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

Anonymous ACL submission

Abstract

LLM agents are advancing in handling web-based tasks. However, most LLM web agents rely on prompting general-purpose, proprietary models like GPT-4, which are not specifically trained to process web languages (e.g., HTML) or perform long-horizon planning. We explore an alternative paradigm of developing specialized web agents, namely supervised fine-tuning of open-source LLMs using production-scale workflow data. This strategy not only reduces serving costs but also substantially improves the empirical results—our agent achieves state-of-the-art action generation performance on the Mind2Web benchmark and improves the task success rate by 7.3% over existing prompting-based agents on WebArena. We further perform detailed ablation studies on various design choices and provide insights into LLM selection, training recipes, context window optimization, and the effect of dataset sizes.

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 (Zheng et al., 2024; Lai et al., 2024; Zhang et al., 2024; Song et al., 2024) have shown that, with better planning and exploration strategies, LLM agents can independently solve various web tasks ranging from simple navigation to more complex tasks, such as booking flights or restaurants.

Despite these improvements, the performance of existing web agents on research benchmarks remains significantly below human levels (Deng et al., 2023; Zhou et al., 2024; Drouin et al., 2024). One potential reason is their dependence on *general-purpose* LLMs. Indeed, all top-performing agents like WebPilot (Zhang et al., 2024), AWM (Wang et al., 2024), and SteP (Sodhi et al., 2024) rely on prompting proprietary models like GPT-4 (OpenAI, 2024a). These general-purpose LLMs are not optimized for interpreting web contexts such as HTML; 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 paradigm—we develop *specialized* web agents by fine-tuning open-source LLMs with large-scale real-world workflow data[1] (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 obtain agent models smaller than proprietary LLMs, reducing the serving costs. The primary contribution of this research is to demonstrate the feasibility of fine-tuning LLM agents and provide new empirical understandings of the critical components to enhance the agent's planning capabilities.

Specifically, we collect a set of proprietary workflow data representing action sequences executed by real users in real web environments. This dataset encompasses a large spectrum of websites (over 250 domains and 10,000 subdomains), task objectives, task difficulty, and task length. Each step in a workflow features not only the raw HTML-DOM of the website but also a comprehensive documentation of the action, including natural language description, mouse or keyboard operation, and the CSS selector of the target element. We format the data into a next-step prediction task and fine-tune open-source LLMs via LoRA (Hu et al., 2022).

Using the production-scale dataset, we develop WorkflowAgent, the first family of specialized

[1]Due to privacy concerns, we restrict access to our proprietary dataset. However, we will release our preprocessing, training, and inference code (see supplementary material), as well as agent models trained on open-source data.

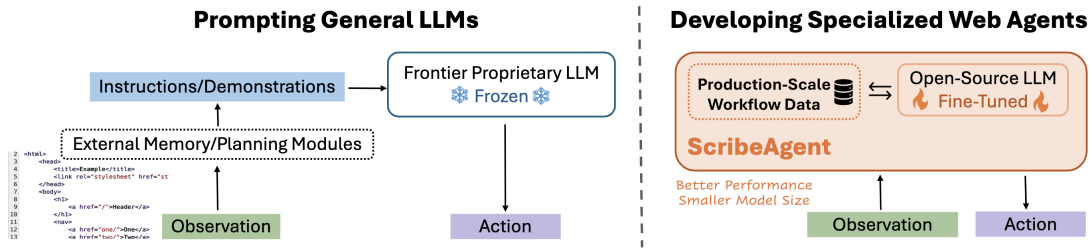

Figure 1: Most existing web agents rely on general-purpose, proprietary LLms like GPT-4 and extensive prompt engineering. We alternatively develop specialized agents by fine-tuning smaller open-source LLMs using high-quality, real-world workflow data. This boosts LLM's navigation and planning capacity and reduces serving costs.

LLM agents capable of directly generating the next step based on the website's DOM and action history. This is in contrast with previous multi-stage agents that first narrow down to a set of candidate elements using a ranking model and then select one of the candidates using a actor model (Deng et al., 2023). To evaluate the capacity and generalization ability of WorkflowAgent, we test it on public benchmarks without any further task-specific adaptation. WorkflowAgent achieves state-of-the-art direct generation performance on Mind2Web (Deng et al., 2023), with step success rate surpassing the baselines by 5-10% across all test sets. On WebArena (Zhou et al., 2024), we improve the previous best task success rate from 45.7% to 53%, marking the highest performance among text-only LLM agents.

Beyond the empirical results, our work also offers actionable insights that can inform future research on web agents: (1) we identify an effective HTML preprocessing strategy that balances between preserving essential information and minimizing context length; (2) we provide a thorough analysis on various design choices in fine-tuning, such as LLM backbone and context window selection; (3) we illustrate how fine-tuning improves agent performance as we scale up the dataset.

In summary, our work highlights how targeted fine-tuning can yield specialized and cost-effective web agents. We hope our study will reinforce interest in specializing LLMs for challenging real-world applications. Although the focus of this work is fine-tuning, WorkflowAgent can be extended to incorporate advanced search (Koh et al., 2024; Wang et al., 2024) or planning frameworks (Yao et al., 2022; Madaan et al., 2023; Shinn et al., 2023) to further enhance the capabilities of LLM web agents.

2 Related Work

Prompting-based agents. The majority of web agent works use existing LLMs and propose different prompting strategies to improve action prediction. One line of research exploits previous experience via self-feedback (Sun et al., 2023) AND in-context demonstrations (Fu et al., 2024; Zheng et al., 2024; Wang et al., 2024; Ou et al., 2024; Shen et al., 2024b). Other works focus on encouraging exploration using external evaluators (Pan et al., 2024), synthesized instructions (Murty et al., 2024b), and advanced search algorithms (Sodhi et al., 2024; Koh et al., 2024; Zhang et al., 2024). These methods rely heavily on the quality of the LLM used. Proprietary models like GPT-4 generally outperform open-source LLMs. However, fine-tuning proprietary LLMs are restricted to being done through APIs and can be costly and challenging. This implies an opportunity for enhancing open-source LLMs to match proprietary agents.

Fine-tuned agents. Compared to developing better prompting frameworks, less attention has been given to optimizing the LLMs themselves. Due to the difficulty of directly generating a single target element from the raw HTML, which often contains thousands of elements, MindAct (Deng et al., 2023) fine-tunes a small LM to filter the web elements and a larger LM to select from the filtered elements. AutoWebGLM (Lai et al., 2024) fine-tunes a single ChatGLM3 6B (GLM et al., 2024) using a combination of curriculum learning, reinforcement learning, and rejection sampling. NNetnav (Murty et al., 2024a), a concurrent work to ours, leverages synthetic demonstrations collected by LLMs for fine-tuning. Despite the complicated training and inference procedures, these methods underperform GPT-4-based agents. In contrast, our work shows that given sufficient high-quality workflow data, fine-tuning a single LLM can achieve strong performance, even outperforming GPT-4 and the more powerful o1-preview (OpenAI, 2024c).

Beyond the aforementioned work, there is an earlier line of research that fine-tunes LLMs for HTML question-answering tasks (Gur et al., 2022; Nakano et al., 2022; Liu et al., 2023). However,

these models cannot be used to generate a sequence of actions based on the user objective.

Lastly, we note that there is an emerging line of research for multi-modal web agents that use screenshots (Hong et al., 2023; Cheng et al., 2024; Furuta et al., 2024; He et al., 2024; JaceAI, 2024). Due to the lack of effective visual preprocessing schemes, we focus purely on text-based agents in this work and thus do not compare with multi-modal methods in our experiments. We leave developing multi-modal WorkflowAgent as future work.

3 WorkflowAgent

In this section, we overview the general setup of LLM web agents and detail our 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 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 it has 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 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 (Deng et al., 2023).

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 (Zhou et al., 2024).

3.2 Collecting Production-Scale Data

We collected a large set of real-world, user-annotated, proprietary data through a software that streamlines the creation of step-by-step guides for web-based tasks. This software allows users to record their interactions with the web through a browser extension and converts the interactions into well-annotated instructions. For double-blind review, we omit the name of the software for now. The collected dataset encompasses diverse domains, including social platforms like Facebook and LinkedIn; shopping sites like Amazon and Shopify; productivity tools like Notion and Calendley; customer relationship management tools like HubSpot and Salesforce; 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 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.

The dataset includes three types of actions: *mouse_click_action*, *keyboard_sequence_action* (typing a string of characters), and *keyboard_combination_action* (pressing multiple keys, e.g., ctrl+c). Note that there is no scroll action because the full DOM is captured and accessible from a system-level perspective. To maintain data quality, we remove workflows with invalid selectors.

The resulting dataset is at production scale: using raw data collected over a two-month period, we 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, real-world dataset is unmatched in prior web agent research.

3.3 DOM Preprocessing

The observation space of WorkflowAgent consists mainly of the URL and HTML-DOM, which provides the agent with all structural and content infor-

mation about the web page necessary to generate the next step. In particular, 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 because it may lose information about the HTML elements, such as the drop-down items, and does not generalize across different browsers and devices.

However, due to the dense information embedded in it, the DOM of common websites can span from 10K to 100K tokens, exceeding the context window of prevailing open-source LLMs. To reduce the DOM sizes, we develop a preprocessing procedure that maintains the essential structure and content while eliminating redundant or disruptive elements that could hinder LLM's understanding.

Specifically, we utilize a tag-attribute white list to retain only interactive elements and useful attributes. As some attribute values can contain random character sequences that do not provide useful information, we propose a new 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. 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 about tokenizer pruning can be found in Appendix A.1.

We restrict the action space of WorkflowAgent to the three types of operations specified in Section 3.2. To facilitate fine-tuning, 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. In Appendix A.2, we provide an example of a full workflow.

For the history, we consider only previous actions, omitting previous observations due to the extensive length of DOMs. Thus, at each step, our agent is given the task objective, URL, HTML-DOM, and all previous actions in the above five-line format. Its goal is to output the next action $a_t = L(q, o_t, a_{1:t-1})$ that completes the task.

3.4 Fine-Tuning with LoRA

After preprocessing, we divide the dataset into two splits. The test set comprises of 1200 workflows. We use the remaining workflows as fine-tuning data. 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, including the natural language description.

To reduce fine-tuning cost, we opt for the parameter efficient LoRA (Hu et al., 2022) instead of full fine-tuning, since we have not observed significant performance gain by updating all parameters. 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.

3.5 Exploring the Design Space

There are multiple design choices that might affect the prediction accuracy, fine-tuning cost, and inference latency. We focus on three aspects and perform detailed ablation studies.

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 (Dubey et al., 2024), Mistral 7B (MistralAI, 2023), Mixtral 8x7B (MistralAI, 2024b), Qwen2 7B (Yang et al., 2024a), Qwen2 57B (Yang et al., 2024a), Qwen2.5 14B (Team, 2024), Qwen2.5 32B (Team, 2024), and Codestral 22B (MistralAI, 2024a). 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 that contains the correct target. For evalu-

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.

ation, we use the last part. Thus, there are tasks that do not have the correct target in the DOM, i.e., the tasks are unachievable. To reflect this, we report two metrics: (1) exact match (EM) measures the model's ability to select exactly the same HTML tag as the ground truth; (2) calibrated exact match (CEM) measures the EM only on the set of achieveable examples. 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 improves the prediction accuracy. We also observe performance gains as model size increases, e.g., the CEM for Qwen2 57B is almost 2% higher than its 7B counterpart. However, fine-tuning larger models requires significantly more resources—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. Overall, the Qwen family demonstrates better performance across small, medium, and large models.

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 EM but approximately 2.5% performance drop for CEM, possibly because it becomes harder to pick the correct target given twice as many options to choose from. Besides, using 65K context window increases the inference time by approximately four times in practice.

Dataset Size. Lastly, we study the effect of fine-tuning dataset size by sampling 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

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.

# 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 evaluated with Qwen2-7B-Instruct and 32K context window.

that there is a roughly 2% performance boost when we double our dataset size.

To sum up, our experiments reveal 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 prediction accuracy but increases the inference time significantly; (3) the performance also scales with the number of training data. Based on these insights, we develop two agents: WorkflowAgent-Small, fine-tuned from Qwen2 7B, and WorkflowAgent-Large, fine-tuned from Qwen2.5 32B. Both agents leverage the full 6B-token dataset and 32K context window. While WorkflowAgent-Large demonstrates better performance in both internal and external evaluations, the 7B WorkflowAgent-Small is cheaper to serve at inference time, particularly when compared to large-scale proprietary models.

4 Results

We evaluate WorkflowAgent on three web datasets: static, text-based benchmarks, including our proprietary dataset and Mind2Web (Deng et al., 2023), as well as the interactive benchmark WebArena (Zhou

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

Table 4: WorkflowAgent v.s. non-fine-tuned, general-purpose baselines on the full test set with 32K context.

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
WorkflowAgent-Small	47.60	50.11
WorkflowAgent-Large	**50.00**	**52.60**

Table 5: Comparing WorkflowAgent with OpenAI baselines on 500 test samples with 128K context window.

et al., 2024). To evaluate the generalization ability of WorkflowAgent, we do not perform any task-specific adaptation for Mind2Web and WebArena, even when additional training data is available. Reported results are single-run due to cost constraints.

4.1 Proprietary Dataset

We first compare the performance of WorkflowAgent 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 3.3. All OpenAI baselines in this work follow the prompt in Appendix A.3.2.

Results on the test workflows are shown in Table 4. WorkflowAgent significantly outperforms the proprietary GPT-4o and 4o mini, showing 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 (WorkflowAgent-Small) boosts its performance by nearly 6 times, which highlights the importance of domain-specific data.

We also compare with OpenAI o1 series (OpenAI, 2024c), which have better planning ability. Due to API call limitations, we subsample 500 workflows for evaluation. As shown in Table 5, o1-preview performs the best among all general-purpose baselines. However, WorkflowAgent still outperforms it by a wide margin. It is important to note that WorkflowAgent-Small only has 7B parameters, while WorkflowAgent-Large has 32B parameters In contrast, most proprietary baselines are typically larger in size and require higher inference costs. This makes WorkflowAgent a better choice in terms of accuracy, latency, and cost.

4.2 Mind2Web

Mind2Web (Deng et al., 2023) features a real-world web tasks, 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 (EA), which checks if the correct target is selected; action F1 (AF1), which measures operation correctness like text input; step success rate (SR), which evaluates whether both the target element and the operation are correct; and task SR, indicating all steps are correct.

The original Mind2Web benchmark consider two sets of baselines: (1) multi-stage, multi-choice question-answering (QA) agents first use a pre-trained 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 until one action is chosen; (2) single-stage agents directly generate 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.

We evaluate WorkflowAgent on both multi-stage QA and direct generation. For the multi-stage setting, we first use the pretrained Mind2Web ranker to obtain the element ranking. Then, given the output of WorkflowAgent, 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 WorkflowAgent's prediction by the element. For direct generation, we simply compare the output of our agent to the ground truth action and target. As stated earlier, we do not fine-tune our agents to test their out-of-distribution generalization abilities.

We report results in Table 6. For the multi-stage setting, WorkflowAgent-Large achieves the best overall zero-shot performance. Our element accuracy and step success rate metrics are 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 are (1) predicting the subchild element of the ground truth instead of the ground truth; (2)

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**
	WorkflowAgent Small **(Ours)**	42.6	50.1	39.7	0	44.9	50.1	41.6	0.6	44.1	51.4	41.4	0
	WorkflowAgent 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**
	Zero-Shot												
	WorkflowAgent Small **(Ours)**	28.6	50.1	26.8	0	27.6	50.1	25.6	0	32.0	51.4	29.9	0
	WorkflowAgent 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: WorkflowAgent achieves SOTA zero-shot results on Mind2Web. Numbers are bolded for each category.

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.5.3.

As for direct generation, WorkflowAgent-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 WorkflowAgent'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 is similar across test sets.

As mentioned earlier, we observe limitations in the Mind2Web evaluation that underestimate our agent's performance. In particular, the evaluation strictly compares element IDs, but WorkflowAgent might select a functionally identical element located in a different part of the website (e.g., consider clicking on the next page button vs. clicking on the page number). To better reflect our agent's capabilities, we refine the evaluation by relaxing the labels to include subchildren of the ground truths and introducing an element attribute matching step that compares not only the element IDs but also tag and text attributes. More details and the refined results are shown in Appendix A.5.3.

We observe an average of 8% increase in task success rate and element accuracy for WorkflowAgent, 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.

4.3 WebArena

WebArena (Zhou et al., 2024) features 812 tasks across five domains: shopping, Reddit, GitLab, content management (CMS), and online map. Unlike the static Mind2Web, it implements a dynamic environment 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 WorkflowAgent operates on DOM and outputs targets in HTML, we employ GPT-4o to map between the different representations.

More specifically, we develop a multi-agent system that utilizes GPT-4o to simulate user interactions with WorkflowAgent. The system contains the following stages: (1) objective refinement: GPT-4o adds details to the task objective to help complete the task; (2) action generation: WorkflowAgent outputs an action based on the current website and action history; (3) action mapping: GPT-4o maps the agent's output in HTML to the accessibility tree format and decides whether the task is completed. More details about the pipeline can be found in Appendix A.6.

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 (JaceAI, 2024) as a baseline because it uses a proprietary system to parse the HTML-DOM and web screenshots,

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$_{axtree}$	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
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**)	WorkflowAgent-Small + GPT-4o	51.3	**48.1**	35.5	70.2	58.8	51.9
	WorkflowAgent-Large + GPT-4o	**53.0**	45.8	37.9	**73.7**	**59.7**	**56.3**

Table 7: Task success rates (SR) on WebArena domains. WorkflowAgent consistently outperforms considered text-only baselines, often improving the previous-best results by more than 5%.

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	WorkflowAgent-Small + GPT-4o	**51.3**	**48.1**	**35.5**	**70.2**	**58.8**	**51.9**

Table 8: We replace WorkflowAgent with GPT-4o to study how much WorkflowAgent contributes to the performance. Given the large number of tasks and evaluation costs, we perform our ablation studies using WorkflowAgent-Small.

rather than building from the WebArena GitHub. This allows them to have richer observations and bypass the accessibility tree action mapping step.

The results are shown in Table 7. Compared with existing text-only baselines, WorkflowAgent 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 (Yang et al., 2024b). In particular, on Reddit and GitLab tasks where the domains are more realistic and thus closer to the ones in our training data, WorkflowAgent 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 given in Appendix A.6.2.

To better understand the contribution of WorkflowAgent to the multi-agent system, we perform an ablation study that uses GPT-4o for all stages of the proposed pipeline. As shown in Table 8, using WorkflowAgent 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 results on WebArena can be mainly attributed to the action generation process of WorkflowAgent. Apart from getting better results, the multi-agent system is cheaper than using GPT-4o alone, as calling WorkflowAgent to generate an action incurs negligible cost locally.

5 Limitations

The long-context nature of DOMs presents great challenges in adapting LLMs. In the short term, we aim to enable WorkflowAgent 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 WorkflowAgent, 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 WorkflowAgent'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.

6 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 WorkflowAgent, 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.

References

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332, Bangkok, Thailand. Association for Computational Linguistics.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. 2024. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*.

Abhimanyu Dubey, ..., and Zhiwei Zhao. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *arXiv preprint arXiv:2403.08978*.

Hiroki Furuta, Kuang-Huei Lee, Ofir Nachum, Yutaka Matsuo, Aleksandra Faust, Shixiang Shane Gu, and Izzeddin Gur. 2024. Multimodal web navigation with instruction-finetuned foundation models. *Preprint*, arXiv:2305.11854.

Team GLM, ..., and Zihan Wang. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *Preprint*, arXiv:2406.12793.

Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2022. Understanding html with large language models. *ArXiv*, abs/2210.03945.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. Webvoyager: Building an end-to-end web agent with large multimodal models. *Preprint*, arXiv:2401.13919.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

JaceAI. 2024. Awa 1.5 achieves breakthrough performance on webarena benchmark.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. 2024. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295—-5306.

Xiao Liu, Hanyu Lai, Hao Yu, Yifan Xu, Aohan Zeng, Zhengxiao Du, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. Webglm: Towards an efficient web-enhanced question answering system with human preferences. *Preprint*, arXiv:2306.07906.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

MistralAI. 2023. Announcing mistral 7b.

MistralAI. 2024a. Codestral: Hello world.

MistralAI. 2024b. Mixtral of experts.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D. Manning. 2024a. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator. *Preprint*, arXiv:2410.02907.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. 2024b. Bagel: Bootstrapping agents by guiding exploration with language. *arXiv preprint arXiv:2403.08140*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

OpenAI. 2024a. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI. 2024b. Gpt-4o.

OpenAI. 2024c. Introducing openai o1.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. 2024. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. *Preprint*, arXiv:2409.15637.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. 2024. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*.

Leonard Richardson. 2007. Beautiful soup documentation. *April*.

Nicholas Roberts, Samuel Guo, Cong Xu, Ameet Talwalkar, David Lander, Lvfang Tao, Linhang Cai, Shuaicheng Niu, Jianyu Heng, Hongyang Qin, Minwen Deng, Johannes Hog, Alexander Pfefferle, Sushil Ammanaghatta Shivakumar, Arjun Krishnakumar, Yubo Wang, Rhea Sanjay Sukthanker, Frank Hutter, Euxhen Hasanaj, Tien-Dung Le, Mikhail Khodak, Yuriy Nevmyvaka, Kashif Rasul, Frederic Sala, Anderson Schneider, Junhong Shen, and Evan R. Sparks. 2021. Automl decathlon: Diverse tasks, modern methods, and efficiency at scale. In *Neural Information Processing Systems*.

Junhong Shen, Mikhail Khodak, and Ameet Talwalkar. 2022. Efficient architecture search for diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Junhong Shen, Tanya Marwah, and Ameet Talwalkar. 2024a. Ups: Towards foundation models for pde solving via cross-modal adaptation. *arXiv preprint arXiv:2403.07187*.

Junhong Shen, Neil Tenenholtz, James Brian Hall, David Alvarez-Melis, and Nicolo Fusi. 2024b. Tag-llm: Repurposing general-purpose llms for specialized domains. *Preprint*, arXiv:2402.05140.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Paloma Sodhi, S. R. K. Branavan, Yoav Artzi, and Ryan McDonald. 2024. Step: Stacked llm policies for web actions. In *Conference on Language Modeling (COLM)*.

Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. 2024. Beyond browsing: Api-based web agents. *Preprint*, arXiv:2410.16464.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. Adaplanner: Adaptive planning from feedback with language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Renbo Tu, Nicholas Roberts, Mikhail Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. 2022. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*.

Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

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. 2024a. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. 2024b. Agentoccam: A simple yet strong baseline for llm-based web agents. *Preprint*, arXiv:2410.13825.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2024. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. *Preprint*, arXiv:2408.15978.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

A Appendix

A.1 Preprocessing

A.1.1 Preprocessing Pipeline

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 (MistralAI, 2023).

For DOM pruning, we first use the BeautifulSoup library (Richardson, 2007) 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. Then, we apply tokenizer pruning for attribute values longer than 32 characters. Lastly, we remove the comments and extra whitespaces to clean up the DOM.

The code for DOM pruning, DOM chunking, fine-tuning, and inference can be found in the supplementary material. 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.

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.

As shown in Table 9, 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-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	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 9: Tokenizer pruning analysis.

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 WorkflowAgent, 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 (Shen et al., 2024a; Tu et al., 2022; Shen et al., 2022; Roberts et al., 2021).

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 (OpenAI, 2024b) 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.

A.2 Example Prompt and Label for WorkflowAgent

```
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="f0 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: <span class="LJOhwe sA" id=":kp"
node="1215" role="link">
```

A.3 OpenAI Prompts

A.3.1 Data Preparation

Below is 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
are 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}
```

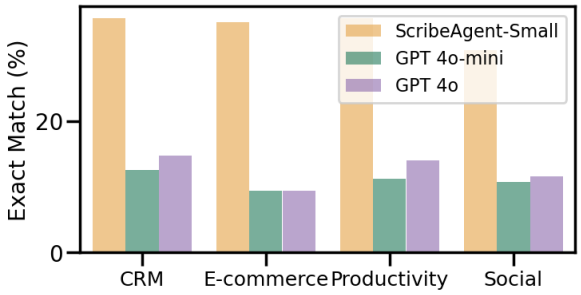

Figure 2: Exact Match (EM) comparison between WorkflowAgent-Small and OpenAI models across different types of websites.

A.4 Proprietary Data Evaluation

Apart from the results shown in the main text, 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 WorkflowAgent consistently outperforms the general-purpose baselines across all of them.

A.5 Mind2Web Experiment Details

Mind2Web is distributed with an MIT license.

A.5.1 Preprocessing

Data and Label Conversion. To apply WorkflowAgent to Mind2Web data, we first re-process the provided DOM using the procedure detailed in Section 3.3. 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 WorkflowAgent 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 WorkflowAgent. 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 WorkflowAgent 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.5.2 Baselines and Evaluation Results

For baselines, we note that AutoWebGLM (Lai et al., 2024) 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.

The outputs of WorkflowAgent on all Mind2Web test datasets are included in the supplementary material.

A.5.3 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 WorkflowAgent and the Mind2Web test set could be due to Mind2Web preferring ancestor/parent nodes in the HTML tree, while WorkflowAgent'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 10 presents the improved performance of WorkflowAgent 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.

A.6 WebArena Experiment Details

WebArena is distributed with an Apache-2.0 license.

A.6.1 Multi-Agent System

For the most up-to-date prompts, please refer to the code in the supplementary material.

Objective Refinement 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:

Action Generation 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 WorkflowAgent 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:

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	WorkflowAgent-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
	WorkflowAgent-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	WorkflowAgent-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
	WorkflowAgent-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 10: We also refine the evaluation procedure to better reflect WorkflowAgent's capacity.

Target Mapping GPT-4o maps the agent output to accessibility tree format using the following prompt. The action is then returned to the environment for execution.

> 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:

Task Completeness Evaluation 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.6.2 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.6.3 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.6.4 Results

Below, we provide example action trajectories produced by our agent and explanations to shed light on its decision making process. The full trajectories for all tasks are included in the supplementary material.

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.

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 11: 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.

4. click [2699] where [2699] is link 'Clone' → Click on the "Clone" button.

5. stop [git clone ssh://github/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.

A.6.5 Using WebArena to Verify Proprietary Evaluation Results

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 3.5 to study the effect of LLM backbones and the number of training tokens. As shown in Table 11, 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.

A.7 Broader Impact

This paper calls for ML community's attention to take advantage of LLMs and apply them to a wider range of real-world problems beyond the traidtional NLP domains, such as web navigation. This moves towards truly democratizing machine learning in real life. In terms of broader societal impact, our work can exert a positive influence as it contributes to reusing existing models and resources, reducing the computational burden of developing new large-scale models on massive data. However, lowering the barrier for applying LLMs to a wide range of tasks necessarily comes with the risk of misuse. For instance, LLM agents can exhibit unintended biases, and they also have the potential to cause harm to users (e.g., economically) in the real world if there are not careful safeguards. Hence, it is imperative to develop adaptation methods with better privacy, safety, and fairness guarantees.

A.8 Intended Use

The code released with this paper is only for research purposes and helps with developing web agents. The models we presented in this paper are not intended for direct deployment in practical applications in their current state due to a lack of safeguards.