Efficient Web-Based LLM Agents: Dynamic Prompting for Faster Inference with Web Context Adaptation

Bingyan Liu, Zhenghao Gong, Stephanie Wang

1 Introduction

Large model agents are among the most promising applications of large language models (LLMs) in downstream tasks. These agents leverage the reasoning, planning, and interactive capabilities of LLMs to autonomously perform complex tasks on the web, such as information retrieval, task automation, and decision support. However, despite their potential, significant challenges still hinder their practical deployment. A key issue lies in balancing the trade-offs between inference cost, response efficiency, and agent effectiveness, particularly in constrained environments where context length and reasoning time are limited. Achieving an efficient response to web environments and specific tasks in such restricted settings is an open challenge in the development of scalable and practical agents.

To address these challenges, our approach focuses on two primary methods: optimizing web element processing and developing an adaptive prompting strategy. Firstly, we use a combination of computer vision techniques, specifically OpenCV, and filtering mechanisms to efficiently identify and locate relevant web elements. By combining visual analysis with HTML element filtering, we significantly reduce the context overhead, thereby optimizing both loading time and the complexity of the elements processed by the agent.

Secondly, we develop a dynamic prompting strategy that adjusts prompting techniques—including Planning, Acting, Memory, and Reflection—based on the task context and the environment. This adaptive approach ensures a balance between inference time and context length, allowing the agent to dynamically manage its reasoning depth and context usage depending on the specific requirements of the task. Our method draws inspiration from the "Tree of Thought" (ToT) structure, employing backpropagation and exploration strategies to enhance stability and task success rates. By dynamically selecting prompting strategies according to the current state, we extend the agent's ability to handle complex tasks effectively while reducing computational costs and response time.

Our work aims to pave the way for the practical deployment of large model agents in environments with limited computational resources, improving their ability to interact efficiently with diverse web content while maintaining accuracy and adaptability.

2 Literature Survey

Prompt engineering. Prompt engineering has become a cornerstone in enhancing the reasoning and interaction capabilities of large language models (LLMs). Early approaches focused on static prompting techniques, where prompts were manually or semi-automatically designed for specific downstream tasks. For instance, COT [1] generates intermediate reasoning steps to enhance the model's problem-solving capabilities, but it lacks dynamic adaptability and cannot interact with external environments. TOT [2] improves in-task exploration and backtracking which demonstrates a greater dynamic interaction potential. However, it requires more computational resources and may introduce latency in real-time applications. ReACT [3] significantly boosts model usability and interaction quality by generating reasoning trajectories that can interact with the external environment. SELF-DISCOVER[4] explores the construction of self-exploratory reasoning structures and utilizes

these for deeper reasoning processes. LATS[5] employs Monte Carlo tree search to construct a Language Agent Tree Search framework, offering an efficient framework that integrates reasoning, exploration, and planning. These methods demonstrate substantial progress in task-specific reasoning and interaction, but their reliance on static prompts tailored to predefined scenarios limits their scalability and adaptability in dynamic and heterogeneous environments, such as web-based tasks.

Dynamic prompting. Dynamic prompting emerges as a solution, introducing mechanisms to adjust prompt complexity, length, and functionality based on task-specific and contextual demands. This paper [6] first proposes dynamically adjusting prompt features (e.g., position, length, representation) based on task characteristics to achieve improved task accuracy and efficiency. However, its experiments are predominantly limited to NLP tasks and restricted to multi-modal web environments. Similarly, Instance Dependent Prompt Generation (IDPG) [7] extends the dynamic prompt concept by tailors prompts to individual instances. It leverages up-and-down adapter modules and parameterized hypercomplex multiplication (PHM) layers to dynamically generate prompts based on the input instance's unique characteristics. This ensures contextually relevant and task-adaptive prompt that the model can handle highly diverse inputs and reduce computation efficiency, but its performance tied to the quality of task-specific data which limit its generalizability to new or underrepresented domains.

The challenges posed by web-based environments make them a compelling area for further research. Web tasks, such as navigation, information extraction, and summarization, often involve complex interactions with multi-modal content (e.g., text, images, and UI elements). Existing static prompting strategies struggle to adapt to such scenarios, while dynamic prompting offers a potential solution by enabling prompts to align with the varying demands of web tasks. For example, WebVoyager [8] showcases the use of a local task pool for navigation planning and employs the multimodal capabilities of GPT-4v for automatic evaluation. However, it relies on predefined strategies, highlighting the need for more adaptive mechanisms to handle task variations effectively. Building on the advancements in dynamic prompting, our research aims to address its application in web agents. We seek to develop a framework that dynamically adjusts prompting strategies based on task-specific characteristics and webpage features to improve task accuracy and efficiency in constrained environments. These efforts are expected to enhance the adaptability and responsiveness of web agents to achieve more effective language model-based systems in diverse and dynamic web scenarios.

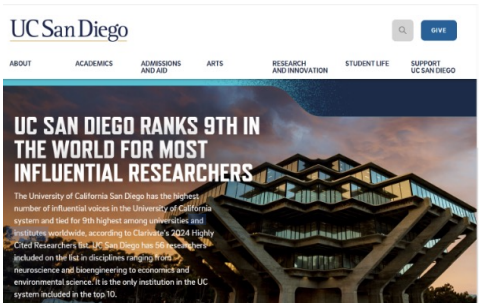
3 Methods

We plan to create and optimize our web agent through two main approaches: obtaining and locating web elements, and developing an effective prompting strategy, to ensure efficient response of the web agent in limited context and restricted reasoning time scenarios.

In terms of obtaining and locating web elements, we combine computer vision techniques with HTML-based analysis. The advantage of using computer vision in web agent tasks lies in preserving the rendered, more structured, and intuitive visual information of the webpage, while HTML elements provide precise positioning and element type details. For example, the type and ARIA label of a web element (such as its type and aria-label attribute) often give the model effective cues about the element's function and the best way to interact with it. To achieve this, we have adopted the following optimization strategies:

Firstly, when computer vision detects that the webpage has loaded, we take a snapshot of the current web elements to avoid unnecessary time consumption caused by waiting for the entire page to fully load. For the collected web elements, we filter them based on relevance using inner_text, type, label, and other textual information, while also defining a set of interactive elements (such as buttons, text boxes, and hyperlinks). By doing this, we avoid the overhead of including the entire web page in the prompt, and instead only collect relevant elements, assigning them IDs, functions, and inner_text to simplify the original information. This method effectively optimizes the context length occupied by web page elements in the agent task.

Additionally, we utilize computer vision techniques to assist in the process. We apply background subtraction and edge detection algorithms to locate potential icon positions, alongside Optical Character Recognition (OCR) to identify areas containing text. The elements' positions determined by computer vision are cross-verified with those obtained from HTML elements, allowing us to retain only high-confidence results. This dual-validation approach enables us to more accurately obtain and locate web elements, thereby providing a more efficient foundation for the web agent.

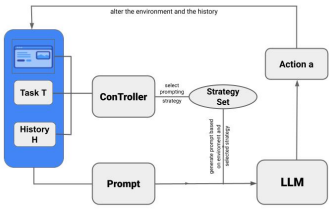


(a) Example Webpage

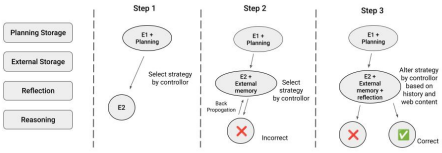

```
"{'id': 0, 'type': 'link', 'link': '/index.html',
'innertext': 'Navigate Back to Home'}",
 "{'id': 1, 'type': 'img', 'innertext': 'UC San
Diego'}",
 "{'id': 5, 'type': 'button'}",
 "{'id': 6, 'type': 'img', 'innertext':
'search-icon'}",
 "{'id': 7, 'type': 'link', 'link':
'https://giving.ucsd.edu/', 'innertext': 'Give to UC
San Diego'}",
 "{'id': 8, 'type': 'link', 'link':
'https://admission.universityofcalifornia.edu/how-to-ap
ply/apply-online/', 'innertext': 'Apply to UC San
Diego'}",
 "{'id': 10, 'type': 'input'}",
 ...
```


(b) Extracted Web Elements

Figure 1: Webpages to Text Example



(a) Prompting Workflow



(b) Tree of Thought in Inferencing

Figure 2: Dynamic Prompting Structures (Side-by-side)

In the context of web-based agent tasks, where external environments are often characterized by dense information and context-specific challenges, we propose a dynamic and adaptive prompting strategy. This approach is designed to balance task accuracy, context length, and inference time by tailoring prompt functionality and complexity to meet the specific demands of the task and the structure of web elements. Our strategy achieves this balance by integrating key functionalities that enhance the agent's flexibility, precision, and reasoning capacity. These functionalities include:

1. External Memory: By leveraging relevant historical interactions, the system builds continuity in its reasoning process, enabling it to incorporate past knowledge for better decision-making.
2. Self-Reflection: After generating instructions or outputs, the agent evaluates their validity and coherence. This built-in quality control mechanism reduces errors and improves reliability.
3. Planning: Before execution, the agent generates a structured and actionable plan based on the task requirements and the current environment. This ensures purposeful, step-by-step execution aligned with the overall objective.
4. Reasoning: The agent provides explanatory reasoning alongside responses, enhancing its ability to navigate complex decision-making processes and deliver more logical, nuanced outputs.

Our prompting strategy is based on a Tree of Thought (ToT)-like search tree structure and designed as a dynamic and adaptive framework for efficiently handling complex web-based agent tasks. By introducing backpropagation, the system can recover from errors, such as function call failures or incorrect page navigation, by backtracking to the previous step and continuing the task. This approach not only enhances the model's stability but also improves its capacity to explore diverse environments effectively. Unlike traditional ToT implementations, which often rely on pre-built controller models to manage backtracking, our method trains a controller model to dynamically select optimal prompting strategies. The controller determines the appropriate strategies (e.g., External Memory, Reflection, Planning, Reasoning) based on the current environment, historical data, and task context. This allows us to extend ToT's exploration and backtracking capabilities while improving the model's ability to handle complex tasks and logical reasoning. It excels in dynamically selecting the most suitable

approach for various tasks, optimizing both inference efficiency and task accuracy. In environments where web elements are cluttered or context size is high, the controller employs Input Filtering to extract only task-relevant elements, reducing unnecessary overhead and ensuring efficient processing. For complex tasks that demand deep reasoning, it utilizes strategies like ReAct, which follows an iterative "observation-thought-action" loop to enhance logical reasoning before generating outputs. On the other hand, for straightforward tasks such as simple navigation or clicking specific elements, the controller avoids additional strategies to maximize inference speed and computational efficiency. This flexibility enables the strategy to adapt seamlessly across a wide range of task complexities and environmental conditions. Additionally, the controller is highly extensible, allowing integration with external tools like OCR systems or search engine APIs to further enhance its capabilities. Compared to existing methods such as LATS, our approach significantly reduces redundant reasoning and computational overhead, streamlining task completion while maintaining high accuracy. By striking a balance between efficiency and adaptability, our strategy ensures optimal performance even in resource-constrained environments.

To enable dynamic prompting, we propose a standalone controller model trained on data collected from ToT-based agents operating across diverse websites and tasks. This data includes two types of function sequences:

- S1: Function sequences generated by the model.
- S2: Human-annotated sequences representing the optimal approach to completing tasks.

The agent's state is defined as

$$s = WebContext(C1) + History(H) + TaskContext(C2). \tag{1}$$

which combines information about the current web elements, historical interactions, and task-specific semantics. Based on this state, the controller selects actions a=SelectPrompt(FC), representing the optimal prompting strategies or combinations thereof. For example, when the history indicates frequent backtracking, the controller might prioritize Reflection + External Memory to reduce errors. For complex tasks with limited historical context, it may choose Planning + Reasoning to enhance logical reasoning and execution. The reward function balances task efficiency and accuracy, expressed as:

$$R(S1, S2) = \frac{LCS(S1, S2)}{S2}(1 - \lambda\frac{S1}{S2}) \tag{2}$$

The model is trained to learn the optimal strategy distribution, represented as $\pi(a \mid s)\pi(a \mid s)$, enabling it to dynamically adjust its prompting strategies based on the current state. To achieve this, we propose method based on these three complementary approaches:

1. Deep Learning/Reinforcement Learning: This approach integrates features derived from web elements, historical interactions, and task-related NLP semantics, along with statistical information such as element types and counts. The model outputs a normalized variable of length N(representing the number of available strategies) that enables dynamic selection of one or more strategies by applying adjustable thresholds.
2. Large Language Model (LLM)-Based: BThis method leverages the flexibility of LLMs to process web elements, historical context, and task descriptions directly, generating a list of optimal prompting strategies. While highly adaptable, this approach incurs additional inference time overhead.
3. Similarity Search: Using pretrained embedding models, this approach identifies the k-most similar tasks and environments from the history. These similar examples provide a basis for predicting the optimal strategy combination which offer an efficient solution for scenarios with recurring patterns or well-understood contexts.

By integrating these methods, our framework ensures robust and adaptive performance in a variety of web-based tasks and environments, dynamically adjusting its strategy according to the complexity and requirements of each situation.

4 Experiments

To evaluate the effectiveness of our proposed dynamic prompting strategy, we conducted preliminary experiments to analyze the impact of different strategies on model performance within a search tree

framework. These experiments were aimed at answering the following key questions: What is the impact of different strategy-selection methods on task accuracy? How does the dynamic-strategy controller compare to fixed-strategy and no-strategy controllers in terms of efficiency? What is the trade-off between task accuracy and inference time when using strategies like reasoning or input filtering? Our testbed operates in real-world web environments using a search-tree-based framework. Each node in the tree stores the current web elements, historical actions and strategies, and the subtasks associated with that state. The framework comprises two main components:

1. LLM Agent: Executes predefined actions based on the generated prompt.
2. Controller Model: Processes the current node's information—including web elements, historical actions, and subtasks—and outputs a strategy list. This strategy list is combined with the node information to generate a prompt for the LLM, which then calls one of the following predefined functions:
    - "click": ["id": "integer", "Click on the element with a specific ID."]
    - "typing": ["id": "integer", "text": "string", "Click on the element and type the specified text."]
    - "back": [None, "Navigate back to the previous page."]
    - "response": ["text": "string", "Respond to the user, report the action taken, or answer the user's question."]

Our experiments used Google's homepage (https://google.com) as the initial state. The LLM Agent was implemented using Arcee Agent, a 7-billion-parameter open-source language model designed for function calling and tool-based tasks. Arcee Agent was selected for its ability to minimize hallucinations and operate effectively under constrained GPU resources. We tested three types of controller models to evaluate different strategy-selection methods:

1. No-Strategy Controller: Does not apply any strategies at any node and directly generates actions to complete tasks.
2. Fixed-Strategy Controller: Applies a fixed Reasoning strategy at every node, utilizing the Re-Act framework for iterative observation, reasoning, and action.
3. Dynamic-Strategy Controller: Adapts its strategy based on the web environment. For simple environments, it applies no strategy; for complex environments, it uses a combination of Input Filtering and Reasoning strategies to optimize performance.

Due to limited training data and time, we used a rule-based controller that classified environments as either "simple" or "complex" based on basic metrics, such as the number of web elements. The experiments were designed around three types of tasks: clicking on specific elements, retrieving information such as the Wikipedia description for "Chongqing," and searching for products like tables on IKEA's website. These tasks reflected varying levels of complexity, ranging from simple environments with straightforward navigation to complex scenarios involving cluttered web elements or nested structures. We also incorporated long-sequence tasks requiring multiple steps to test the agent's ability to maintain context and adapt strategies dynamically over extended interactions. During the experiments, we measured performance using two metrics: hit rate and average inference time. The hit rate represents the percentage of successfully accessed valid web pages required to complete a task. For example, retrieving a Wikipedia description involves visiting pages like google.com/search?q and wikipedia.org. To calculate the hit rate, we logged all the web pages necessary to complete each task during data collection and compared them with the pages visited by the agent during testing:

$$HitRate = \frac{ValidPagesVisited}{RequiredPages} \tag{3}$$

The average inference time measures the time taken by the agent to perform a single action. This metric is influenced by the selected strategies. For instance, strategies such as summarization, which increase the output context length, can enhance task accuracy but also lead to higher inference times.

Table 1,2,3 are our tests of three basic tasks, we conducted 10 test runs with each agent configuration and calculated the average results. To ensure efficiency, we set a maximum action limit of 7; if an agent failed to complete a task within seven steps, the task was terminated early.

Table 1: Search for wiki description for Chongqing

	Strategy				
	No Strategy	Summarization	Reasoning	Summarization + Reasoning	Dynamic Strategy
Hit rate	50%	50%	50%	47.6%	60%
Time use	7.82	12	9.42	10.1	9.32

Table 2: Search for an IKEA desk

	Strategy				
	No Strategy	Summarization	Reasoning	Summarization + Reasoning	Dynamic Strategy
Hit rate	43%	46.2%	46.2%	46.2%	46.2%
Time use	8.2	13	10.8	11.1	10.6

Table 3: Find the release date of a popular movie

	Strategy				
	No Strategy	Summarization	Reasoning	Summarization + Reasoning	Dynamic Strategy
Hit rate	55%	55%	55%	50%	60%
Time use	7.2	11	8.9	10.5	9.2

Our experiments demonstrated clear performance differences between the three controller configurations. The No-Strategy Controller performed adequately in simple environments but struggled in complex tasks, often failing to navigate efficiently or interact correctly with relevant web elements. The Fixed-Strategy Controller achieved higher accuracy in complex tasks but incurred significant inference time overhead due to indiscriminate reasoning, which sometimes resulted in lower accuracy due to context length issues. In contrast, the Dynamic-Strategy Controller struck an optimal balance, leveraging input filtering and reasoning for complex environments while avoiding unnecessary overhead in simpler tasks. This adaptability allowed it to achieve comparable performance to the Fixed-Strategy Controller in challenging scenarios, while significantly reducing computational costs. These findings underscore the importance of context-aware dynamic strategy selection for optimizing agent performance.

Our experiments revealed significant performance differences among the three controller configurations. The no-strategy controller performed well in simple environments but struggled with complex tasks, often failing to navigate effectively or interact correctly with relevant web elements. The fixed-strategy controller achieved higher accuracy in complex tasks but incurred additional inference overhead due to indiscriminate reasoning across varying environments, with performance heavily influenced by the type of web environment. Integrating multiple strategies did not lead to further improvements; instead, the increased context length from excessive prompting strategies caused "memory loss," resulting in reduced accuracy. In contrast, the dynamic-strategy controller achieved the best balance, leveraging input filtering and reasoning in complex environments while avoiding unnecessary overhead in simpler tasks and keeping prompt context at an appropriate length. This adaptability allowed it to perform comparably to the fixed-strategy controller in challenging scenarios while significantly reducing computational costs. These findings highlight the importance of context-aware dynamic strategy selection in optimizing agent performance.

5 Conclusion

The results of our study demonstrate the clear advantages of employing a dynamic prompting strategy for web-based agents powered by large language models (LLMs). Compared to static or no-strategy approaches, our framework demonstrates superior adaptability and efficiency across diverse tasks. By dynamically adjusting reasoning depth, memory usage, and planning based on task complexity, the dynamic strategy achieves higher task accuracy while maintaining a balance between computational

cost and response time. For example, it consistently outperformed static strategies in complex environments, achieving approximately 10% higher hit rates and reducing inference time by up to 20%. These improvements make the system suitable for real-world applications where speed and accuracy are critical, even in resource-constrained settings.

However, although the results are encouraging, our framework still has certain limitations. One notable limitation stems from the lack of generalization caused by missing visual capabilities. Despite incorporating computer vision methods such as background segmentation and OCR, the framework provides only limited web interaction capabilities. When dealing with tasks that require understanding visual content, such as image links, the accuracy of the framework drops significantly. Moreover, while our model is more computationally efficient compared to static complex strategies, tasks involving complex web content still incur noticeable time overhead. This overhead mainly arises from extensive content filtering, including filtering based on web element types, inner text, and links. A significant portion of the overhead is introduced by the controller model selecting inputs for filtering, followed by the large model's additional inference cost to pre-screen relevant elements. Another major limitation is the model's unstable performance. Although the dynamic prompting framework outperformed static strategies across several websites in experiments, the framework struggles with high variance in accuracy across different websites, reflecting relative instability and sensitivity to environmental changes. Generally, while we have partially achieved a balance between inference time and performance and demonstrated that the dynamic prompting strategy framework outperforms single static strategies in web agent tasks, issues such as performance fluctuation with environmental and task changes persist. The stability of web agents remains a challenge to be addressed.

To address these limitations, future efforts will focus on improving the model's stability, scalability, and visual capabilities. This includes training on a broader dataset of web layouts and integrating reinforcement learning to enable real-time adaptation in unfamiliar environments. Unlike the simple logic-based models used in preliminary experiments, we plan to employ deep learning-based models as the controller model to enhance stability. Additionally, to efficiently collect large-scale data and train the controller model, we intend to adopt an online learning approach. In live environments, visual large models like GPT-4 Vision will supervise and evaluate the performance of our web agent framework, enabling automated data collection and training without human intervention. Another priority is fine-tuning large visual models with web agent capabilities to address the current framework's significant shortcomings in visually dependent web environments. We plan to build on open-source visual models like LLaVA and Olmo, fine-tuning stable versions for agent function calls and integrating them into our framework. Lastly, we will expand experiments to cover various web domains, such as e-commerce and technical documentation, to validate the model's effectiveness in handling more complex and unstructured scenarios.

References

[1] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, Denny Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. arXiv:2201.11903v6.

[2] Jieyi Long. *Large Language Model Guided Tree-of-Thought*. arXiv:2305.08291v1.

[3] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv:2210.03629v3.

[4] Pei Zhou, Jay Pujara, Xiang Ren, Xinyun Chen, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, Denny Zhou, Swaroop Mishra, Huaixiu Steven Zheng. *Self-Discover: Large Language Models Self-Compose Reasoning Structures*. arXiv:2402.03620v1.

[5] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, Yu-Xiong Wang. *Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models*. arXiv:2310.04406v3.

[6] Xianjun Yang, Wei Cheng, Xujiang Zhao, Wenchao Yu, Linda Petzold, Haifeng Chen. *Dynamic Prompting: A Unified Framework for Prompt Tuning*. arXiv:2303.02909.

[7] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Rui Hou, Yuxiao Dong, V.G.Vinod Vydiswaran, Hao Ma. *IDPG: An Instance-Dependent Prompt Generation Method*. arXiv:2204.04497.

[8] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, Dong Yu. *WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models*. arXiv:2401.13919.