Prune4Web: DOM Tree Pruning Programming for Web Agent

Jiayuan Zhang*, **Kaiquan Chen***, **Zhihao Lu, Enshen Zhou, Qian Yu, Jing Zhang**†

School of Software & QRI, Beihang University, Beijing, China
{zhangjiayuan42, zhang_jing}@buaa.edu.cn

Abstract

Web automation uses intelligent agents to perform high-level tasks by mimicking human interactions with webpages. Despite recent advances in LLM-based web agents, efficiently navigating complex, real-world webpages remains challenging due to massive DOM structures (10,000∼100,000 tokens). Current approaches either truncate DOMs—losing vital information—or use inefficient heuristics and separate ranking models, failing to balance precision and scalability. We introduce **Prune4Web**, a novel paradigm that transforms DOM processing from LLM-based filtering to programmatic pruning. Our key innovation is DOM Tree Pruning Programming, where an LLM generates executable Python scoring programs to dynamically filter DOM elements based on semantic clues from decomposed sub-tasks. This approach eliminates the need for LLMs to process full DOMs, instead delegating traversal and scoring to lightweight, interpretable programs. The result is a **25∼50 times reduction** in candidate elements for grounding, enabling precise action localization without attention dilution. Additionally, we propose a data annotation method and a two-turn dialogue training strategy that jointly optimizes Planner, Programmatic Filter, and Grounder in a unified framework. Experiments demonstrate state-of-the-art performance. On our low-level task grounding task, our approach dramatically increases grounding accuracy from **46.80% to 88.28%**, highlighting its effectiveness.

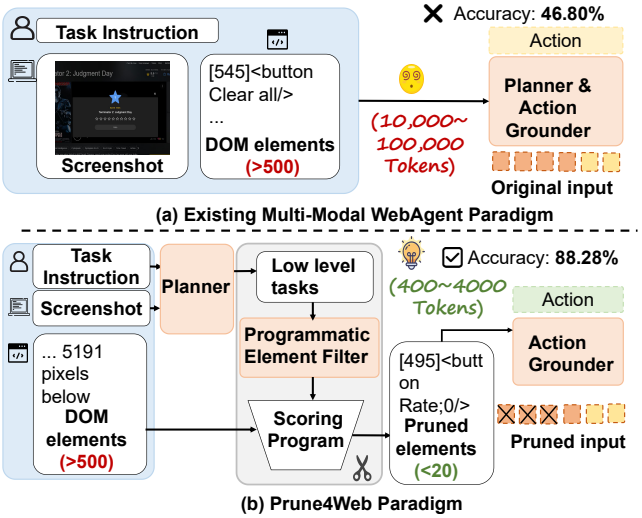

Figure 1: Comparison between existing multi-modal web agents and our Prune4Web paradim. Compared to existing multi-modal web agent paradigms, we propose a programmatic pruning strategy that efficiently removes redundant DOM elements. Our Prune4Web approach relaxes the token limits of LLMs and increases accuracy on low-level sub-task grounding from 46.80% to 88.28%.

1 Introduction

Web automation enables the completion of high-level tasks, such as booking flights or shopping online, through intelligent agents that mimic human interaction on webpages. These agents achieve this by interpreting high-level tasks, breaking them down into low-level sub-tasks, and seamlessly interacting with web elements. Recently, large language models (LLMs) have demonstrated impressive capabilities in autonomous web navigation through their strong reasoning and decision-making abilities (Yao et al. 2022; Deng et al. 2023a). Current web agents approaches fall into three main categories: 1) Textual HTML/DOM-based (Yao et al. 2022; Song et al. 2025), 2) Visual Screenshot-based (Lin et al. 2024; Cheng et al. 2024), and 3) Multi-modal-based (He et al. 2024; Zheng et al. 2024a). Visual screenshots provide an intuitive, human-like understanding of webpage state, making them effective for reasoning about low-level sub-tasks. However, they contain limited semantic information, especially for special icons, and are sensitive to variations in resolution and overlapping elements. In contrast, HTML/DOMs offer precise and stable semantic and structural information that enables accurate element selection with minimal ambiguity.

In this paper, we leverage the complementary advantages of text and visual multi-modal information and design a multi-stage framework: A **planner** model takes the high-level task (e.g., "Book a flight to New York") and a **screenshot**, then decomposes it into a low-level sub-task (e.g., "Find the destination field and Type NYC"). Based on the sub-task, an **action grounder** model processes the **DOMs** to precisely localize and execute the required op-

*These authors contributed equally.
†Corresponding author: Jing Zhang

erations (e.g., selecting <input id="destination">to type "NYC"). However, modern webpage DOMs typically contain 10,000–100,000 tokens—far exceeding the context capacity of most LLMs. This results in token truncation and attention dilution, leading to critical information loss and significant processing delays (Gou et al. 2024a; Deng et al. 2023a). Existing HTML pruning methods fall short, either relying on overly simplistic heuristic filtering (He et al. 2024; Pan et al. 2024) or requiring separate language models for element-ranking (Deng et al. 2023a). Neither approach effectively addresses the core issue. The fundamental challenge remains: *how to efficiently and accurately navigate task-relevant elements from complete DOM structures.*

To this end, we propose a **Prune4Web** pipeline through a novel paradigm: DOM Tree Pruning Programming. We observe that the low-level sub-tasks (e.g.,"Find the destination field") output by the planner contain extensive semantic clues about potentially relevant DOM elements. This insight motivates us to shift the LLM's role from directly locating elements in lengthy DOMs to generating a locator program based solely on the low-level sub-tasks, thereby avoiding the need to feed long DOM sources into the LLMs (Jiang et al. 2024a; Zhang et al. 2023b). Specifically, we implement this concept through our **Programmatic Element Filter** model. This filter receives a specific low-level sub-task from the upstream Planner and prompts the LLM to generate a concise, task-specific Python *scoring program*. We design a heuristic-based scoring program template, requiring the LLM to generate only key parameters for better controllability and flexibility. The generated program runs independently outside the LLM, efficiently traversing the complete DOM tree to score and rank all elements. This approach reduces candidate elements by **25∼50 times**, enabling precise action localization without attention dilution. A downstream LLM-based Action Grounder then selects the final element from this refined shortlist, completing the grounding task.

To train the models within Prune4Web, we create an automated data synthesis pipeline that annotates structured intermediate outputs from raw data with minimal human intervention. These include low-level sub-tasks for the Planner and key parameters for the Programmatic Element Filter. For optimization, we develop a novel two-turn dialogue training strategy that jointly trains the Planner, Filter, and Grounder as a unified model. We initially use Supervised Fine-Tuning (SFT) with our annotated data to train a base model (Zheng et al. 2024b). Subsequently, we apply Reinforcement Fine-Tuning (RFT) to enhance the Planner's long-term planning capabilities while integrating the programmatic filtering process into this optimization framework. Extensive experiments on benchmark datasets (Deng et al. 2023a; Pan et al. 2024) demonstrate the effectiveness of the proposed Prune4Web. Notably, on our low-level sub-task grounding benchmark, our approach greatly boosts grounding accuracy from **46.8%** to **88.28%**, showing its core advantage. Our contributions are summarized as follows:

- We design a multimodal web agent framework that seamlessly combines the intuitive reasoning of visual inputs with the semantic precision of HTML/DOM.
- We introduce Prune4Web with a Programmatic Element Filter that generates task-specific Python scoring programs to efficiently filter and rank elements to address the DOM scalability bottleneck.
- We present a data annotation method and a two-turn dialogue training strategy that jointly optimize the planner, filter, and grounder. We use SFT and RFT to enhance planning and programmatic filtering. Strong empirical evidence validates our method on standard benchmarks.

2 Related Work

Multimodal-based Web Agents. To achieve higher precision in web interaction, directly processing HTML source code has recently become a significant research direction for LLM-based agents (Lai et al. 2024; Song et al. 2025), leading to notable advancements. Researchers have leveraged the rich semantic and structural information within the DOM by developing multimodal fusion techniques (Zheng et al. 2024a; Furuta et al. 2023) or more powerful end-to-end models (Lin et al. 2024; Cheng et al. 2024; Hong et al. 2024; Xu et al. 2025) for precise element localization and operation. However, these efforts toward precision inevitably face the challenge of information overload (Gou et al. 2024a; Deng et al. 2023a; Xue et al. 2025). Modern webpage HTML sources typically contain vast amounts of irrelevant information. Feeding this directly into an LLM wastes computational resources and dilutes the model's focus across lengthy context (Gur et al. 2023). Balancing HTML's precision with efficient information processing remains a critical, unsolved challenge.

DOM Tree Pruning Strategies. DOM tree pruning is a key technique for addressing information overload challenge. Existing methods fall into two categories. The first is rule-based filtering, which relies on fixed heuristics like converting the DOM to a simplified accessibility tree (He et al. 2024; Zhou et al. 2023). The second is LLM-based ranking, where the model is prompted to score and select from a large number of element candidates (Deng et al. 2023a; Lù, Kasner, and Reddy 2024; Kerboua et al. 2025). Rule-based approaches are too rigid and generalize poorly. LLM-based ranking fails to reduce the burden of processing long contexts. In contrast, our work introduces DOM Tree Pruning Programming, a new paradigm that addresses both limitations by having the LLM generate a lightweight locator program (Qiao et al. 2024; Jiang et al. 2024b).

Programmatic Thinking for Agents. Our method is rooted in programmatic thinking—a paradigm that enhances LLM abilities by prompting them to generate intermediate code or plans to solve complex problems. This approach has shown effective for general reasoning and planning (Jiang et al. 2024a; Zhang et al. 2023b; Wei et al. 2022; Gupta and Kembhavi 2023; Zhou et al. 2025b; Qin et al. 2024). In the agent field, programmatic thinking typically generates high-level action sequences that control agent behavior on the web (Ma et al. 2023), mobile devices (Wen et al. 2024; Zhang et al. 2023a), or general computer operations (Zhang et al. 2024; Tan et al. 2024; Wu et al. 2024; Xie et al. 2025). Our work innovatively applies this paradigm to the lower-level prob-

lem of DOM filtering by generating an executable scoring function that actively reshapes the model's input.

Reinforcement Fine-Tuning for Agents. To enable agents to learn complex policies beyond static datasets, Reinforcement Fine-Tuning (RFT) is increasingly used to optimize LLM agents for sequential decision-making in dynamic environments (Lu et al. 2025; Qi et al. 2024; Bai et al. 2025; Guo et al. 2025). RFT allows agents to learn from outcomes via a reward mechanism, enabling them to master complex strategies. In Prune4Web, we not only employ RFT to optimize the Planner's capabilities but also innovatively use the success or failure of our DTPP process to provide rich intermediate reward signals (Wang et al. 2025; Zhou et al. 2025a), facilitating more efficient policy learning.

3 Method

3.1 Prune4Web Framework and Workflow

We introduce Prune4Web, a multi-stage framework for complex web automation tasks. The complete workflow is shown in Figure 2. The framework consists of three stages: task planning, element filtering, and action grounding.

Planning Stage. The workflow begins with the Planner model, which decomposes a high-level task T into low-level sub-tasks S_t based on the current webpage screenshot Sc_t and operational history H_t. This process is formally expressed as: $S_t = \text{Planner}(T, Sc_t, H_t)$. For example, given the high-level task "Book a flight to New York," the Planner might generate low-level sub-tasks like "Find the destination field" and "Type NYC", as well as current states. The Planner intentionally does not access the HTML source code, keeping its focus on high-level strategic decomposition.

Filtering Stage. If a low-level sub-task S_t requires interaction with a specific element, the workflow proceeds to the filtering stage managed by the Programmatic Element Filter model. This model implements our core method, DOM Tree Pruning Programming, to generate a refined list of candidate elements C_t from the complete HTML source code: $C_t = \text{ProgrammaticElementFilter}(S_t, \text{HTML}_t)$. The resulting list C_t then serves as the sole input for the subsequent Action Grounder.

Action Grounding Stage. The Action Grounder completes the workflow by generating the final executable action A_t. It takes two inputs: the low-level sub-task S_t from the Planner and the pruned candidate list C_t from the Programmatic Element Filter. This is formally expressed as $A_t = \text{ActionGrounder}(S_t, [C_t])$, where the brackets indicate that $[C_t]$ is conditional. This is because $[C_t]$ is only required for element-specific actions (e.g., 'click'), whereas abstract actions (e.g., 'task complete') are grounded directly from S_t.

In summary, the Prune4Web framework offers a dual advantage. It uses the structured, precise information from the DOM to avoid the pitfalls of visual-based localization in complex scenarios. At the same time, its innovative filtering stage distills verbose HTML into a concise list of candidates. This effectively mitigates information overload and significantly reduces the difficulty and error rate of the grounding task for the Action Grounder.

3.2 DOM Tree Pruning Programming

DOM Tree Pruning Programming is the technical core of Prune4Web. It offloads the heavy task of element filtering from the LLM itself to a lightweight, dynamically generated program.

Step 1: Initial Rule-based Filtering. The process begins with a rule-based preliminary filtering of the raw HTML_t. The core principle is to retain elements with clear interactive features based on their tags (e.g., <a>, <button>, <input>) or 'role' attributes (e.g., 'checkbox'). For non-interactive elements, we extract key textual information (from 'text', 'aria-label', etc.) and attach it to the nearest interactive element as supplementary context. This step yields a pre-processed DOM tree containing only context-enriched interactive elements, serving as a more structured and less noisy initial candidate set.

Step 2: Scoring Function Generation. The core task of the Programmatic Element Filter is to generate a Python scoring function f_{score_t} for the current step. We design a heuristic-based Scoring Function Template, where the LLM only needs to generate key parameters for this template. This approach significantly improves the stability and controllability of the generated code while maintaining flexibility. Algorithm 1 shows the pseudo-code of the template. The template mimics human intuition when searching for elements using keywords. It assumes that a target element contains identifiable textual features within the HTML. The template performs tiered, weighted matching across different attributes: Tier 1 includes visible 'text'; Tier 2 includes non-visible but high-semantic attributes like 'aria-label' and 'placeholder'; and Tier 3 includes other attributes like 'class' or 'id' that may contain semantic cues. The template also integrates multiple matching types (e.g., exact, substring, fuzzy) and assigns weights based on match quality. With this design, the Programmatic Element Filter simply generates a set of keywords and their corresponding base weights based on the low-level sub-task S_t, enabling multi-faceted relevance scoring for each element.

Step 3: Pruning Execution and Output Formatting. The generated scoring function f_{score_t} is immediately executed to compute a score s for each element e in the pre-processed DOM tree. The system then selects the Top-N highest-scoring elements, where N defaults to 20. The impact of varying N from 1 to 20 on pruning efficiency is visualized and analyzed in the Experiments section.

Discussion. The primary advantage of DOM Tree Pruning Programming lies in its combination of flexibility and structure. The LLM provides high-level, context-aware intelligence by generating keywords and weights, while the hard-coded function template ensures robust, efficient, and interpretable scoring and execution. By generating a lightweight function instead of directly processing lengthy raw HTML, this paradigm avoids attention dilution from long contexts and significantly reduces inference latency.

3.3 Data Synthesis

To effectively train our multi-stage architecture and rigorously evaluate DOM Tree Pruning Programming, we reconstructed and re-annotated the public Multimodal-Mind2Web

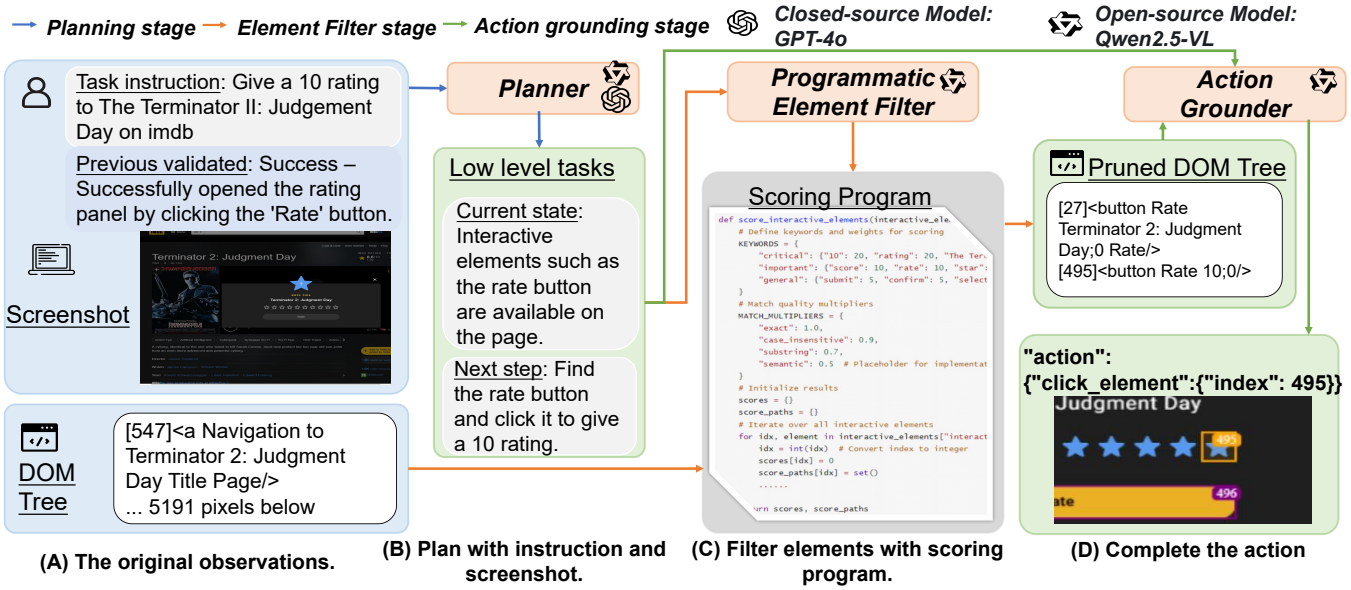

Figure 2: The Prune4Web framework pipeline. (A) The input observations include high-level task, history, screenshot, and DOM tree. (B) The Planner generates low-level subtasks based on user instruction, history, and screenshot. (C) Guided by the plan, the Programmatic Element Filter produces a scoring program that is applied to the DOM tree elements to yield a pruned DOM tree. (D) This pruned DOM forms the refined input for the Action Grounder, which then selects an executable action.

(MM2W) dataset (Deng et al. 2023a). The original MM2W dataset contains only high-level tasks, per-step source code, and final target elements and actions, lacking the intermediate reasoning steps our framework requires. To address this, we used GPT-4o (Hurst et al. 2024) as an annotation tool to add rich intermediate labels for each step. These labels include: (1) low-level sub-tasks for the Planner; (2) keywords and their weights for the Programmatic Element Filter; and (3) pruned DOM trees and thought processes for the Action Grounder. After annotation, we performed secondary cleaning and manual verification to ensure high data quality while strictly adhering to MM2W's original train/test splits. Using this annotated data, we constructed a new evaluation set that treats generated low-level sub-tasks as direct input to evaluate the grounding performance of subsequent models, directly validating the effectiveness of DOM Tree Pruning Programming. Our final dataset contains approximately 5,000 high-quality interaction steps, divided into training and test sets.

3.4 Training Strategy

Our training strategy involves two core stages: Supervised Fine-Tuning (SFT) and Reinforcement Fine-Tuning (RFT), both conducted on the Qwen2.5VL-3B-Instruct model.

Supervised Fine-Tuning (SFT). The goal of SFT is to teach the base model to perform three distinct roles: Planner, Programmatic Element Filter, and Action Grounder. We explored two SFT paradigms: Separate Models and Unified Model. In the Separate Models approach, we fine-tune three independent models, each specialized for one task. The Planner maps high-level tasks and screenshots to low-level sub-tasks. The Programmatic Element Filter maps low-level sub-tasks to scoring program parameters. The Action Grounder maps low-level sub-tasks and pruned lists to final actions. In the Unified Model approach, we designed an innovative two-turn dialogue template to optimize a single model. In the first turn, the model acts as both Planner and Programmatic Element Filter, generating the low-level subtask and scoring parameters simultaneously. In the second turn, after receiving the pruned list from the executed program, the model acts as the Action Grounder to output the final action. Experiments show the unified model is better suited for highly-coupled web automation tasks.

Reinforcement Fine-Tuning (RFT). As SFT is insufficient for teaching complex, long-horizon planning and task decomposition, we employ Group Relative Policy Optimization (GRPO) (Shao et al. 2024) for targeted RFT of the Planner (or the first turn of the unified model). We apply RFT selectively to the Planner because the Programmatic Element Filter and Action Grounder handle more deterministic tasks that can be effectively learned through SFT. The success of RFT depends on our Hierarchical Reward Mechanism, which provides timely feedback to the Planner based on downstream model performance. The Planner's reward R_{total} at each step combines format and accuracy components: $R_{total} = R_{format} + R_{filtering} + R_{grounding}$. Here, R_{format} ensures the generated low-level sub-task follows the correct format, $R_{filtering}$ provides critical intermediate feedback by verifying whether the Programmatic Element Filter's program successfully retains the ground-truth element in the pruned list, and $R_{grounding}$ measures final subtask success based on the Action Grounder's output. In our design, these rewards are binary rewards (1 for success, 0 for failure).

Algorithm 1: Scoring Function Template

Input: E: Pre-processed list of candidate elements;
W: Keywords with base weights ($\{k : w_{base}\}$) generated by LLM
Output: S: Final relevance scores for ranking;
P: Scoring justifications (paths)
Hyperparameters: $\alpha_1 > \alpha_2 > \alpha_3 > \alpha_4$ *// Match Quality: Exact > Phrase > Word > Fuzzy*
$\beta_1 > \beta_2 > \beta_3$ *// Attribute Priority: Visual Text > Trusted Attribute > Other Attribute*

for each element e in E **do**
  $S[e] \leftarrow 0$; $P[e] \leftarrow \emptyset$
  for each attribute pair $(text, type)$ in e **do**
    if $type$ is Visual Text (e.g., text_content) **then**
      $\beta \leftarrow \beta_1$
    else if $type$ is Trusted Attribute (e.g., name) **then**
      $\beta \leftarrow \beta_2$
    else if $type$ is Other Attribute (e.g., class, id) **then**
      $\beta \leftarrow \beta_3$
    end if
    $tokens \leftarrow$ Split$(text)$
    for each keyword k in W **do**
      $\alpha \leftarrow 0$
      if $text = k$ **then**
        $\alpha \leftarrow \alpha_1$ *// full exact match*
      else if (k has spaces) **and** (k is substring of $text$) **then**
        $\alpha \leftarrow \alpha_2$ *// phrase match (on raw text)*
      else if (**not** k has spaces) **and** ($k \in tokens$) **then**
        $\alpha \leftarrow \alpha_3$ *// word match (on token list)*
      else if FuzzyScore$(k, text, tokens) > \theta$ **then**
        $\alpha \leftarrow \alpha_4 \times$ FuzzyScore$(k, text, tokens)$ *// fuzzy match*
      end if
      if $\alpha > 0$ **then**
        $S[e] \leftarrow S[e] + (W[k] \times \alpha \times \beta)$
        $P[e].add($Path$(k, type, \alpha))$
      end if
    end for
  end for
end for
return (S, P)

4 Experiments

4.1 Experimental Setup

Benchmarks, Datasets, and Metrics. We conduct our primary offline evaluation on the standard Multimodal-Mind2Web benchmark (Deng et al. 2023a), following its official evaluation metrics (Element Accuracy, Operation F1, and Step Success Rate). For model fine-tuning, we use a custom dataset of approximately 5,000 interaction steps created by re-annotating and cleaning the Multimodal-Mind2Web training and development sets (detailed in Section 4.2). To assess the effectiveness of DOM Tree Pruning Programming, we build a new evaluation set from our re-annotated data. This benchmark uses ground-truth low-level sub-tasks as direct input to evaluate the grounding performance of the Programmatic Element Filter and Action Grounder models. We measure low-level sub-task grounding results using grounding accuracy. Additionally, we conduct targeted ablation studies on a curated set of online, dynamic websites, using LLM-Verified Task Completion Rate as the primary metric in Section 4.3.

Implementation Details. Our evaluation focuses on two versions of Prune4Web: a Two-turn Dialogue Unified version and a Separate Models version, both fine-tuned from Qwen2.5VL-3B-Instruct (Bai et al. 2023). To assess low-level sub-task grounding performance, we also trained a lighter Qwen2.5-0.5B-Instruct (Bai et al. 2023) model, demonstrating that our Programmatic Element Filter and Action Grounder operate effectively with lightweight LLMs. We developed all Prune4Web models using the two-stage SFT+RFT training approach described in Section 4.3.

Baselines. We compare our method with proprietary models such as GPT-4 (Achiam et al. 2023), GPT-4o (Hurst et al. 2024), and SeeAct (Zheng et al. 2024a), as well as state-of-the-art fine-tuning methods based on open-source models, including SeeClick-9.6B (Cheng et al. 2024), MiniCPM-3.1B (Hu et al. 2024), ScribeAgent-32B (Shen et al. 2024), GPT-4o UGround (Gou et al. 2024b), EDGE-9.6B (Chen et al. 2024), and MindAct Flan-T5XL (Deng et al. 2023b).

4.2 Main Results

Performance on Standard Web Benchmarks. On the official Multimodal-Mind2Web test splits (results in Table 1), our proposed Prune4Web, particularly the Two-turn Dialogue unified model, demonstrates strong performance and significantly outperforms several baselines. Notably, our model achieves this competitive performance on a moderately sized training set of only ∼5,000 trajectories while directly processing raw, complex HTML. This demonstrates our method's excellent data efficiency and significant potential for improvement.

Performance on Low-Level Sub-Task Grounding. To precisely and isolatingly evaluate the effectiveness of DOM Tree Pruning Programming, we use a ground-truth low-level sub-task as direct input to evaluate the grounding performance of the Programmatic Element Filter and Action Grounder models. Since the unified Two-turn Dialogue model cannot be easily dissected for this purpose, we evaluate the Programmatic Element Filter and Action Grounder models trained using the Separate Models strategy. We report results for: 1) fine-tuning the Qwen2.5VL-3B-Instruct model using original HTML without pruning, 2) oracle pruning (GT elements guaranteed in top candidates), 3) direct pruning and decision with LLMs, and 4) our Prune4Web pruning and decision. The results (Table 2) show that, given a perfect low-level sub-task, our full Programmatic Element Filter–Action Grounder pipeline achieves a grounding accuracy of 88.28%. This performance far surpasses the baseline without pruning (46.8%) and significantly outperforms using the more powerful GPT-4o as the Action Grounder (80.65%). Additionally, even with the much lighter Qwen2.5-0.5B-Instruct, our method shows superior performance on both pruning results and grounding

Method	Cross-Task			Cross-Website			Cross-Domain		
	Ele. Acc	Op. F1	Step SR	Ele. Acc	Op. F1	Step SR	Ele. Acc	Op. F1	Step SR
Proprietary Models									
GPT-4 (Achiam et al. 2023)	40.8	63.1	32.3	30.2	61.0	27.0	35.4	61.9	29.7
GPT-4o (Hurst et al. 2024)	5.7	77.2	4.3	5.7	79.0	3.9	5.5	**86.4**	4.5
SeeAct (Zheng et al. 2024a)	46.4	73.4	40.2	38.0	67.8	32.4	42.4	69.3	36.8
Open-Source Models									
SeeClick-9.6B (Cheng et al. 2024)	26.3	86.2	23.7	21.9	**82.9**	18.8	22.1	84.1	20.2
MiniCPM-3.1B (Hu et al. 2024)	23.8	**86.8**	20.8	20.3	81.7	17.3	17.9	74.5	14.6
ScribeAgent-32B (Shen et al. 2024)	38.0	52.9	35.6	34.1	52.7	32.5	39.4	54.7	37.3
GPT-4o UGround (Gou et al. 2024b)	47.7	–	–	46.0	–	–	46.6	–	–
EDGE-9.6B (Chen et al. 2024)	–	–	30.0	–	–	21.1	–	–	22.4
MindAct Flan-T5XL (Deng et al. 2023b)	55.1	75.7	52.0	42.0	65.2	38.9	42.1	66.5	39.6
Prune4Web Variants									
Prune4Web-3B (Separate Models)	46.0	83.4	42.2	43.0	77.3	37.8	42.2	84.4	40.6
Prune4Web-3B (Two-turn Dialogue Unified)	**58.4**	84.1	**52.4**	**50.2**	81.2	**44.9**	**49.2**	84.4	**46.1**

Table 1: Performance on the *Multimodal-Mind2Web* benchmarks across different methods. (Ele. Acc: Element Accuracy; Op. F1: Operation F1; Step SR: Step Success Rate). Two variants of our Prune4Web are evaluated. The Separate Models approach uses three independent models for the planner, filter, and grounder. The Unified Model approach uses an innovative two-turn dialogue training strategy to optimize three stages as a single model. Top-1 accuracy is represented using **bold text**.

accuracy. This experiment demonstrates that our DOM Tree Pruning Programming method achieves state-of-the-art performance in precise element localization and operation.

4.3 Ablation Studies and Further Analyses

To meticulously validate the contributions of our key design choices, we conduct ablation studies and further analyses. These experiments investigate the precision of our filtering mechanism, the effectiveness of programmatic filtering compared to simpler baselines, the contribution of our multi-stage architecture, and the efficacy of our training strategies. We also evaluate the framework's robustness in dynamic online environments to demonstrate its practical applicability.

Performance in Dynamic Online Environments. The effectiveness of Prune4Web in dynamic online environments is demonstrated through our ablation studies in Table 3 and Table 4. Component analysis on a curated set of 30 online tasks shows consistent performance improvements. Our programmatic filtering significantly enhances task completion rates for smaller models like GPT-4o-mini, while the complete three-stage architecture achieves the best overall results. These findings confirm the framework's generalization capability and practical applicability in real-world settings.

Effectiveness of Programmatic DOM Filtering. We compared our programmatic filtering against a baseline where the LLM directly performs Top-N selection in Table 3. The results show that for the powerful GPT-4o, our method maintains a high level of performance. However, its true value is demonstrated on smaller models. For GPT-4o-mini, Prune4Web's filtering boosts the task completion rate by over 5 percentage points (from 26.3% to 31.6%). For our fine-tuned Qwen2.5VL-3B, the baseline fails completely (0.0%), while our structured method achieves a functional score (5.2%). This highlights that the programmatic approach is essential for enabling smaller or specialized models to handle complex filtering tasks. filtering tasks.

Contribution of the Multi-Stage Architecture. We evaluated the necessity of our three-stage Planning-Filtering-Action Grounding architecture. As shown in Table 4, each stage provides a clear benefit. For GPT-4o-mini, starting with only an Action Grounder yields 21.1% task completion. Adding the Planner boosts this to 26.3%, and further adding our Programmatic Element Filter brings the final performance to 31.6%. This steady improvement validates each component's contribution and confirms the rationality of our complete multi-stage design.

Efficacy of Training Strategies. We also assessed the impact of the RFT in Table 5. The results show that adding RFT on top of SFT consistently and significantly improves the Planner's capabilities. For the Separate Models framework, RFT boosts the Step Success Rate (Step SR) from 37.9% to 42.2%. For the Two-turn Dialogue Unified model, RFT provides an even larger boost, from 46.5% to 52.4%. These results confirm that our synergistic RFT approach, which uses filtering success as a reward, effectively optimize the Planner's policy for both training paradigms.

Filtering Recall Analysis. To evaluate the effectiveness of the scoring programs generated by the Programmatic Element Filter, we measured the Recall@N performance across various backbone models, as shown in Figure 3. The results clearly indicate that our fine-tuned models significantly outperform the zero-shot GPT models at all values of N. Specifically, both our fine-tuned Qwen2.5-0.5B and 3B models achieve a recall rate of over 90% when considering just the top 3 candidates ($N = 3$), and approach 95% at $N = 5$. In contrast, the powerful GPT-4o model only reaches approximately 72% recall at $N = 3$ and ends at around 86% at $N = 20$. A particularly noteworthy finding

Method	Recall@20	Grounding Accuracy (%)
GT Task + Original HTML (No Pruning)		
Qwen2.5VL-3B-instruct (FT)	–	46.80
Oracle Pruning (GT guaranteed in top 20 candidates)		
GPT-4o	–	82.83
GPT-4o-mini	–	75.39
Qwen2.5VL-3B-instruct (ZS)	–	11.99
Qwen2.5VL-7B-instruct (ZS)	–	12.08
Qwen2.5VL-3B-instruct (FT)	–	**90.28**
GT Task + End-to-End LLM Pruning & Decision		
GPT-4o	85.56	70.84
GPT-4o-mini	89.19	67.57
GT Task + Prune4Web's Programmatic Element Filter Pruning		
GPT-4o	85.56	80.65
GPT-4o-mini	89.19	73.75
Qwen2.5-0.5B-instruct (FT)	**97.64**	**88.28**
Qwen2.5VL-3B-instruct (FT)	97.46	**88.28**

Table 2: Performance with Ground-truth (GT) low-level subtasks on our custom grounding benchmark (1101 trajectories), evaluating Programmatic Element Filter and Action Grounder capabilities under various conditions. Recall@20 indicates the percentage of times the ground-truth element is successfully included within the top 20 candidates after the filtering stage. ZS denotes zero-shot, and FT denotes fine-tuning. Top-1 accuracy is indicated by **bold text**.

Filtering Method	GPT-4o	GPT-4o-mini	Qwen2.5 VL-3B
LLM Top-N Selection	42.1	26.3	0.0
Prune4Web Filtering	**42.1**	**31.6**	**5.2**

Table 3: Ablation on Programmatic DOM Filtering (LLM-Verified Task Completion Rate %).

Architecture	GPT-4o	GPT-4o-mini
Action Grounder Only	36.8	21.1
Planner + Action Grounder	42.1	26.3
Full Framework (Prune4Web)	**42.1**	**31.6**

Table 4: Ablation on multi-stage framework (LLM-Verified Task Completion Rate %).

Training Strategy	Framework	Step SR (%)
SFT Only	Separate Models	37.9
SFT + RFT	Separate Models	**42.2**
SFT Only	Two-turn Dialogue	46.5
SFT + RFT	Two-turn Dialogue	**52.4**

Table 5: Ablation on training strategies (Step SR: Step Success Rate %) with Qwen2.5VL-3B-instruct on offline Multimodal-Mind2Web Cross-Task subset.

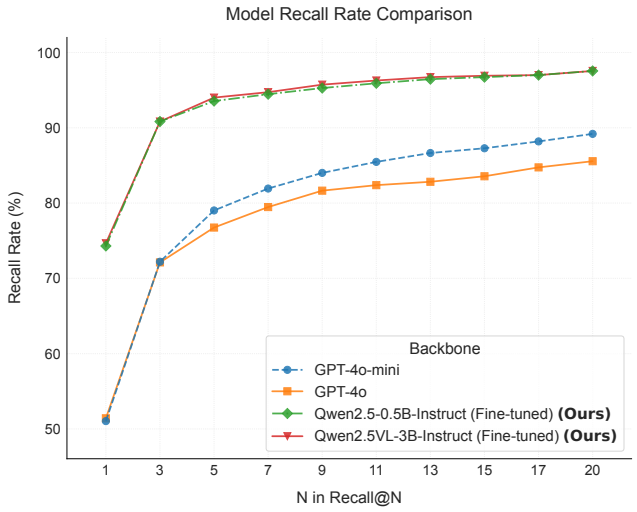

Figure 3: Recall@N performance of our programmatic filtering stage with different backbone models. The y-axis represents the percentage of times the ground-truth element was successfully included in the Top-N candidates.

is that our fine-tuned 0.5B model performs almost identically to the 3B model. This demonstrates that our DOM Tree Pruning Programming paradigm effectively distills the complex filtering task into a simple program generation problem that can be mastered even by smaller, more efficient models. This high recall with a small N is crucial, as it provides the downstream Action Grounder with a small, high-quality set of candidates, significantly reducing the difficulty of the final grounding step.

5 Conclusion

This paper addressed the significant challenge of information overload for LLM-based web agents by introducing Prune4Web, a multi-stage architecture based on a Planning $\rightarrow$ Programmatic Filtering $\rightarrow$ Action Grounding workflow and the core method of **DOM Tree Pruning Programming**. Our key innovation leverages LLMs to generate lightweight, interpretable Python programs that dynamically score and prune DOM elements based on semantic clues from decomposed sub-tasks. This approach eliminates the need for LLMs to process massive DOMs, reducing candidate elements by 25$\sim$50 times while maintaining high accuracy. Our automated data annotation pipeline provides supporting data for training our model. Additionally, our two-turn dialogue training strategy jointly trains the Planner, Filter and Grounder as a unified model. This training approach combines SFT with a targeted RFT that uses intermediate filtering results as reward signals for the upstream Planner, significantly improving the model's strategic task decomposition capabilities. In conclusion, Prune4Web offers an effective and efficient solution to information overload through its innovative programmatic pruning paradigm and synergistic training strategy, laying a solid foundation for more accurate and reliable next-generation web automation systems.

6 Acknowledgments

This work was supported in part by National Natural Science Foundation of China (No.62461160331, No.62132001, No.62572039), in part by Huawei-BUAA Joint Lab, in part by the Fundamental Research Funds for the Central Universities, and in part by the Young Elite Scientists Sponsorship Program by CAST.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Bai, H.; Zhou, Y.; Pan, J.; Cemri, M.; Suhr, A.; Levine, S.; and Kumar, A. 2025. Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *Advances in Neural Information Processing Systems*, 37: 12461–12495.

Bai, J.; Bai, S.; Yang, S.; Wang, S.; Tan, S.; Wang, P.; Lin, J.; Zhou, C.; and Zhou, J. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 1(2): 3.

Chen, X.; Li, H.; Liang, J.; Jiang, S.; and Yang, D. 2024. Edge: Enhanced grounded gui understanding with enriched multi-granularity synthetic data. *arXiv preprint arXiv:2410.19461*.

Cheng, K.; Sun, Q.; Chu, Y.; Xu, F.; Li, Y.; Zhang, J.; and Wu, Z. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. *arXiv preprint arXiv:2401.10935*.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2023a. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36: 28091–28114.

Deng, X.; Gu, Y.; Zheng, B.; Chen, S.; Stevens, S.; Wang, B.; Sun, H.; and Su, Y. 2023b. Mind2Web: Towards a Generalist Agent for the Web. In *Proceedings of NeurIPS*.

Furuta, H.; Lee, K.-H.; Nachum, O.; Matsuo, Y.; Faust, A.; Gu, S. S.; and Gur, I. 2023. Multimodal web navigation with instruction-finetuned foundation models. *arXiv preprint arXiv:2305.11854*.

Gou, B.; Wang, R.; Zheng, B.; Xie, Y.; Chang, C.; Shu, Y.; Sun, H.; and Su, Y. 2024a. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.

Gou, B.; Wang, R.; Zheng, B.; Xie, Y.; Chang, C.; Shu, Y.; Sun, H.; and Su, Y. 2024b. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Gupta, T.; and Kembhavi, A. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14953–14962.

Gur, I.; Furuta, H.; Huang, A.; Safdari, M.; Matsuo, Y.; Eck, D.; and Faust, A. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

He, H.; Yao, W.; Ma, K.; Yu, W.; Dai, Y.; Zhang, H.; Lan, Z.; and Yu, D. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

Hong, W.; Wang, W.; Lv, Q.; Xu, J.; Yu, W.; Ji, J.; Wang, Y.; Wang, Z.; Dong, Y.; Ding, M.; et al. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 14281–14290.

Hu, S.; Tu, Y.; Han, X.; He, C.; Cui, G.; Long, X.; Zheng, Z.; Fang, Y.; Huang, Y.; Zhao, W.; et al. 2024. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Jiang, J.; Wang, F.; Shen, J.; Kim, S.; and Kim, S. 2024a. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Jiang, X.; Dong, Y.; Wang, L.; Fang, Z.; Shang, Q.; Li, G.; Jin, Z.; and Jiao, W. 2024b. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7): 1–30.

Kerboua, I.; Shayegan, S. O.; Thakkar, M.; Lù, X. H.; Caccia, M.; Eglin, V.; Aussem, A.; Espinas, J.; and Lacoste, A. 2025. LineRetriever: Planning-Aware Observation Reduction for Web Agents. *arXiv preprint arXiv:2507.00210*.

Lai, H.; Liu, X.; Iong, I. L.; Yao, S.; Chen, Y.; Shen, P.; Yu, H.; Zhang, H.; Zhang, X.; Dong, Y.; et al. 2024. AutoWebGLM: A Large Language Model-based Web Navigating Agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 5295–5306.

Lin, K. Q.; Li, L.; Gao, D.; Yang, Z.; Wu, S.; Bai, Z.; Lei, W.; Wang, L.; and Shou, M. Z. 2024. Showui: One vision-language-action model for gui visual agent. *arXiv preprint arXiv:2411.17465*.

Lù, X. H.; Kasner, Z.; and Reddy, S. 2024. Weblinx: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

Lu, Z.; Chai, Y.; Guo, Y.; Yin, X.; Liu, L.; Wang, H.; Xiao, H.; Ren, S.; Xiong, G.; and Li, H. 2025. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*.

Ma, K.; Zhang, H.; Wang, H.; Pan, X.; Yu, W.; and Yu, D. 2023. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arXiv:2309.08172*.

Pan, Y.; Kong, D.; Zhou, S.; Cui, C.; Leng, Y.; Jiang, B.; Liu, H.; Shang, Y.; Zhou, S.; Wu, T.; et al. 2024. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*.

Qi, Z.; Liu, X.; Iong, I. L.; Lai, H.; Sun, X.; Zhao, W.; Yang, Y.; Yang, X.; Sun, J.; Yao, S.; et al. 2024. WebRL: Training LLM Web Agents via Self-Evolving Online Curriculum Reinforcement Learning. *arXiv preprint arXiv:2411.02337*.

Qiao, S.; Zhang, N.; Fang, R.; Luo, Y.; Zhou, W.; Jiang, Y. E.; Lv, C.; and Chen, H. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. *arXiv preprint arXiv:2401.05268*.

Qin, Y.; Zhou, E.; Liu, Q.; Yin, Z.; Sheng, L.; Zhang, R.; Qiao, Y.; and Shao, J. 2024. Mp5: A multi-modal open-ended embodied system in minecraft via active perception. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16307–16316. IEEE.

Shao, Z.; Wang, P.; Li, S.; Zhang, Z.; Jin, P.; Liu, Z.; Chen, R.-Z.; Tu, Y.; Liu, S.; Wang, C.; et al. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.01339*.

Shen, J.; Jain, A.; Xiao, Z.; Amlekar, I.; Hadji, M.; Podolny, A.; and Talwalkar, A. 2024. ScribeAgent: Towards Specialized Web Agents Using Production-Scale Workflow Data. *arXiv preprint arXiv:2411.15004*.

Song, Y.; Xu, F. F.; Zhou, S.; and Neubig, G. 2025. Beyond Browsing: API-Based Web Agents. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics: ACL 2025*, 11066–11085. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Tan, W.; Zhang, W.; Xu, X.; Xia, H.; Ding, Z.; Li, B.; Zhou, B.; Yue, J.; Jiang, J.; Li, Y.; et al. 2024. Cradle: Empowering foundation agents towards general computer control. *arXiv preprint arXiv:2403.03186*.

Wang, W.; Gao, Z.; Chen, L.; Chen, Z.; Zhu, J.; Zhao, X.; Liu, Y.; Cao, Y.; Ye, S.; Zhu, X.; et al. 2025. VisualPRM: An Effective Process Reward Model for Multimodal Reasoning. *arXiv preprint arXiv:2503.10291*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35: 24824–24837.

Wen, H.; Li, Y.; Liu, G.; Zhao, S.; Yu, T.; Li, T. J.-J.; Jiang, S.; Liu, Y.; Zhang, Y.; and Liu, Y. 2024. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 543–557.

Wu, Z.; Han, C.; Ding, Z.; Weng, Z.; Liu, Z.; Yao, S.; Yu, T.; and Kong, L. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.

Xie, T.; Zhang, D.; Chen, J.; Li, X.; Zhao, S.; Cao, R.; Toh, J. H.; Cheng, Z.; Shin, D.; Lei, F.; et al. 2025. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37: 52040–52094.

Xu, Y.; Wang, Z.; Wang, J.; Lu, D.; Xie, T.; Saha, A.; Sahoo, D.; Yu, T.; and Xiong, C. 2025. Aguvis: Unified Pure Vision Agents for Autonomous GUI Interaction. In *Proceedings of ICML*.

Xue, T.; Qi, W.; Shi, T.; Song, C. H.; Gou, B.; Song, D.; Sun, H.; and Su, Y. 2025. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*.

Yao, S.; Chen, H.; Yang, J.; and Narasimhan, K. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35: 20744–20757.

Zhang, C.; Li, L.; He, S.; Zhang, X.; Qiao, B.; Qin, S.; Ma, M.; Kang, Y.; Lin, Q.; Rajmohan, S.; et al. 2024. Ufo: A ui-focused agent for windows os interaction. *arXiv preprint arXiv:2402.07939*.

Zhang, C.; Yang, Z.; Liu, J.; Han, Y.; Chen, X.; Huang, Z.; Fu, B.; and Yu, G. 2023a. Appagent: Multimodal agents as smartphone users. *arXiv preprint arXiv:2312.13771*.

Zhang, S.; Chen, Z.; Shen, Y.; Ding, M.; Tenenbaum, J. B.; and Gan, C. 2023b. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

Zheng, B.; Gou, B.; Kil, J.; Sun, H.; and Su, Y. 2024a. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

Zheng, Y.; Zhang, R.; Zhang, J.; Ye, Y.; Luo, Z.; Feng, Z.; and Ma, Y. 2024b. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Bangkok, Thailand: Association for Computational Linguistics.

Zhou, E.; An, J.; Chi, C.; Han, Y.; Rong, S.; Zhang, C.; Wang, P.; Wang, Z.; Huang, T.; Sheng, L.; et al. 2025a. RoboRefer: Towards Spatial Referring with Reasoning in Vision-Language Models for Robotics. *arXiv preprint arXiv:2506.04308*.

Zhou, E.; Su, Q.; Chi, C.; Zhang, Z.; Wang, Z.; Huang, T.; Sheng, L.; and Wang, H. 2025b. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 6919–6929.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.