct-v0.3	7B	3.89	5.13	19.92	26.31
Qwen2-7B-Instruct	7B	6.06	7.92	**29.34**	**38.72**
Llama-3.1-Instruct-8B	8B	1.42	1.88	28.34	37.42
Qwen2.5-14B-Instruct	14B	8.79	11.60	**31.76**	**41.89**
Codestral-22B-v0.1	22B	4.53	6.08	31.11	41.25
Qwen2.5-32B-Instruct	32B	10.02	13.21	**32.98**	**43.51**
Mixtral-8x7B-Instruct-v0.1	56B-A12B	7.35	9.82	28.38	37.49
Qwen2-57B-A14-Instruct	57B-A14B	5.72	7.51	31.02	40.10

Table 1. Performance of different LLMs fine-tuned on 1B workflow tokens on the test split of our proprietary dataset. We highlight the best results for small/medium/large models. EM is short for Exact Match.

Pretrained LLM Selection.

Intuitively, the quality of a fine-tuned web agent should be relevant to the quality of the pretained LLM. We identify two axes that are crucial to performance—model architecture and model size—and explore seven open-source LLMs spanning these axes: Llama 3.1 8B[23], Mistral 7B[38], Mixtral 8x7B[43], Qwen2 7B[44], Qwen2 57B[44], Qwen2.5 14B[45], Qwen2.5 32B[45], and Codestral 22B[46]. We fine-tune these models with 1 billion training tokens and evaluate their performance on the test split of the dataset we collected.

Given that many of the evaluated LLMs have a maximum context window of approximately 32K, and the processed DOM can exceed this limit, we divide the DOM sequentially into chunks that fit into the context window. For fine-tuning, we use the chunk containing the correct target, but for evaluation, we use the last chunk since the target's location is not known beforehand. When evaluating at a 32K context window, 25% of the test data do not have the correct target tag in the DOM, i.e., these tasks are unachievable. Thus, we compute two metrics for evaluation: (1) exact match (EM) measures the model's ability to select

exactly the same HTML tag as the ground truth; (2) calibrated exact match (Calibrated EM, or CEM) measures the percentage of correct target predictions where the target tag was present in the truncated HTML DOM, i.e., it is EM on the set of examples where the observation contains sufficient information to complete the task. As we scale the context window, these two metrics converge.

We report the performance of different LLMs before and after fine-tuning in Table 1. Notably, for all models, specialized fine-tuning drastically increases the prediction accuracy. Both before and after fine-tuning, the Qwen family demonstrates better EM and CEM across small, medium, and large models. We observe performance gains as model size increases. For example, the calibrated EM for Qwen2 57B is higher than its 7B counterpart. Qwen2.5 32B is also better than Qwen2.5 14B. Mixtral 8x7B outperforms Mistral 7B by a large margin as well. However, fine-tuning larger models is significantly more resource-intensive—while Qwen2 7B can be fine-tuned using 8 H100 GPUs in just one day, Qwen2 57B takes over a week using the same hardware configuration. Larger models also incur longer inference times and require multiple GPUs even at a 32K context length. To balance effectiveness and efficiency, we develop two versions of ScribeAgent using Qwen2 7B and Qwen2.5 32B, respectively. The Qwen2 7B model offers an optimal balance between prediction accuracy and the costs associated with fine-tuning and inference. Meanwhile, the Qwen2.5 32B model provides stronger performance when we have sufficient computational resources.

Model	Context	EM (%)	CEM (%)
Qwen2 7B	32K	29.34	38.72
Qwen2 7B	65K	31.42	36.22
Qwen2.5 14B	32K	31.76	41.89
Qwen2.5 14B	65K	33.96	39.15
Qwen2.5 32B	32K	32.98	43.51
Qwen2.5 32B	65K	36.16	41.69

Table 2. Ablations on context window length.

doi.org/10.32388/8VOGoO

Context Window Length.

We evaluate the models with 65K context window to add additional context and increase the rate of solvable tasks (Table 2). On both Qwen2 and Qwen2.5, scaling the context window from 32K to 65K leads to approximately 2% performance boost for Exact Match but approximately 2.5% performance drop for Calibrated Exact Match. We hypothesize that this performance degradation might be due to rotary position embedding[47] and the fact that it becomes harder to pick the correct target given twice as many options to choose from. Besides, we note that using 65K context window increases the inference time by approximately four times in practice.

# Train Tokens	EM (%)	CEM (%)
1B	29.34	38.72
3B	32.65	43.06
6B	34.96	46.42

Table 3. Ablations on dataset size. All settings are trained and evaluated with Qwen2-7B-Instruct and 32K context window.

Dataset Size.

Lastly, we are interested in understanding the effect of fine-tuning dataset size on the agent's performance. To this end, we sample our training set without replacement into smaller subsets and fine-tune Qwen2 7B on them. Results are shown in Table 3. Plotting on a log-linear scale, we observe that there is a roughly 2% performance boost when we double our dataset size.

To sum up, using our proprietary dataset, we study the effect of LLM backbone, context window, and dataset size on the agent performance. We find that (1) scaling parameter count generally improves prediction quality, but the latency and training time of large LLMs can be prohibitive; (2) using longer context window boosts model performance on EM but increases the inference time significantly; (3) training with more tokens is helpful. Based on these insights, we develop two versions of ScribeAgent: ScribeAgent-Small, based on Qwen2 7B, and ScribeAgent-Large, based on Qwen2.5 32B. Both versions are fine-tuned on the full 6B-token dataset at a 32K context window. While ScribeAgent-Large demonstrates

better performance in both internal and external evaluation,s the 7B ScribeAgent-Small is cheaper to serve at inference time, particularly when compared to large-scale proprietary models.

4. Results

We evaluate ScribeAgent on three web datasets. We first consider the next-step prediction setting, where performance is evaluated only on a single next step. We show that ScribeAgent not only outperforms various general-purpose LLMs on our proprietary dataset but also achieves state-of-the-art on the public benchmark Mind2Web[5]. Then, we move to the end-to-end task completion benchmark WebArena[6] and show that ScribeAgent augmented with GPT-4o achieves top performance among all existing agent systems. Note that we do not perform any task-specific adaptation for Mind2Web and WebArena, even when additional training data is available. This allows us to evaluate the generalization ability of ScribeAgent. Our focus is on ensuring that ScribeAgent remains versatile and robust across diverse settings, rather than optimizing for specific benchmarks.

4.1. Proprietary Dataset

To study whether specialized fine-tuning is indeed beneficial, we first compare the performance of ScribeAgent with general-purpose baselines on our proprietary test data. We consider the non-fine-tuned Qwen2 7B, GPT-4o, and GPT-4o mini. We use in-context demonstrations to prompt them to generate actions in the same five-line format as defined in Section 3.2.2. All OpenAI baselines in this work follow the prompt in Appendix A.3.2.

Results on the full 1200 test workflows are shown in Table 4. First, we note that ScribeAgent significantly outperforms the proprietary GPT-4o and 4o mini. This shows the benefit of specialized fine-tuning over using general-purpose LLMs. Moreover, while the non-fine-tuned Qwen2 performs extremely poorly, fine-tuning with our dataset (ScribeAgent-Small) boosts its performance by nearly $6\times$, which highlights the importance of domain-specific data.

Model	EM (%)	CEM (%)
Qwen2 7B	6.28	8.20
GPT-4o mini	12.60	13.26
GPT-4o	15.24	16.02
ScribeAgent-Small	34.96	46.42
ScribeAgent-Large	**37.67**	**49.67**

Table 4. ScribeAgent *v.s.* non-fine-tuned, general-purpose baselines on the full test set with truncation at 32K tokens.

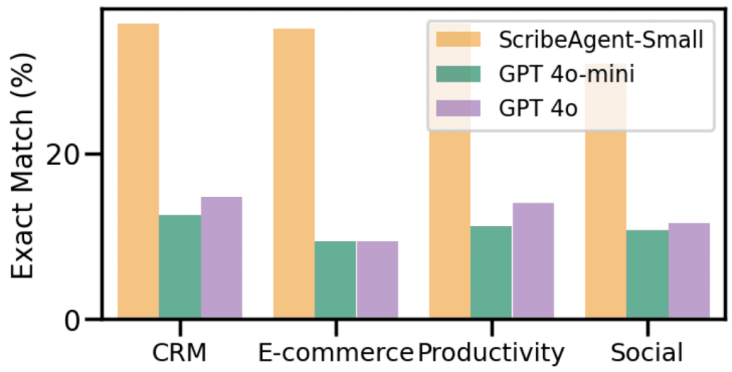

Figure 2. Exact Match (EM) comparison between ScribeAgent-Small and OpenAI models across different types of websites.

We also plot the Exact Match metric for four types of commonly seen domains, including customer relationship management (CRM) tools, E-commerce platforms, productivity tools, and social platforms (Figure 2). While our agent's performance varies by domain, with a 6% gap between the best performing domain and the worst performing one, we observe that ScribeAgent consistently outperforms the general-purpose baselines across all of them.

doi.org/10.32388/8VOGoO

Models	EM (%)	CEM (%)
o1-mini	17.40	18.32
o1-preview	22.60	23.79
GPT-4o mini	13.80	14.53
GPT-4o	16.60	17.96
ScribeAgent-Small	47.60	50.11
ScribeAgent-Large	**50.00**	**52.60**

Table 5. Comparing ScribeAgent with OpenAI baselines on 500 test samples. All models are evaluated at a 128K context window.

As we were wrapping up this work, OpenAI released o1[29], a series of specialized models for solving complex tasks in science, coding, and math. Since it has better planning ability, we also include it in our baselines. However, we did not run the o1 models on the full test set due to cost and API call limitations. Instead, we subsample 500 workflows and compare with ScribeAgent. As shown in Table 5, o1-preview performs the best among all general-purpose baselines. However, ScribeAgent still outperforms it by a wide margin, highlighting the importance of fine-tuning on real-world web navigation data. It is important to note that ScribeAgent-Small only has 7B parameters, while ScribeAgent-Large has 32B parameters, and neither model requires additional scaling during inference. In contrast, most proprietary baselines are typically larger in size and require more compute at inference. This makes ScribeAgent a better choice in terms of accuracy, latency, and cost.

4.2. Mind2Web

Mind2Web[5] is a text-based dataset for assessing the navigation ability of web agents across different tasks, websites, and domains. Each task features a human demonstration of a real-world workflow, such as booking a hotel on Airbnb. At each step, the agent is asked to predict a single action, consisting of an operation and the target element. Performance is measured by element accuracy, which checks if the correct target is selected; action F1 score, which measures operation correctness like text input; step

success rate, which evaluates whether both the target element and the operation are correct; and task success rate, indicating all steps are correct.

The original Mind2Web benchmark reports two sets of baselines: (1) multi-stage, multi-choice question-answering agents (i.e., the MindAct family) first use a pretrained element-ranking model to filter out 50 candidate elements from the full DOM and then use a separate LLM to recursively select an action from five candidates in a multi-choice question-answering (QA) fashion until one action is chosen; (2) a single-stage, generation-based agent (i.e., fine-tuned Flan-$T5_B$) directly generates the operation and the target based on the full DOM. The multi-stage baselines generally show higher metrics than direct generation models, as the element selection process effectively filters out noise, simplifying the task.

Beyond these baselines, we also consider recent published work such as AWM[8], Synapse[1], and fine-tuned HTML-T5[26]. AutoWebGLM[2] reports only step success rate among all four metrics. While the reported numbers are high, it uses a different and possibly more favorable evaluation procedure, so we do not compare against it. For both single-stage and multi-stage settings, we further categorize the baselines into those leveraging the Mind2Web training data for fine-tuning or in-context demonstrations and zero-shot methods. As stated earlier, we do not fine-tune our agents because our goal is to test the agent's out-of-distribution generalization abilities.

Method		Cross-Task				Cross-Website				Cross-Domain			
		EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR
Multi-Stage QA	*Uses M2W Train Set*												
	MindAct (Flan-$T5_B$)	43.6	76.8	41.0	4.0	32.1	67.6	29.5	1.7	33.9	67.3	31.6	1.6
	MindAct (Flan-$T5_L$)	53.4	75.7	50.3	7.1	39.2	67.1	35.3	1.1	39.7	67.2	37.3	2.7
	MindAct (Flan-$T5_{XL}$)	55.1	75.7	52.0	5.2	42.0	65.2	38.9	5.1	42.1	66.5	39.6	2.9
	AWM-offline (GPT-4)	50.6	57.3	45.1	4.8	41.4	46.2	33.7	2.3	36.4	41.6	32.6	0.7
	HTML-T5-XL	**60.6**	**81.7**	**57.8**	**10.3**	**47.6**	**71.9**	**42.9**	**5.6**	**50.2**	**74.9**	**48.3**	**5.1**
	Zero-Shot												
	MindAct (GPT-4)	41.6	**60.6**	36.2	2.0	35.8	51.1	30.1	2.0	21.6	52.8	18.6	1.0
	AWM-online (GPT-4)	50.0	56.4	43.6	**4.0**	42.1	45.1	33.9	1.6	40.9	46.3	35.5	**1.7**
	ScribeAgent Small (**Ours**)	42.6	50.1	39.7	0	44.9	50.1	41.6	0.6	44.1	51.4	41.4	0
	ScribeAgent Large (**Ours**)	**53.5**	52.9	**51.2**	0	**53.4**	**52.8**	**51.3**	**2.3**	**53.3**	**54.7**	**51.2**	0
Direct Generation	*Uses M2W Train Set*												
	Flan-$T5_B$	20.2	**52.0**	17.5	0	13.9	**44.7**	11.0	0	14.2	**44.7**	11.9	0.4
	Synapse (GPT-3.5)	**34.0**	-	**30.6**	**2.4**	**29.1**	-	**24.2**	**0.6**	**29.6**	-	**26.4**	**1.5**

doi.org/10.32388/8VOGoO

Method		Cross-Task				Cross-Website				Cross-Domain			
		EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR
	Zero-Shot												
	ScribeAgent Small (**Ours**)	28.6	50.1	26.8	0	27.6	50.1	25.6	0	32.0	51.4	29.9	0
	ScribeAgent Large (**Ours**)	**38.0**	**52.9**	**35.6**	0	**34.1**	**52.7**	**32.5**	0	**39.4**	**54.7**	**37.3**	0

Table 6. ScribeAgent achieves state-of-the-art zero-shot performance on Mind2Web. EA is short for element accuracy, AF_1 is short for action F_1, and SR is short for success rate. We note that the three categories are based on increasing level of domain generalization difficulty. However, since we do not train on Mind2Web data, our performance is similar across different test sets. Numbers are bolded for each method category.

We evaluate ScribeAgent on both multi-stage QA and direct generation. For the multi-stage setting, we first use the pretrained Mind2Web element-ranker to obtain the element ranking. Then, given the output of ScribeAgent, we traverse the sorted list of HTML elements from top to bottom, and stop when the agent's generated HTML element is a subchild of the element. We then replace ScribeAgent's prediction by the element. For direct generation, we simply compare the output of our agent to the ground truth action and target.

We report results following the evaluation procedure specified in the Mind2Web repository in Table 6. For the multi-stage setting, ScribeAgent-Large achieves the best overall zero-shot performance. Our element accuracy and step success rate metrics are also competitive with the best fine-tuned baseline, HTML-T5-XL, on cross-website and cross-domain tasks. However, our task success rates are not satisfactory, which is mainly due to the distribution differences between our training data and the Mind2Web data. Upon inspection, we find that the primary failure cases of our models are (1) predicting the subchild element of the ground truth instead of the ground truth; (2) predicting another element with identical function but is different from the ground truth; and (3) our agent tends to decompose type

actions into click followed by type actions. In many cases, we actually correctly predict the action description. These situations can be addressed by improving the evaluation procedure, which we discuss later in this section and in Appendix A.4.2.

As for direct generation, ScribeAgent-Large outperforms all existing baselines. Our step success rates are 2-3 times higher than those achieved by the fine-tuned Flan-T5 and show an improvement of 5-10% over Synapse, which utilizes GPT-3.5. We attribute ScribeAgent's strong performance to the diversity and high quality of the workflows in our dataset. Relatedly, the three test sets (Cross-Task, Cross-Website, Cross-Domain) are designed to capture different degrees of domain generalization difficulty. Since we do not train on Mind2Web data, the performance of ScribeAgent is similar across all three test sets.

As mentioned earlier, we observe several limitations in the Mind2Web evaluation that underestimate our agent's performance. First, the evaluation strictly compares element IDs, where only the outermost tags are labeled as ground truths, disregarding the subchildren. This leads to inaccuracies when our agent selects functionally identical but hierarchically deeper elements, which are marked as incorrect. Second, for direct generation, ScribeAgent might select a functionally identical element that is located in a different part of the website (e.g., consider clicking on the next page button vs. clicking on the page number). Lastly, there is a notable discrepancy between the distribution of our training data and the synthetic trajectories of Mind2Web. For instance, Mind2Web expects immediate type actions for `textarea` or `input` tags, whereas our model first clicks before typing.

To better reflect ScribeAgent's capabilities, we refine the evaluation method by relaxing the labels to include subchildren of the ground truths. For direct generation, we also introduce an element attribute matching step that compares not only the element IDs but also tag and text attributes. The results with refined evaluation is shown in Appendix A.4.2. We observe an average of 8% increase in task success rate and element accuracy for ScribeAgent, showing the need for enhancing evaluation of text-based benchmarks. It is worth noting again that results in Table 6 follow the original evaluation procedures to ensure fair comparison with established baselines.

While the Mind2Web results are promising, we note that another limitation of static, text-based benchmark is that the ground truth evaluation does not account for different action sequences that could reach the same goal. For instance, to book a flight, one can first enter the destination or first choose the departure date, but the ground truth trajectory only accounts for one possibility. Considering this, we also evaluated ScribeAgent on a dynamic benchmark WebArena[6].

4.3. End-to-End Task Execution on WebArena

WebArena[6] features 812 web navigation tasks across five domains: E-commerce (OneStopShop), social forums (Reddit), software development (GitLab), content management (CMS), and online map (OpenStreetMap). Unlike the static Mind2Web, it implements a dynamic environment for agents to interact with and allows for assessing the functional accuracy of action sequences. Since the WebArena environment is implemented to accept only target element IDs specified in the accessibility tree, whereas ScribeAgent operates on DOM and outputs targets in HTML, we employ GPT-4o to map between the different representations.

More generally, we tackle end-to-end task solving by developing a multi-agent system that utilizes GPT-4o to simulate user interactions with ScribeAgent. Our system contains four stages: (1) objective refinement: user adds details about the task objective to help complete the task; (2) action generation: based on the current website and action history, agent outputs an action suggestion; (3) action execution: user executes the suggested action, e.g., clicking a button; (4) completeness evaluation: user observes the current state and decides whether the task is completed.

We apply the above pipeline to solve the WebArena tasks. In stage 3, GPT-4o maps the agent's output in HTML to the accessibility tree format, which is then processed by the WebArena environment. To further improve performance, we allow ScribeAgent to generate multiple actions in stage 2 and select the one with the highest confidence using majority vote and GPT-4o analysis. More details about evaluating ScribeAgent on WebArena can be found in Appendix A.5.

Method	LLM	Total SR	Shopping	CMS	Reddit	GitLab	Maps
AutoWebGLM	ChatGLM3 6B	18.2	-	-	-	-	-
NNetnav	Llama 3 8B Instruct	7.2	7.4	4.2	0	0	28.5
AutoEval	GPT-4	20.2	25.5	18.1	25.4	28.6	31.9
BrowserGym	GPT-4	23.5	-	-	-	-	-
BrowserGym$_{axtree}$	GPT-4	15.0	17.2	14.8	20.2	19.0	25.5
SteP	GPT-4	33.0	37.0	24.0	59.0	32.0	30.0
AWM	GPT-4	35.5	30.8	29.1	50.9	31.8	43.3
Tree Search	GPT-4o	19.2	-	-	-	-	-
WebPilot	GPT-4o	37.2	36.9	24.7	65.1	39.4	33.9
Broswing+API Hybrid Agent	GPT-4o	35.8	25.7	41.2	28.3	44.4	45.9
AgentOccam with Judge	GPT-4-Turbo	45.7	43.3	**46.2**	67.0	38.9	52.3
Multi-Agent System (Ours)	ScribeAgent-Small + GPT-4o	51.3	**48.1**	35.5	70.2	58.8	51.9
	ScribeAgent-Large + GPT-4o	**53.0**	45.8	37.9	**73.7**	**59.7**	**56.3**

Table 7. Task success rates (SR) on WebArena and score breakdown on five web domains. ScribeAgent consistently outperforms considered text-only baselines, often improving the previous-best results by more than 5%. Note that we only test ScribeAgent-Small due to the large number of tasks in WebArena and the evaluation costs associated with the multi-agent system.

We compare our performance with all top-performing, text-only agents on the WebArena leaderboard. We note that we do not include Autonomous Web Agent (AWA) 1.5[37] as a baseline because it uses a

proprietary system to parse the HTML-DOM and web screenshots, rather than building from the WebArena GitHub. This allows them to have richer observations and bypass the accessibility tree action mapping step. In contrast, ScribeAgent is single-modal, text-only, and we stick to the original WebArena implementation. That said, AWA 1.5 employs more advanced reasoning, planning, and progress tracking techniques and is the only agent system with a higher average task success rate than ours.

The results are shown in Table 7. Compared with existing text-only baselines, ScribeAgent augmented with GPT-4o obtains the highest task success rate in 4 of 5 categories, leading to 7.3% performance improvements in total success rate over the previous-best GPT-4-Turbo-based AgentOccam[48]. In particular, on Reddit and GitLab tasks where the domains are more realistic and thus closer to the ones in our training data, ScribeAgent demonstrates stronger generalization ability and higher task success rates than in other domains. Despite known issues with `combobox` selection and the absence of scroll actions in our training data, our agent effectively navigates these challenges through strategic keyboard actions. More details are provided in Appendix A.5.2.

To better understand the contribution of ScribeAgent to the multi-agent system, we perform an ablation study that leverages GPT-4o for all four stages of the proposed pipeline. Given the large number of tasks in WebArena and the evaluation costs associated with using a multi-agent system, we perform our ablation studies using ScribeAgent-Small. As shown in Table 8, using ScribeAgent consistently outperforms only using GPT-4o, and the GPT-4o-only setting is less effective than existing agents like WebPilot. This shows that our strong performance on WebArena can be mainly attributed to the action generation process of ScribeAgent. Apart from getting better results, the multi-agent system is cheaper than using GPT-4o alone, as calling ScribeAgent to generate a next action incurs negligible cost as it is served locally.

Method	LLM	Total SR	Shopping	CMS	Reddit	GitLab	Maps
Single-Agent	**GPT-4o**	34.2	31.9	21.3	44.7	38.2	42.6
Multi-Agent	**ScribeAgent-Small + GPT-4o**	**51.3**	**48.1**	**35.5**	**70.2**	**58.8**	**51.9**

Table 8. We replace ScribeAgent with GPT-4o in our four-stage pipeline to study how much ScribeAgent contributes to the performance. The success rates drop significantly for all domains.

Agent Backbone	# Train Tokens	Total SR (158)	Shopping (36)	CMS (39)	Reddit (24)	GitLab (33)	Maps (26)
Mistral 7B	1B	41.8	41.7	30.8	50.0	42.4	42.3
Qwen2 7B	1B	44.3	52.8	33.3	50.0	48.5	42.3
Qwen2 7B	3B	47.5	55.6	33.3	58.3	48.5	**46.2**
Qwen2 7B	6B	**55.0**	**58.3**	**41.0**	**70.8**	**63.6**	**46.2**

Table 9. Task success rates on a subset of WebArena. The numbers after the domains indicate the number of tasks considered. All models are used along with GPT-4o to formulate the multi-agent system. We see that the general trends agree with what we found on our proprietary dataset.

We also use WebArena to verify the signals observed in our proprietary test data. To do so, we randomly select a subset of 158 WebArena tasks with non-overlapping objective templates and run ablation studies following the ones presented in Section 3.2.4 to study the effect of LLM backbones and the number of training tokens. As shown in Table 9, on all domains, Qwen2 7B outperforms Mistral 7B, and the task success rate increases as the number of training tokens increases. These trends suggest that improvements on our proprietary dataset lead to even greater improvements on WebArena, further highlighting the advantages of fine-tuning web agents with large-scale datasets.

5. Conclusion

In this work, we explore how fine-tuning open-source LLMs with high-quality real-world workflow data can benefit developing specialized web agents. We present ScribeAgent, which consistently outperforms existing methods that prompt proprietary models in various evaluation settings and benchmarks. We also provide empirical insights into data processing and model fine-tuning.

Limitations and Future Work.

The long-context nature of DOMs presents great challenges in adapting LLMs. In the short term, we aim to enable ScribeAgent to compare and reason over multiple DOM chunks so that its observation is always complete. This might require integrating a memory component, which could also aid in maintaining

context or state across interactions to improve multi-step reasoning. Besides, we currently do not incorporate planning into ScribeAgent, so its output will be directly used as the next action. However, adding better action selection strategies such as Monte Carlo Tree Search (MCTS) could potentially facilitate online planning and exploration, further improving the agent's decision-making processes in complex scenarios. In the long run, we aim to expand ScribeAgent's capabilities to handle multi-modal inputs and multilingual content. This would significantly broaden its applicability across different linguistic and visual contexts, making it more versatile and robust in real-world web environments.

Acknowledgments

We would like to thank the team at Scribe for their invaluable help and guidance throughout the project. Special thanks go to Thomas Kao for his assistance during the data collection and curation process. We would also like to thank Wayne Chi for helpful discussions that improved the work. This work was supported in part by the National Science Foundation grants IIS1705121, IIS1838017, IIS2046613, IIS2112471, and funding from Meta, Morgan Stanley, Amazon, Google, and Scribe. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of these funding agencies.

Appendix A.

A.1. Preprocessing

A.1.1. Pruning Pipeline

The code for preprocessing, chunking the DOM for fine-tuning, fine-tuning, and inference can be found in our GitHub.

A.1.2. Tokenizer Pruning

In this section, we provide more details on the tokenizer-based detection method to remove random character strings. The rationale behind our approach is based on the observation that typical English words consist of more than two characters. Assuming the token count is t and the character count is s, this means that when $t = 1$, $s \geq 2$, leading to $\frac{s}{t} \geq 2$. By setting the pruning threshold to 2 and removing tag attributes with $\frac{s}{t} < 2$, we aim to eliminate strings composed solely of single-character tokens, which are likely to be nonsensical.

In our actual implementation, we employ this technique only for tag attributes with $s > 32$, being more lenient for shorter attributes. To show that this tokenizer pruning strategy is effective and to study the performance across different tokenizers and pruning thresholds, we perform the following experiments.

We take three tokenizers from different models: Qwen2-7B-Instruct, Mistral-7B-Instruct-v0.3, and Meta-Llama-3-8B. For each tokenizer, we vary the pruning thresholds across a set of values: $\{1.5, 1.75, 2, 2.25, 2.5\}$. Note that it is meaningless to study overly small thresholds (e.g., it is impossible to have $\frac{s}{t} < 1$) or overly large thresholds (e.g., $\frac{s}{t} < 3$ could result in the loss of meaningful attributes, as many English words contain three letters). We randomly sample 1000 DOMs from our proprietary test dataset, apply our standard pruning pipeline followed by tokenizer pruning, and then perform three analysis:

- False positives: we use the Python enchant library to detect if there are meanful English words within the pruned strings. Note that even though these are actual words, many of them are related to DOM structure and can be safely ignored. Still, we count them as false positives since the tokenizer method is designed to remove random character strings.
- Average s and t for the entire DOM before and after tokenizer pruning: this is for understanding the reduction in content length.
- Lastly, we sort tags and attributes by the frequency of being pruned to identify patterns.

Tokenizer	Prune Threshold	False Positive (%) ↓	Before Pruning (K)		After Pruning (K)		
			s	t	s	t	Δt
Qwen2-7B-Instruct	1.5	0.025	224.3	79.14	221.4	77.11	2.03
	1.75	0.013			217.3	74.67	4.47
	2	0.18			215.7	73.89	5.21
	2.25	0.36			213.9	73.13	6.01
	2.5	0.38			210.0	71.63	7.51
Mistral-7B-Instruct-v0.3	1.5	0.012	224.3	90.54	219.5	87.10	3.44
	1.75	0.18			216.1	85.07	5.47
	2	0.44			212.7	83.40	7.14
	2.25	0.49			205.3	80.20	10.34
	2.5	11.28			190.3	74.44	16.10
Meta-Llama-3-8B	1.5	0.0097	224.3	71.44	223.1	70.60	0.84
	1.75	0.012			218.3	67.85	3.59
	2	0.035			216.8	67.09	3.43
	2.25	0.043			215.2	66.41	5.03
	2.5	0.10			212.7	65.46	5.98

Table 10. Tokenizer pruning analysis.

As shown in Table 10, there is a clear trade-off between precision and context reduction: greater reductions in content length tend to result in higher false positive rates. While different tokenizers exhibit varying sensitivities to the pruning thresholds, a threshold of 2 achieves the most balanced trade-

doi.org/10.32388/8VOGoO

off, which aligns with our intuition. We then list the top-5 tag-attribute pairs most frequently pruned under threshold 2 along with their pruning counts:

- Qwen: (‘div’, ‘class’): 3188, (‘span’, ‘class’): 11426, (‘a’, ‘href’): 8802, (‘button’, ‘class’): 6844, (‘i’, ‘class’): 5010
- Mistral: (‘div’, ‘class’): 5288, (‘span’, ‘class’): 15824, (‘a’, ‘href’): 12948, (‘button’, ‘class’): 7998, (‘svg’, ‘class’): 5871
- Llama: (‘div’, ‘class’): 29559, (‘span’, ‘class’): 8823,(‘button’, ‘class’): 5889, (‘i’, ‘class’): 4608, (‘svg’, ‘class’): 2577

Attributes such as ‘class’ often contain random character strings and are frequently pruned. However, we observe differences in how tokenizers handle the href attribute: both Qwen and Mistral tokenizers tend to prune it away, whereas the Llama tokenizer preserves it, indicating its better capability in tokenizing URLs. Although we currently use the Qwen tokenizer in our preprocessing pipeline to align with the backbone model of ScribeAgent, the Llama tokenizer can be a compelling alternative for future consideration since it is better at recognizing URLs and producing shorter token sequences. In general, we believe developing specialized models can be important to achieve strong results, as evidenced in prior works[49][50][51][52].

A.2. Example Prompt and Label for ScribeAgent

Objective: Grant delegation access to another user in Gmail settings.

URL: https://mail.google.com/mail/u/0/

Observation: {processed dom}

Step-by-step guide:

1. Description: Click “See all settings”

Action: mouse_click_action

Node: 254

Target: <button class="Tj" node="254">

2. Description: Click “Accounts”

Action: mouse_click_action

Node: 2625

Target: <a class="fo LJOhwe" href="https://mail.google.com/mail/u/0/? tab=#settings/accounts"

node="2625" role="tab">

3. Description: Click “Add another account”

Action: mouse_click_action

Node: 1215

Target:

A.3. OpenAI Prompts

A.3.1. Data Preparation

Below shows the prompt to generate step descriptions.

You are navigating a webpage to achieve an objective. Given the objective, a list of the previous actions, the current action, and a screenshot of the current action on the webpage. The objective and previous steps are only here to ground the current step, the current action and its screenshot are the most useful to your task. Give me a concise description of the current action being done on the webpage. You should look at the part of the webpage with the red circle, this is where the user clicked for the current action. Describe this action and ensure your response is in the same format, concise, coherent. Use any relevant information in the image to ground the action description. Your response should NOT use any json or markdown formatting. The response should be a single sentence that starts with an action verb. For example, 'Click on the 'SUBMIT' button.'

Regenerated Action Descriptions.

We provide a few examples of generated action descriptions using GPT-4o.

- "Click on the Submit button."
- "Type in the name of the item."
- "Double-click on the highlighted text."

A.3.2. Proprietary Benchmark Baselines

Below shows the prompt for all OpenAI baselines. The text is the prepend for every input to which we append the task input with the corresponding objective, URL, DOM, and action history.

You are an autonomous intelligent agent tasked with solving web-based tasks. These tasks will be accomplished through the use of specific actions you can issue.

Here's the information you'll have:

- The user's objective: This is the task you're trying to complete.

- The current web page's URL: This is the page you're currently navigating.

- Part of the current web page's HTML: Each element is assigned in descending order with an unique ID, denoted by the attribute \"node\".

The actions you can perform include:

- mouse_click_action: click

- keyboard_sequence_action: type a sequence of characters

- keyboard_combination_action: press a set of keys together (e.g., hotkey like ctrl+c)

You will generate a step-by-step guide to complete the task based on the given information. You will only produce a SINGLE next step.

Do NOT use additional punctuation, or any markdown formatting.

The output should be in the following format:

Description: Click \"Users\"

Action: mouse_click_action

Node: 93

Target: <a node=\"93\" class=\"slds-tree___item-label\">

Now complete the following task by generating the next step. {task input}

A.4. Mind2Web Experiment Details

A.4.1. Preprocessing

Data and Label Conversion.

To apply ScribeAgent to Mind2Web data, we first re-process the provided DOM using the procedure detailed in Section 3.2.2. We store a map between our node ID and the backend ID given in the dataset. Then, we transform the history action provided in the dataset to our 5-line format. After ScribeAgent generates the next step, we check the backend ID of the provided label and map it to the node ID in our processed DOM. We then compare this label with the target node ID generated by ScribeAgent. We provide

the code for the DOM processing and label conversion process in the supplementary material and will release them later.

DOM Chunking and Action Generation.

When the DOM length exceeds the 32K context window, we chunk the DOM sequentially and run the prediction workflow on each piece. For each piece of DOM, we call ScribeAgent five times to obtain five valid actions. We then aggregate all possible actions and select the one with the highest number of appearances. We use the following generation configuration: do_sample=True, top_p=0.95, temperature=0.6.

A.4.2. Refined Evaluation

As mentioned in the main text, we improve the Mind2Web evaluation from two perspectives:

- **Subchild label relaxation:** We hypothesize that the distribution gap between our training data for ScribeAgent and the Mind2Web test set could be due to Mind2Web preferring ancestor/parent nodes in the HTML tree, while ScribeAgent's training data prefers lower HTML elements. To this effect, we relax the Mind2Web set of positive candidates to include not only the positive candidates, but also their children (direct children and grandchildren).
- **Attribute matching:** Direct generation setting enables higher degree of freedom in element selection. To address scenarios where the predicted element has the same function as the ground truth but is in a different location, we enhance the direct generation evaluation by introducing an element attribute comparison step. Rather than merely comparing the node ID of the predicted and the ground truth elements, we also evaluate the tag and text attributes (e.g., the text displayed on a button). If these attributes match, we consider the prediction to be correct as it has identical functionality.

Lastly, we note that in Mind2Web, whenever there is a `textarea` or an `input` tag, the expected behavior is to directly execute the type action. However, our model is trained to first click on the input element and then perform the type action. Thus, for actions predicted on `textarea` or `input` tags, we adjust our model to replace click actions with type actions and then compare with the ground truths.

Table 11 presents the improved performance of ScribeAgent after refining the evaluation method, showing significant gains in both settings. We find that the label relaxation strategy helps bridge part of the distribution gap, and our multi-stage pipeline effectively covers most of the gains from this label relaxation strategy by using the Mind2Web ranker. However, inspecting cases that are not covered by

label relaxation, we found that there still remains a distribution gap. As a result, there is large room for improving the evaluation criteria of text-based benchmark to bridge this gap.

Models		Eval	Cross-Task				Cross-Website				Cross-Domain			
			EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR	EA	AF_1	Step SR	Task SR
Multi-Stage	ScribeAgent-Small	M2W	42.6	50.1	39.7	0	44.9	50.1	41.6	0.6	44.1	51.4	41.4	0
		M2W + Subchild	42.6	50.1	39.8	0	45.2	50.1	41.5	0.6	44.3	51.4	41.6	0
	ScribeAgent-Large	M2W	53.5	52.9	51.2	0	53.4	52.8	51.3	2.3	53.3	54.7	51.2	0
		M2W + Subchild	53.8	52.9	51.3	0	54.0	52.8	51.9	2.3	53.5	54.7	51.4	0
Direct Gen	ScribeAgent-Small	M2W	28.6	50.1	26.8	0	27.6	50.1	25.6	0	32.0	51.4	29.9	0
		M2W + Subchild + Attr Match	48.8	60.8	48.3	5.5	58.0	66.2	56.7	6.8	52.9	62.1	52.4	6.5
	ScribeAgent-Large	M2W	38.0	52.9	35.6	0	34.1	52.7	32.5	0	39.4	54.7	37.3	0
		M2W + Subchild + Attr Match	58.0	63.8	52.0	5.7	67.3	69.4	59.8	11.8	62.0	63.7	52.9	10.8

Table 11. We also refine the evaluation procedure to better reflect ScribeAgent's capacity.

A.5. WebArena Experiment Details

A.5.1. Four-Stage Pipeline

For the most up-to-date prompts, please refer to our GitHub.

Stage 1:

GPT-4o refines the intent. We use the following prompt:

I have a simple task objective related to {domain}, rewrite it into a single paragraph of detailed step-by-step actions to achieve the task. When revising the objective, follow the rules:\\ - Assume you are already on the correct starting website and are logged in.\\ - Do not include any newlines, tabs, step numbers in the rewritten objective.\\ - Follow the example as much as possible.\\ {In-context demonstrations for domain rules}\\ Here is an example:\\ Simple Task Objective: {in-context demonstration}\\ Detailed Task Objective: {in-context demonstrations}\\ Now, rewrite the following objective:

Stage 2:

We process the environment-generated DOM using our preprocessing procedure. When the DOM length exceeds the 32K context window, we chunk the DOM sequentially and run the prediction workflow on each piece. For each piece of DOM, we call ScribeAgent multiple times to obtain multiple valid actions. We use the following generation configuration: do_sample=True, top_p=0.95, temperature=0.6. We then aggregate all possible actions, pick the top candidates, and prompt GPT-4o to select the best candidate using the following prompt:

You are an autonomous agent helping users to solve web-based tasks. These tasks will be accomplished through series of actions. The information you'll have includes:\\ - The user's objective\\ - The current web page's URL\\ - The current web page's accessibility tree\\ - Previous steps performed by the user, where each step includes a description of the action and the target web element\\ - Several proposed next steps, labeled by ''No."\\ Your goal is to select the best next step that can complete the task and output this candidate's number, follow the following rules:\\ - Do not repeat previous steps\\ - Reject candidates with incorrect intentions, e.g., searching for an item different from the one specified in the objective\\ - Reject candidates with factual errors, e.g., the description and the chosen web target do not match\\ - Only output a single number after to represent the selected candidate but not explanation\\ Now analyze the following case:

Stage 3:

GPT-4o maps the agent output to accessibility tree format using the following prompt:

You are an autonomous agent helping users to solve web-based tasks. These tasks will be accomplished through series of actions.

The information you'll have includes:\\

- The user's objective\\

- The current web page's URL\\

- A snippit of the current web page's HTML\\

- A snippit of the current web page's accessibility tree\\

- Previous steps performed by the user\\

Your goal is to translate a proposed next step, which consists of an action and a HTML element, into the following format:\\

- 'click [accessibility tree id]': This action clicks on an interactive (non-static) element with a specific id. Note this id is the number inside ''[]" in the accessibility tree, not the HTML attribute ''node". Brackets are required in the response. For example, a valid response is ''click [1234]"\\

- 'type [accessibility tree id] [content]': Use this to type the content into the field with a specific id in the accessibility tree. For example, a valid response is ''type [1234] [New York]". The second bracket should include everything that needs to appear in the textbox, but not only the added content. Do not change the letter case\\

- 'press [key_comb]': Simulates pressing a key combination on the keyboard (e.g., press [PageDown], press [Enter])\\

- 'go_back': Return this when the current web page does not contain useful information and the user should go back to the previous web page\\

When mapping the next step into actions in the above formats, follow the following rules:\\

- Take the user's objective into consideration, so the action must help complete the task\\

- Do not repeat previous steps\\

- Only output a single step in the above format but not explanation\\

Note also: {in-context demonstration of rules}\\

Now analyze the following case:

The action is then returned to the environment for execution.

Stage 4: GPT-4o evaluates if the task objective is achieved. For operational tasks, if the task is completed, nothing is returned. For information seeking tasks, if the task is completed, GPT-4o retrieves the answer

to the question. The prompt looks like the following:

You are an autonomous agent helping users to solve web-based tasks. These tasks will be accomplished through series of actions. The information you'll have includes:\\ - The user's task, including a high-level objective and a more detailed illustration\\ - The current web page's URL and accessibility tree\\ - Previous steps performed by the user, where each step includes a description of the action and the target web element\\ You should follow the rules: {in-context demonstration rules}\\ You will decide whether the task specified by the high-level objective is completed (which means the **last** step of the detailed instruction is completed and the current webpage completes the task) and respond ‘‘completed" or ‘‘incomplete". If the task requires returning a number or a string and the answer can be obtained in the current webpage, reply ‘‘completed, [answer]" where ‘‘[answer]" is the number or string. If the task requires finding a webpage and the current webpage satisfies the requirement, reply ‘‘completed, [answer]" where ‘‘[answer]" is the current URL. Now analyze the following case. First provide the reasonings. Then summarize the answer with ‘‘Summary:", followed by ‘‘completed" or ‘‘incomplete", followed by the answer to the question if applicable. Do not include newlines after ‘‘Summary:".

A.5.2. Scrolling Actions and Combobox Selection

In our data collection process, we capture the full DOM from a system perspective, which inherently includes the entire webpage as observed from the backend. This method differs from user-centric data collection, where only the elements within the visible browser viewport are captured. Consequently, there is no concept of scrolling in our training datasets since all elements are already fully accessible in the captured data.

However, we recognize the importance of scroll actions in solving WebArena from a user perspective. To address this, before issuing any action to the environment, our multi-agent system includes a viewport check that uses the bounding box position to determine if the target element is within the visible webpage

area. If not, the system manually inserts necessary scroll actions to bring the element into view. This ensures accurate interaction with web elements in a typical user scenario.

To handle combox selection, our agent discovers a workaround that bypasses the need for scrolling through comboboxes. Specifically, after clicking on the combobox, it types the name of the desired item in the combobox, which brings the item to the top of the dropdown menu. Then, the agent can simply click the item or press Enter. This approach avoids the need for scrolling and is especially effective in densely populated lists. It improves the task success rate on a large number of Map, Reddit, and GitLab tasks.

A.5.3. GPT-4o-Only Setting

When we use GPT-4o for stage 2, we use the following prompt:

You are an autonomous intelligent agent tasked with solving web-based tasks.

These tasks will be accomplished through the use of specific actions you can issue. Here's the information you'll have:\\

- The user's objective: This is the task you're trying to complete.\\

- The current web page's URL: This is the page you're currently navigating.\\

- The current web page's HTML: Each element is assigned with an unique ID, denoted by the attribute ''node".\\

The actions you can perform include:\\

- mouse_click_action: click\\

- keyboard_sequence_action: type a sequence of characters\\

- keyboard_combination_action: press a set of keys together (e.g., hotkey like ctrl+c)\\

You will generate a step-by-step guide to complete the task based on the given information. At each step, you can perform only one action to one web element. The output should be in the correct format: a single step consisting of a text description, an action, as well as the node and HTML of the target web element to perform the action. Be coherent, concise, and accurate in your response.

Do NOT use any special characters (e.g., ''*", ''#", etc.) in your response.

Follow EXACTLY the format of the response below.\\

Here is one example:\\

Description: click ''Users"\\

Action: mouse_click_action\\

Node: 93\\ Target: <a class=''slds-tree___item-label" node=''93">\\

Now complete the following task by generating a single next step:

A.5.4. Results

Below, we provide example action trajectories produced by our agent and explanations to shed light on its decision making process.

Domain: Shopping

Task ID: 466

Intent: Add 2 Hawaiian Bamboo Orchid Roots #zc50 - by Discount Hawaiian Gifts to my wish list.

Actions:

1. click [1585] where [1585] is combobox ‘Search’ autocomplete: both hasPopup: listbox required: False expanded: False → Click on the search bar.
2. type [1585] [Hawaiian Bamboo Orchid Roots #zc50 - by Discount Hawaiian Gifts] where [1585] is combobox ‘Search’ focused: True autocomplete: both hasPopup: listbox required: False expanded: True → Type the target item name.
3. press [enter] → Do the search.
4. scroll [down] → Scroll down the page to find out the target item.
5. click [4859] where [4859] is link ‘2 Hawaiian Bamboo Orchid Roots #zc50 - by Discount Hawaiian Gifts’ → Open the item page.
6. click [6733] where [6733] is link ‘Add to Wish List’→ Add item to wish list.

Domain: CMS

Task ID: 244

Intent: Show me the email address of the customer who is the most unhappy with Olivia zip jacket.

Actions:

1. click [108] where [108] is link ‘MARKETING’ → Click on the “MARKETING” button in the side bar.
2. click [419] where [419] is link ‘All Reviews’ → Clicking the side bar shows more option. We can then click on the “All Reviews” button.
3. click [2195] where [2195] is button ‘Reset Filter’ → Remove all previous set filters.
4. type [4435] [olivia] where [4435] is textbox ‘’ required: False → In the search box, type “olivia”.
5. press [enter]→ Do the search
6. stop [emma.lopez@gmail.com] → Inspect all listed reviews and return the email of the negative review.

Domain: Reddit

Task ID: 403

Intent: Change my reddit bio to “Awesome Prompt Artist”

Actions:

1. click [12] where [12] is button ‘MarvelsGrantMan136’ hasPopup: menu expanded: False → Click my user name on the top right corner.
2. click [249] where [249] is link ‘Profile’ → After the drop-down menu shows up, click “Profile”.
3. click [886] where [886] is link ‘Edit biography’ → Click the “Edit biography” button.

4. type [2456] [Awesome Prompt Artist] where [2456] is textbox 'Biography' required: False describedby: user_biography_biography_help → Type the target content.
5. click [2474] where [2474] is button 'Save' → Save the new profile.

Domain: GitLab

Task ID: 293

Intent: Show me the command to clone Super_Awesome_Robot with SSH.

Actions:

1. click [1507] where [1507] is textbox 'Search GitLab' required: False → Click on the search bar.
2. type [1516] [Super_Awesome_Robot] where [1516] is searchbox 'Search GitLab' focused: True describedby: search-input-description → Type the repo name in the search bar.
3. click [2082] where [2082] is link 'Convex Eggtart / Super_Awesome_Robot' → Click on the correct repo.
4. click [2699] where [2699] is link 'Clone' → Click on the "Clone" button.

- stop [git clone ssh://git@metis.lti.cs.cmu:2222/convexegg/super_awesome_robot.git] → Read the command from the pop-up window.

Domain: Maps

Task ID: 7

Intent: Tell me the full address of all international airports that are within a driving distance of 50 km to Carnegie Mellon University.

Actions:

1. click [35] where [35] is textbox 'Search' focused: True required: False → Click on the search box.
2. type [35] [airport Pittsburgh] where [35] is textbox 'Search' focused: True required: False → Type "airport Pittsburgh" in the search box.
3. stop [Pittsburgh International Airport, Airport Boulevard, Findlay Township, Allegheny County, 15231, United States.] → Return "Pittsburgh International Airport, Airport Boulevard, Findlay Township, Allegheny County, 15231, United States." as the answer.

Notes

Github: https://github.com/colonylabs/ScribeAgent

Footnotes

[1] Due to privacy concerns, we restrict access to our proprietary dataset. However, we release our complete preprocessing, training, and inference code, along with a version of ScribeAgent trained on open-source datasets [5] on GitHub.

References

1. ^a, ^b, ^c *Zheng L, Wang R, Wang X, An B. Synapse: Trajectory-as-Exemplar Prompting with Memory for Comp uter Control. In: The Twelfth International Conference on Learning Representations; 2024. Available from: https://openreview.net/forum?id=Pc8AU1aF5e.*
2. ^a, ^b, ^c *Lai H, Liu X, Iong IL, Yao S, Chen Y, Shen P, Yu H, Zhang H, Zhang X, Dong Y, Tang J (2024). "AutoWeb GLM: A Large Language Model-based Web Navigating Agent". In: Proceedings of the 30th ACM SIGKDD Co nference on Knowledge Discovery and Data Mining. pp. 5295–5306.*
3. ^a, ^b, ^c *Zhang Y, Ma Z, Ma Y, Han Z, Wu Y, Tresp V (2024). "WebPilot: A Versatile and Autonomous Multi-Age nt System for Web Task Execution with Strategic Exploration". arXiv. Available from: https://arxiv.org/abs/ 2408.15978.*
4. ^ *Song Y, Xu F, Zhou S, Neubig G (2024). "Beyond Browsing: API-Based Web Agents". arXiv. Available from: https://arxiv.org/abs/2410.16464.*
5. ^a, ^b, ^c, ^d, ^e, ^f, g, ^h *Deng X, Gu Y, Zheng B, Chen S, Stevens S, Wang B, Sun H, Su Y (2023). "Mind2Web: Toward s a Generalist Agent for the Web". Thirty-seventh Conference on Neural Information Processing Systems Da tasets and Benchmarks Track. Available from: https://openreview.net/forum?id=kiYqbO3wqw.*
6. ^a, ^b, ^c, ^d, ^e, ^f *Zhou S, Xu FF, Zhu H, Zhou X, Lo R, Sridhar A, Cheng X, Ou T, Bisk Y, Fried D, Alon U, Neubig G. WebArena: A Realistic Web Environment for Building Autonomous Agents. In: The Twelfth International Co nference on Learning Representations; 2024. Available from: https://openreview.net/forum?id=oKn9c6ytL x.*
7. ^ *Drouin A, Gasse M, Caccia M, Laradji IH, Del Verme M, Marty T, Boisvert L, Thakkar M, Cappart Q, Vazque z D, et al. WorkArena: How Capable are Web Agents at Solving Common Knowledge Work Tasks? arXiv prep rint arXiv:2403.07718. 2024.*
8. ^a, ^b, ^c, ^d *Wang Z, Mao J, Fried D, Neubig G (2024). "Agent Workflow Memory". arXiv preprint arXiv:2409.07 429. Available from: https://arxiv.org/abs/2409.07429.*
9. ^a, ^b *Sodhi P, Branavan SRK, Artzi Y, McDonald R. "SteP: Stacked LLM Policies for Web Actions." In: Conferen ce on Language Modeling (COLM); 2024. Available from: https://arxiv.org/abs/2310.03720.*

10. ^OpenAI (2024). "GPT-4 Technical Report". arXiv. Available from: https://arxiv.org/abs/2303.08774.
11. a, bHu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L, Chen W (2022). "LoRA: Low-Rank Adaptati on of Large Language Models". International Conference on Learning Representations. Available from: https://openreview.net/forum?id=nZeVKeeFYf9.
12. a, bKoh JY, McAleer S, Fried D, Salakhutdinov R (2024). "Tree Search for Language Model Agents". arXiv pr eprint arXiv:2407.01476.
13. ^Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y (2022). "ReAct: Synergizing Reasoning and Acti ng in Language Models". ArXiv. **abs/2210.03629**. S2CID 252762395.
14. ^Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, Wiegreffe S, Alon U, Dziri N, Prabhumoye S, Yang Y, Wel leck S, Majumder BP, Gupta S, Yazdanbakhsh A, Clark P (2023). "Self-Refine: Iterative Refinement with Sel f-Feedback". arXiv. arXiv:2303.17651 [cs.CL].
15. ^Shinn N, Cassano F, Berman E, Gopinath A, Narasimhan K, Yao S (2023). "Reflexion: Language Agents wit h Verbal Reinforcement Learning". arXiv. arXiv:2303.11366.
16. ^Wang G, Xie Y, Jiang Y, Mandlekar A, Xiao C, Zhu Y, Fan L, Anandkumar A. Voyager: An open-ended embo died agent with large language models. Transactions on Machine Learning Research. 2024. ISSN 2835-885 6. Available from: https://openreview.net/forum?id=ehfRiFoR3a.
17. ^Sun H, Zhuang Y, Kong L, Dai B, Zhang C. "AdaPlanner: Adaptive Planning from Feedback with Language Models." In: Thirty-seventh Conference on Neural Information Processing Systems; 2023. Available from: https://openreview.net/forum?id=rnKgbKmelt.
18. ^Fu Y, Kim DK, Kim J, Sohn S, Logeswaran L, Bae K, Lee H (2024). "Autoguide: Automated generation and selection of state-aware guidelines for large language model agents". arXiv preprint arXiv:2403.08978.
19. ^Ou T, Xu FF, Madaan A, Liu J, Lo R, Sridhar A, Sengupta S, Roth D, Neubig G, Zhou S (2024). "Synatra: Tur ning Indirect Knowledge into Direct Demonstrations for Digital Agents at Scale". arXiv. arXiv:2409.15637 [c s.AI].
20. ^Shen J, Tenenholtz N, Hall JB, Alvarez-Melis D, Fusi N (2024). "Tag-LLM: Repurposing General-Purpose LLMs for Specialized Domains". arXiv. **cs.LG**: 2402.05140.
21. ^Pan J, Zhang Y, Tomlin N, Zhou Y, Levine S, Suhr A (2024). "Autonomous evaluation and refinement of di gital agents". arXiv preprint arXiv:2404.06474.
22. ^Murty S, Manning C, Shaw P, Joshi M, Lee K (2024). "BAGEL: Bootstrapping Agents by Guiding Exploratio n with Language". arXiv preprint arXiv:2403.08140.
23. a, bDubey A, ..., Zhao Z (2024). "The Llama 3 Herd of Models". arXiv. Available from: https://arxiv.org/abs/2 407.21783.

24. ^Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu J, Sauvestre R, Remez T, Rapin J, Kozhev nikov A, Evtimov I, Bitton J, Bhatt M, Canton Ferrer C, Grattafiori A, Xiong W, Défossez A, Copet J, Azhar F, T ouvron H, Martin L, Usunier N, Scialom T, Synnaeve G. Code Llama: Open Foundation Models for Code, 202 4. https://arxiv.org/abs/2308.12950.

25. ^Chung HW, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, Li Y, Wang X, Dehghani M, Brahma S, Webson A, Gu SS, Dai Z, Suzgun M, Chen X, Chowdhery A, Castro-Ros A, Pellat M, Robinson K, Valter D, Narang S, Mishra G, Yu A, Zhao V, Huang Y, Dai A, Yu H, Petrov S, Chi EH, Dean J, Devlin J, Roberts A, Zhou D, Le QV, Wei J (20 22). "Scaling instruction-finetuned language models". arXiv. arXiv:2210.11416.

26. a, bGur I, Furuta H, Huang A, Safdari M, Matsuo Y, Eck D, Faust A (2023). "A Real-World WebAgent with Pl anning, Long Context Understanding, and Program Synthesis". ArXiv. **abs/2307.12856**. Available from: http s://api.semanticscholar.org/CorpusID:260126067.

27. ^Team GLM, ..., and Zihan Wang. "ChatGLM: A Family of Large Language Models from GLM-130B to GLM -4 All Tools". 2024. arXiv [cs.CL]. Available from: https://arxiv.org/abs/2406.12793.

28. ^Murty S, Bahdanau D, Manning CD (2024). "NNetscape Navigator: Complex Demonstrations for Web Age nts Without a Demonstrator". arXiv. arXiv:2410.02907 [cs.CL].

29. a, bOpenAI (2024). "Introducing OpenAI o1". Available from: https://openai.com/o1/.

30. ^Gur I, Nachum O, Miao Y, Safdari M, Huang A, Chowdhery A, Narang S, Fiedel N, Faust A (2022). "Underst anding HTML with Large Language Models". ArXiv. **abs/2210.03945**. S2CID 252780086.

31. ^Nakano R, Hilton J, Balaji S, Wu J, Ouyang L, Kim C, Hesse C, Jain S, Kosaraju V, Saunders W, Jiang X, Cobb e K, Eloundou T, Krueger G, Button K, Knight M, Chess B, Schulman J (2022). "WebGPT: Browser-assisted q uestion-answering with human feedback". arXiv. Available from: https://arxiv.org/abs/2112.09332.

32. ^Liu X, Lai H, Yu H, Xu Y, Zeng A, Du Z, Zhang P, Dong Y, Tang J (2023). "WebGLM: Towards An Efficient W eb-Enhanced Question Answering System with Human Preferences". arXiv. arXiv:2306.07906 [cs.CL].

33. ^Hong W, Wang W, Lv Q, Xu J, Yu W, Ji J, Wang Y, Wang Z, Dong Y, Ding M, Tang J (2023). "CogAgent: A Visu al Language Model for GUI Agents". arXiv. **cs.CV**. Available from: arXiv:2312.08914.

34. ^Cheng K, Sun Q, Chu Y, Xu F, YanTao L, Zhang J, Wu Z. "SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents." In: Proceedings of the 62nd Annual Meeting of the Association for Computational Lingu istics (Volume 1: Long Papers). Bangkok, Thailand: Association for Computational Linguistics; 2024. p. 9313 -9332. Available from: https://aclanthology.org/2024.acl-long.505.

35. ^Furuta H, Lee KH, Nachum O, Matsuo Y, Faust A, Gu SS, Gur I (2024). "Multimodal Web Navigation with I nstruction-Finetuned Foundation Models". arXiv. 2305.11854.

36. ^He H, Yao W, Ma K, Yu W, Dai Y, Zhang H, Lan Z, Yu D (2024). "WebVoyager: Building an End-to-End We b Agent with Large Multimodal Models". arXiv. arXiv:2401.13919 [cs.CL].

37. a, bJaceAI (2024). "AWA 1.5 Achieves Breakthrough Performance on WebArena Benchmark". Available fro m: https://www.jace.ai/post/awa-1-5-achieves-breakthrough-performance-on-webarena-benchmark.

38. a, bMistralAI (2023). "Announcing Mistral 7B". https://mistral.ai/news/announcing-mistral-7b/.

39. ^Richardson L (2007). "Beautiful soup documentation". April.

40. ^OpenAI (2024). "GPT-4o". Available from: https://openai.com/index/hello-gpt-4o/.

41. ^Zhao B, Tu H, Wei C, Mei J, Xie C (2023). "Tuning LayerNorm in Attention: Towards Efficient Multi-Modal LLM Finetuning". arXiv. arXiv:2312.11420.

42. ^Shen J, Li L, Dery LM, Staten C, Khodak M, Neubig G, Talwalkar A (2023). "Cross-modal fine-tuning: alig n then refine". In: Proceedings of the 40th International Conference on Machine Learning; 2023; Honolulu, Hawaii, USA. Article No. 1285, 27 p.

43. ^MistralAI (2024). "Mixtral of Experts". https://mistral.ai/news/mixtral-of-experts/.

44. a, bYang A, Yang B, Hui B, Zheng B, Yu B, Zhou C, Li C, Li C, Liu D, Huang F, Dong G, Wei H, Lin H, Tang J, W ang J, Yang J, Tu J, Zhang J, Ma J, Yang J, Xu J, Zhou J, Bai J, He J, Lin J, Dang K, Lu K, Chen K, Yang K, Li M, Xue M, Ni N, Zhang P, Wang P, Peng R, Men R, Gao R, Lin R, Wang S, Bai S, Tan S, Zhu T, Li T, Liu T, Ge W, Deng X, Zhou X, Ren X, Zhang X, Wei X, Ren X, Liu X, Fan Y, Yao Y, Zhang Y, Wan Y, Chu Y, Liu Y, Cui Z, Zha ng Z, Guo Z, Fan Z. "Qwen2 Technical Report". arXiv. 2024. Available from: https://arxiv.org/abs/2407.106 71.

45. a, bQwen Team. Qwen2.5: A Party of Foundation Models. September 2024. Available from: https://qwenlm. github.io/blog/qwen2.5/.

46. ^MistralAI (2024). "Codestral: Hello World". Available from: https://mistral.ai/news/codestral/.

47. ^Su J, Lu Y, Pan S, Wen B, Liu Y (2021). "RoFormer: Enhanced Transformer with Rotary Position Embeddin g". arXiv. arXiv:2104.09864 [cs.CL].

48. ^Yang K, Liu Y, Chaudhary S, Fakoor R, Chaudhari P, Karypis G, Rangwala H (2024). "AgentOccam: A Simp le Yet Strong Baseline for LLM-Based Web Agents". arXiv. arXiv:2410.13825 [cs.AI].

49. ^Shen J, Marwah T, Talwalkar A (2024). "Ups: Towards foundation models for pde solving via cross-modal adaptation". arXiv preprint arXiv:2403.07187. Available from: https://arxiv.org/abs/2403.07187.

50. ^Tu R, Roberts N, Khodak M, Shen J, Sala F, Talwalkar A (2022). "NAS-Bench-360: Benchmarking Neural Architecture Search on Diverse Tasks". Advances in Neural Information Processing Systems (NeurIPS) Data sets and Benchmarks Track.

51. ^*Shen J, Khodak M, Talwalkar A (2022). "Efficient architecture search for diverse tasks". In: Advances in Neural Information Processing Systems (NeurIPS).*

52. ^*Roberts N, Guo S, Xu C, Talwalkar A, Lander D, Tao L, Cai L, Niu S, Heng J, Qin H, Deng M, Hog J, Pfefferle A, Shivakumar SA, Krishnakumar A, Wang Y, Sukthanker RS, Hutter F, Hasanaj E, Le TD, Khodak M, Nevmyvaka Y, Rasul K, Sala F, Schneider A, Shen J, Sparks ER. AutoML Decathlon: Diverse Tasks, Modern Methods, and Efficiency at Scale. In: Neural Information Processing Systems; 2021. Available from: https://api.semanticscholar.org/CorpusID:265536645.*

Declarations

Funding: No specific funding was received for this work.

Potential competing interests: No potential competing interests to declare.