DEPART: A Hierarchical Multi-Agent System for Multi-Turn Interaction

Hao-Lun Hsu[*] [†], **Jing Xu**[‡], **Nikhil Vichare**[‡]
Francesco Carbone[‡], **Miroslav Pajic**[*], **Giuseppe Carenini**[‡] [§]

Abstract

Large Language Models (LLMs) perform well on short-horizon tasks but struggle with long-horizon, multimodal scenarios that require multi-step reasoning, perception, and adaptive planning. We identify two key challenges in these settings: the difficulty of long-term coordination between planning and execution within single-agent architectures and the inefficiency of indiscriminate visual grounding. To address this, we propose **DEPART**, a hierarchical multi-agent framework that separates planning, perception, and execution across three specialized agents. These agents collaborate through an iterative loop: **D**ivide, **E**valuate, **P**lan, **A**ct, **R**eflect, and **T**rack, supporting dynamic task decomposition, selective vision use, and feedback-driven control. Evaluated on two web-based benchmarks, DEPART outperforms strong baselines, including agents enhanced with reinforcement learning, while improving efficiency through dynamic vision invocation. Our results highlight the value of modular, agent reasoning in complex multimodal tasks.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities across diverse domains, particularly in single-turn, non-interactive tasks such as mathematical reasoning [16]. Large Multimodal Models (LMMs) extend these capabilities by incorporating visual inputs, enabling perception-based reasoning. However, those models still struggle with long-horizon tasks that require multi-step decision-making, cross-modal reasoning, and coherent interaction over extended sequences [29]. These limitations point to a fundamental gap in the algorithmic reasoning abilities of current models, particularly when reasoning must span both time and modalities.

A key challenge lies in the monolithic design of many existing agents, where a single model is responsible for high-level planning, low-level action execution, and multimodal understanding. This tight coupling introduces architectural and cognitive bottlenecks: combining strategic reasoning with fine-grained execution demands competencies that are difficult to align within a single model [27]. Our empirical results confirm this issue: performance declines as task complexity grows even with pure text-based input in Figure 2 in Appendix, highlighting the need to decouple these responsibilities.

Furthermore, LMMs are computationally expensive, and many interaction steps do not require visual input. Including vision unnecessarily can degrade performance due to cross-modal distraction[17], increasing reasoning complexity and inference cost (see Table1). These findings motivate not just modality separation, but a broader architectural shift toward modular, hierarchical reasoning.

To address these challenges, we introduce **DEPART** (see Figure 1), a hierarchical multi-agent framework for long-horizon, multimodal algorithmic reasoning. DEPART decomposes task-solving into an iterative loop of **D**ivide, **E**valuate, **P**lan, **A**ct, **R**eflect, and **T**rack, distributed across three

[*]Duke University; email: {hao-lun.hsu, miroslav.pajic}@duke.edu
[†]Work done during internship at Applied AI, Amazon
[‡]Applied AI, Amazon; email: {xuzjx, nvichare, carbonef}@amazon.com
[§]University of British Columbia; email: carenini@cs.ubc.ca

Multimodal Algorithmic Reasoning Workshop at NeurIPS 2025 (MAR-NeurIPS 2025).

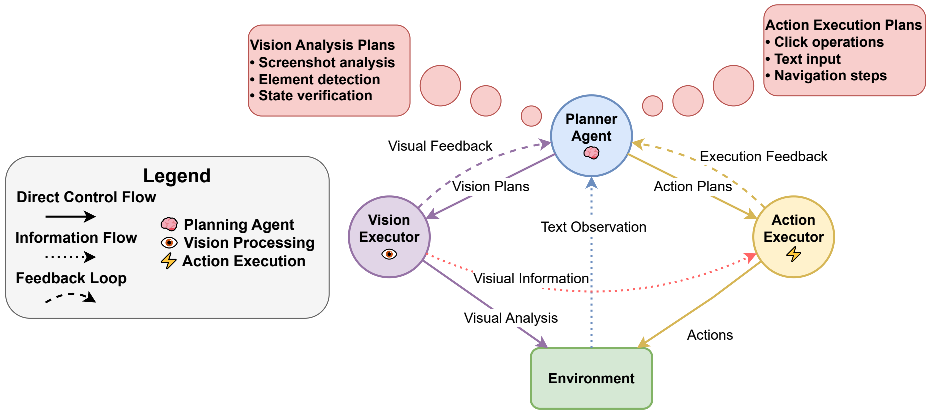

Figure 1: **Overview of DEPART**: The framework (1) divides tasks into distinct modalities and stages to reduce complexity; (2) evaluates the environment and task progress via observations and feedback; (3) plans high-level strategies based on evolving context; (4) acts through specialized executors for grounded interaction and vision understanding; (5) reflects on action outcomes to guide replanning; and (6) tracks global task status and historical context to maintain coherence across multiple turns. For example, in the case of a web agent, the observations $\mathcal{O} = (\mathcal{O}^{text}, \mathcal{O}^{img})$ include both textual and visual information, while the action executor uses Playwright to perform actions such as clicking, typing, and scrolling.

specialized agents: a planner, a vision executor, and an action executor. Through structured feedback and support for retry and replanning [2, 5], DEPART enables robust, adaptive behavior in dynamic environments. By separating planning, vision, and execution into distinct components, it improves reasoning stability, avoids unnecessary modality fusion, and supports modular reuse of pretrained components. Evaluated on WebArena-Lite [32, 12] and VisualWebArena [9], DEPART consistently outperforms strong single-agent and RL-finetuned baselines [14, 24], underscoring the benefits of structured, multi-agent coordination for scalable multimodal reasoning.

2 DEPART: Hierarchical Multi-Agent System

2.1 Problem Formulation

We formulate a web browsing task as a Partially Observable Markov Decision Process (POMDP) to capture the inherent uncertainty and limited observability present in complex and dynamic web environments. A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R})$. The state space $\mathcal{S}$ encompasses the entire internet content, browser context, and user-related metadata. $\mathcal{O}$ denotes the observation space, which may consist of structured text $\mathcal{O}^{text}$ and image elements $\mathcal{O}^{img}$. As the full state $\mathcal{S}$ is prohibitively large and often not fully observable in practice, decision-making is based only on partial observations $\mathcal{O} = (\mathcal{O}^{text}, \mathcal{O}^{img})$. The action space $\mathcal{A}$ includes low-level browser interaction primitives such as clicking, typing, and scrolling. $\mathcal{T}(s_{t+1}|s_t, a_t)$ is the transition function, capturing the environment dynamics given the current state and action. We define a sparse binary reward $r_t \in \{0, 1\}$ at each time step, where the agent receives $r_t = 1$ only if the submitted answer successfully completes the task; otherwise, $r_t = 0$. The interaction process terminates either when an answer is submitted or when a maximum step limit is reached.

2.2 From Single Agent to Multi-Agent

We start with a single-agent setup modeled as a POMDP, where the agent policy π_θ is built on the top of an autoregressive language model. Given a user query $\mathbf{x}$, the agent first generates a high-level multi-step plan $\mathbf{m}$, followed by a response $\mathbf{y}$ that encodes executable actions [15]. The overall plan $\mathbf{m}$ consists of not only the plan steps but also the reasoning to generate the plan. Each action a_t at time step t is deterministically extracted from $\mathbf{y}_t$ via a mapping function f, i.e., $a_t = f(\mathbf{y}_t)$. For instance, if the response includes the span <act >do('Scroll Down') </act >, the resulting action is $a_t =$ do('Scroll Down'). The generation process for a single interaction step can be modeled probabilistically as:

$$\pi_\theta(\mathbf{y}, \mathbf{m}|\mathbf{x}, o) = \pi_\theta(\mathbf{y}|\mathbf{m}, \mathbf{x}, o) \cdot \pi_\theta(\mathbf{m}|\mathbf{x}, o) \tag{1}$$

where $\pi_\theta(\mathbf{y}, \mathbf{m}|\mathbf{x}, o)$ denotes the probability of producing a plan $\mathbf{m}$ and a response $\mathbf{y}$ given input query $\mathbf{x}$ and $o \in \mathcal{O}$. Assuming that $\mathbf{y}$ is conditionally independent of $\mathbf{x}$ given $\mathbf{m}$, the factorization simplifies to:

$$\pi_\theta(\mathbf{y}, \mathbf{m}|\mathbf{x}, o) = \pi_\theta(\mathbf{y}|\mathbf{m}, o) \cdot \pi_\theta(\mathbf{m}|\mathbf{x}, o) \tag{2}$$

This assumption reflects the intuition that once a coherent plan $\mathbf{m}$ is formed, the initial query $\mathbf{x}$ provides no further information required for generating the response $\mathbf{y}$. However, handling both planning and execution within a single model limits scalability and modularity. Inspired by prior work showing that separating these roles improves performance [5, 22] and by hierarchical planning in robotics [30, 7, 23], we propose a hierarchical multi-agent framework with three specialized policies: a planner π_p, a vision executor π_v, and an action executor π_a. The planner generates two high-level plans: $\mathbf{m}^v$ for perception and $\mathbf{m}^a$ for interaction, delegated respectively to the vision and action executors. Formally, the decision process at a single time step is modeled as:

$$\mathbf{y} \sim \pi_a(\mathbf{y}|\mathbf{v}, \mathbf{m}^a, o^{text}) \cdot \pi_v(\mathbf{v}|\mathbf{m}^v, o^{img}) \cdot \pi_p((\mathbf{m}^a, \mathbf{m}^v)|\mathbf{x}, o^{text}) \tag{3}$$

where the planner $\pi_p((\mathbf{m}^a, \mathbf{m}^v)|\mathbf{x}, o^{text})$ generates two distinct plans $\mathbf{m} = (\mathbf{m}^a, \mathbf{m}^v)$ for action and vision executors given query $\mathbf{x}$ and o^{text}. The vision executor then produces vision-derived information $\mathbf{v}$, and the action executor generates the response $\mathbf{y}$.

2.3 Multi-Turn Interactions

To handle long-horizon interactions, we extend this model over T time steps:

$$\mathbf{y}_T \sim \prod_{t=1}^{T} \pi_a\Big(\mathbf{y}_t|\mathbf{v}_t, \mathbf{m}_t^a, o_t^{text}\Big) \cdot \pi_v\Big(\mathbf{v}_t|\,\mathbf{m}_t^v, o_t^{img}\Big) \cdot \pi_p\Big((\mathbf{m}_t^a, \mathbf{m}_t^v)|\mathbf{x}, o_t^{text}, \{\mathbf{v}, \mathbf{m}^a, \mathbf{m}^v, \mathbf{y}\}_{<t}\Big) \tag{4}$$

Execution proceeds in a sequential loop. Given a user query and current text observation $o_t^{text} \in \mathcal{O}^{text}$, the planner generates two parallel high-level plans: one for the vision executor and one for the action executor. If visual input is deemed unnecessary, the planner can omit visual steps entirely to reduce processing overhead. Each plan is dispatched step-by-step to the appropriate executor. When vision is needed, the vision executor processes the page image $o_t^{img} \in \mathcal{O}^{img}$ and shares relevant findings with both the planner and the action executor. The action executor then integrates planner instructions, textual observations, and visual feedback (if any) to perform interactions. The planner monitors execution and textual updates to decide whether to proceed, retry, or replan, maintaining an internal state for adaptive coordination. The overall architecture is shown in Figure 1. In this paper, the framework in this manuscript is implemented by prompting while RL post-training of the policies is left as future work.

3 Experiments

We evaluate DEPART in WebArena [32], a realistic, self-hostable environment with rule-based evaluation across domains. Agents use 12 high-level actions (via Playwright [9]) and receive observations from the accessibility tree, i.e., a compact, semantically rich DOM representation. When needed, visual elements are linked to corresponding image assets for cross-modal grounding. Experiments are conducted on two benchmarks. WebArena-Lite[12] includes 165 primarily text-based tasks where vision is optional but occasionally helpful. VisualWebArena[31] features 114 tasks requiring visual understanding of layout, color, and shape. We assess DEPART under three configurations: single-agent (unified), two-agent (separate planner and executor), and full three-agent (planner, vision, and action executors). Baselines include general-purpose LLMs (Qwen2.5, LLaMA3.1, GPT-4), reasoning-specialized models (QwQ, OpenAI), and RL-finetuned agents such as WebRL and WebAgent-R1 [14, 24]. Full experimental details are provided in Appendix B.

3.1 WebArena-Lite

To ensure comparability with prior work [24, 14], we evaluate DEPART without visual input on the WebArena-Lite benchmark (Table 1). In the single-agent setting, foundation models like Claude

Table 1: Task success rate in WebArena-Lite [32], where the single value in each entry represents the success rate without visual input. For those entry with the format: $X \rightarrow Y$, X denotes the success rate without visual input and Y represents the success rate with visual input (increasing in green and decreasing in red). Avg SR denotes the average success rate over the whole dataset.

Category	Model	Reddit		GitLab		CMS		Map		Shopping		Avg SR	
Fine-tuning	WebRL	**63.2**		46.7		54.3		36.7		31.1		42.4	
	WebAgent-R1	47.4		**56.7**		**57.1**		23.1		44.4		44.8	
1 Agent	Qwen2.5-3B	5.3		13.3		5.7		0.0		4.4		6.1	
	Llama3.1-8B	5.3		10.0		5.7		15.4		8.9		8.5	
	Qwen2.5-32B	10.5		20.0		20.0		19.2		17.8		16.9	
	GPT-4o	10.5		10.0		20.0		20.0		11.1		13.9	
	GPT-4o-Turbo	10.5		16.7		14.3		36.7		13.3		17.6	
	QwQ-32B	15.8		33.3		25.7		15.4		20.0		22.4	
	OpenAI-o3	36.8		46.7		45.7		**38.5**		33.3		39.4	
	OpenAI-o4-mini	47.4		43.3		45.7		26.9		28.9		36.9	
	Qwen3-4B	5.3		0		11.4		8.3		26.7		12.1	
	Claude 3.7	42.1	→ 31.6	16.7	→ 33.3	40.0	→ 0.0	16.7	→ 13.8	57.8	→ 55.6	35.8	→ 27.9
2 Agents	Qwen3-4B	5.3		3.3		14.3		8.3		31.1		14.5	
	Claude 3.7	47.4	→ 42.1	**56.7**	→ 36.7	42.9	→ 31.4	16.7	→ 19.4	**60.0**	→ 44.4	**46.1**	→ 34.5
3 Agents (ours)	Qwen3-4B		10.5		3.3		14.3		13.9		33.3		17.0
	Claude 3.7		42.1		53.3		40.0		33.3		57.8		**46.1**

Table 2: Task success rate in VisualWebArena [9], where the entry with the format: $X \rightarrow Y$, X and Y denote the success rate without visual input and with visual input (increasing in green and decreasing in red). Avg SR denotes the average success rate over the whole dataset.

Category	Model	Reddit		Shopping		Avg SR	
1 Agent	Claude 3.7	15.8	→ 26.3	10.5	→30.2	12.3	→28.9
2 Agents	Claude 3.7	18.4	→ **36.8**	17.1	→**39.5**	18.4	→**38.6**
3 Agents (ours)	Qwen3-4B		5.3		11.8		9.6
	Claude 3.7		34.2		35.5		35.1

3.7 [1] and OpenAI variants achieve over 35% success with carefully designed prompts, while task-specific models with RL post-training like WebRL [14] and WebAgent-R1 [24] exceed 40%. Introducing a two-agent setup (separating planning from execution) consistently improves performance across models. Notably, Claude 3.7 achieves 46.1% in this configuration, outperforming both RL-trained baselines. These results highlight the effectiveness of modular coordination in long-horizon tasks. We further test the impact of adding visual input to single and two-agent setups using Claude 3.7. As shown in Table 1, vision sometimes helps but often harms performance due to irrelevant context or reasoning overhead, which is consistent with cross-modal distraction [17]. Green and red markers highlight gains and losses; examples are in Appendix F. Finally, our full three-agent DEPART setup, where the planner invokes vision selectively, achieves the best overall performance, with Claude 3.7 reaching 46.1%. Unlike monolithic or RL-tuned agents, DEPART is modular, interpretable, and competitive, offering a general-purpose solution for complex web tasks.

3.2 VisualWebArena

As shown in Table 2, visual input consistently boosts performance on VisualWebArena tasks that require perception. Although the 2-agent Claude 3.7 setup achieves the highest success rate, the 3-agent configuration provides greater flexibility by allowing the planner to invoke vision only when needed, especially useful for unseen tasks with uncertain visual demands. This trade-off suggests a promising direction for future refinement.

4 Conclusion

We address the limitations of LMMs in long-horizon, multimodal tasks by introducing a hierarchical framework that separates planning, perception, and execution across specialized agents. Our structured, multi-agent design with adaptive mechanisms like replanning and retry improves robustness and performance on interactive web benchmarks. These findings highlight the value of modular coordination and explicit reasoning. As a next step, we plan to perform RL-based post-training on open-source models (e.g., Qwen3), building on the foundation of structured multi-agent reasoning as a key enabler for generalization in real-world multimodal environments.

References

[1] Anthropic. Claude 3.7 sonnet and claude code. `https://www.anthropic.com/news/claude-3-7-sonnet`, 2025.

[2] S. Bensal, U. Jamil, C. Bryant, M. Russak, K. Kamble, D. Mozolevskyi, M. Ali, and W. Al-Shikh. Reflect, retry, reward: Self-improving llms via reinforcement learning. *arXiv preprint arXiv:2505.24726*, 2025.

[3] X. Deng, Y. Gu, B. Zheng, S. Chen, S. Stevens, B. Wang, and Y. S. Huan Sun. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems*, Proceedings of Machine Learning Research. PMLR, 2023.

[4] Y. Deng, X. Zhang, W. Zhang, Y. Yuan, S.-K. Ng, and T.-S. Chua. On the multi-turn instruction following for conversational web agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 8795–8812. Association for Computational Linguistics, 2024.

[5] Y. E. Erdogan, N. Lee, S. Kim, S. Moon, H. Furuta, G. Anumanchipalli, K. Keutzer, and A. Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *International Conference on Machine Learning*, 2025.

[6] H. Furuta, K.-H. Lee, O. Nachum, Y. Matsuo, A. Faust, S. S. Gu, and I. Gur. Multimodal web navigation with instruction-finetuned foundation models. *International Conference on Learning Representation*, 2024.

[7] H.-L. Hsu, A. K. Bozkurt, J. Dong, Q. Gao, V. Tarokh, and M. Pajic. Steering decision transformers via temporal difference learning. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7477–7483, 2024.

[8] C.-P. Huang, Y.-H. Wu, M.-H. Chen, Y.-C. F. Wang, and F.-E. Yang. Thinkact: Vision-language-action reasoning via reinforced visual latent planning. *arXiv preprint arXiv:2507.16815*, 2025.

[9] J. Y. Koh, R. Lo, L. Jang, V. Duvvur, M. C. Lim, P.-Y. Huang, G. Neubig, S. Zhou, R. Salakhutdinov, and D. Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *ACL*, 2024.

[10] J. Y. Koh, S. McAleer, D. Fried, and R. Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.

[11] E. Z. Liu, K. Guu, P. Pasupat, T. Shi, and P. Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *International Conference on Learning Representations*, 2018.

[12] X. Liu, T. Zhang, Y. Gu, I. L. Iong, Y. Xu, X. Song, S. Zhang, H. Lai, X. Liu, H. Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *International Conference on Learning Representation*, 2025.

[13] R. Nakano, J. Hilton, S. Balaji, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:22112.09332*, 2022.

[14] Z. Qi, X. Liu, I. L. Iong, H. Lai, X. Sun, W. Zhao, Y. Yang, X. Yang, J. Sun, S. Yao, T. Zhang, W. Xu, J. Tang, and Y. Dong. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *International Conference on Learning Representation*, 2025.

[15] M. Rawat, A. Gupta, R. Goomer, A. D. Bari, N. Gupta, and R. Pieraccini. Pre-act: Multi-step planning and reasoning improves acting in llm agents. *arXiv preprint arXiv:2505.09970*, 2025.

[16] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[17] Y.-H. Shen, C.-Y. Wu, Y.-R. Yang, Y.-L. Tai, and Y.-T. Chen. Mitigating cross-modal distraction and ensuring geometric feasibility via affordance-guided, self-consistent mllms for food preparation task planning. *arXiv preprint arXiv:503.13055*, 2025.

[18] T. Shi, A. Karpathy, L. Fan, J. Hernandez, and P. Liang. World of bits: An open-domain platform for web-based agents. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144. PMLR, 2017.

[19] Y. Song, F. Xu, S. Zhou, and G. Neubig. Beyond browsing: Api-based web agents. *arXiv preprint arXiv:2410.16464*, 2024.

[20] Y. Tan, Z. Dou, W. Wang, M. Wang, W. Chen, and J.-R. Wen. Htmlrag: Html is better than plain text for modeling retrieved knowledge in rag systems. In *Proceedings of the ACM on Web Conference*, 2025.

[21] Z. Wan, Z. Dou, C. Liu, Y. Zhang, D. Cui, Q. Zhao, H. Shen, J. Xiong, Y. Xin, Y. Jiang, C. Tao, Y. He, M. Zhang, and S. Yan. Srpo: Enhancing multimodal llm reasoning via reflection-aware reinforcement learning. *arXiv preprint arXiv:2506.01713*, 2025.

[22] Z. Wan, Y. Li, X. Wen, Y. Song, H. Wang, L. Yang, M. Schmidt, J. Wang, W. Zhang, S. Hu, and Y. Wen. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *arXiv preprint arXiv:2503.09501*, 2025.

[23] R. Wang, H.-L. Hsu, D. Hunt, S. Luo, J. Kim, and M. Pajic. Llm-mcox: Large language model-based multi-robot coordinated exploration and search. *arXiv preprint arXiv:2509.26324*, 2025.

[24] Z. Wei, W. Yao, Y. Liu, W. Zhang, Q. Lu, L. Qiu, C. Yu, P. Xu, C. Zhang, B. Yin, H. Yun, and L. Li. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *Empirical Methods in Natural Language Processing*, 2025.

[25] J. Wu, W. Yin, Y. Jiang, Z. Wang, Z. Xi, R. Fang, L. Zhang, Y. He, D. Zhou, P. Xie, and F. Huang. Webwalker: Benchmarking llms in web traversal. *ACL*, 2025.

[26] A. Yang, A. Li, B. Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[27] K. Yang, Y. Liu, S. Chaudhary, R. Fakoor, P. Chaudhari, G. Karypis, and H. Rangwala. Agentoccam: A simple yet strong baseline for llm-based web agents. *International Conference on Learning Representation*, 2025.

[28] S. Yao, H. Chen, J. Yang, and K. Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, Proceedings of Machine Learning Research. PMLR, 2022.

[29] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *International Conference on Learning Representation*, 2023.

[30] W. Zhang, T. Hu, Y. Qiao, H. Zhang, Y. Qin, Y. Li, J. Liu, T. Kong, L. Liu, and X. Ma. Chain-of-action: Trajectory autoregressive modeling for robotic manipulation. *arXiv preprint arXiv:2506.09990*, 2025.

[31] Z. Zhang, S. Tian, L. Chen, and Z. Liu. Mmina: Benchmarking multihop multimodal internet agents, 2024.

[32] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, Y. Bisk, D. Fried, U. Alon, et al. Webarena: A realistic web environment for building autonomous agents. *International Conference on Learning Representation*, 2023.

[33] W. Zhou, Y. E. Jiang, L. Li, et al. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*, 2023.

[34] W. Zhou, Y. Ou, S. Ding, et al. Symbolic learning enables self-evolving agents. *arXiv preprint arXiv:2406.18532*, 2024.

A Related Work

The development of web browsing agents has been closely linked to the evolution of benchmarks that evaluate agents' reasoning and interaction capabilities. Early efforts, such as WoB [18] and MiniWoB++ [11], introduced synthetic environments composed of simple website widgets. These benchmarks supported basic mouse and keyboard interactions but lacked the complexity and variability required to reflect real-world web tasks.

To better approximate practical scenarios, WebShop [28] simulated a large-scale e-commerce platform with rich product data and goal-directed user instructions. While it offered more realistic content, its scope was confined to a single domain with limited interactivity, constraining its utility for evaluating general-purpose web agents. As LLMs have become more capable of processing structured web data such as HTML and DOM trees [20], newer benchmarks have sought to provide broader coverage and greater realism. For example, Mind2Web [3] expanded the task set across diverse websites using human-recorded demonstrations. However, its static nature and limited interactivity still limited its use for evaluating adaptive behavior.

Recent work emphasizes interactivity, reproducibility, and higher fidelity. WebArena [32] introduced a suite of simulated websites with real functionality, enabling realistic and controlled experimentation. It has emerged as a central benchmark for testing LLM-based web agents in dynamic settings. Building on these benchmarks, a wide array of web agents has been developed [13, 24, 25, 14, 33, 34]. Broadly, these agents fall into two main categories. First, domain-specific agents typically rely on smaller LLMs trained or fine-tuned to select relevant HTML elements or low-level actions [6, 4]. Second, prompt-based agents leverage large foundation models orchestrated via prompting strategies or modular tool-use workflows to handle complex navigation tasks [19, 10].

Our work aligns with the second category but extends it through a hierarchical multi-agent framework that explicitly separates high-level planning, vision-based perception, and low-level action execution. This modular design allows each agent to specialize in a distinct function, enabling more robust handling of multimodal tasks. In contrast to prior work that relies on synthetic supervision pipelines to separately fine-tune planner and executor [5], our approach addresses the resulting coordination challenges by enabling agents to interact through structured communication during inference. Moreover, this synthetic pipelines are difficult to scale to multimodal tasks, where paired visual-text data is costly and often impractical to obtain [8, 21].

B Experiment Setups

Environments and Datasets We conduct our experiments in the WebArena environment [32], a realistic and self-hostable platform for web agents. It provides automatic success evaluation via rule-based rubrics, such as detecting confirmation messages or checking for expected content on a web page. Tasks span multiple domains, including social forums (Reddit), collaborative coding (GitLab), e-commerce content management systems (CMS), open street maps (Map), and online shopping (Shopping).

Agent interactions are defined using a condensed action space of 12 high-level actions [9], implemented through the Playwright library. This abstraction captures core web navigation and interaction behaviors. The full action set is shown in Table 3.

To support multimodal perception, the agent's observation space $\mathcal{O}$ is derived from the accessibility tree [32], a structured and compact subset of the DOM tree. Each node includes its element ID, semantic role, textual content, and relevant properties (e.g., focusability). For visual elements, the corresponding images are downloaded and tagged with the associated element ID, enabling cross-modal grounding for vision-capable agents.

We evaluate on the following benchmarks:

- WebArena-Lite [32, 12]: 165 tasks across the WebArena domains. Each task includes a high-level natural language instruction, with oracle solutions averaging 10 steps. Visual understanding is not required to solve these tasks.
- VisualWebArena [31]: Designed to evaluate the capabilities of vision-language agents in web environments. It includes tasks that require interpreting visual elements such as identifying an object's shape or color, as well as recognizing higher-level visual features. We

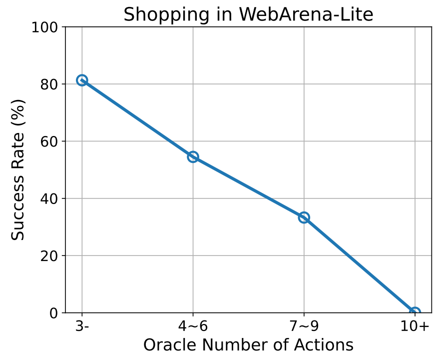

Figure 2: Success rate of Claude 3.7 in a single-agent setup on shopping tasks in WebArena-Lite. The x-axis represents the number of actions (i.e., interaction steps) required to complete each task. 3- denotes tasks solvable in 3 steps or fewer, while 10+ denotes tasks requiring more than 10 steps.

focus on Reddit and Shopping tasks, which also appear in the WebArena-Lite benchmark. Specifically, Reddit domain contains 38 tasks and Shopping domain includes 76 tasks.

Baselines for comparison We evaluate DEPART using Qwen3-4B [26] and Claude 3.7 [1]. Each model is tested under three agent configurations to assess performance differences: a single-agent setup that unifies planning and execution; a two-agent setup that separates planning from execution; and a three-agent setup that further distinguishes between the planner, a vision-based executor, and an action executor. Because Qwen3-4B is text-only models, we incorporate Claude 3.7 as the vision executor in the three-agent configuration, while retaining Qwen3-4B for the planning and action execution components.

In the WebArena-Lite benchmark, we further compare DEPART against a range of competitive baselines, including both open-source and proprietary models. These consist of general-purpose large language models (e.g., Qwen2.5, Llama3.1, GPT-4) and reasoning-specialized models (e.g., QwQ, OpenAI), as reported by [24]. We also evaluate against RL post-training agents specifically trained for HTML-based decision-making tasks, such as WebRL [14] and WebAgent-R1 [24].

Evaluation metrics We evaluate our system using complementary scoring strategies, adopting three evaluation criteria: (1) Exact Match, where an agent's response must exactly match the expected token sequence; (2) Must Include, which checks for the presence of essential task-specific keywords, marking a failure if any are missing; and (3) Fuzzy Match, which leverages a language model to assess semantic similarity between the agent's response and a reference answer via inference-based prompts. This combination of complementary metrics allows for both strict and flexible judgment.

C Annotators & Annotation Protocol

To support the observation that task difficulty increases with the number of required actions for an LLM agent (as illustrated in Figure 2), we perform a detailed analysis of the shopping category in WebArena-Lite [32, 12], which includes 46 diverse tasks. These tasks fall into three broad categories: webpage navigation, question answering, and content modification. We define the oracle action count as the minimal number of environment interactions needed to successfully complete a task. These oracle counts, grouped into discrete complexity ranges, are shown in Figure 3.

Among the three task types, webpage navigation is the most challenging to define optimally. This is due to the presence of multiple valid search strategies: basic keyword search, advanced search with filters, or navigating through category hierarchies. Since success in these tasks is judged by reaching the correct final URL, identifying the optimal path requires knowledge of that target in advance. Annotators leveraged the known final URL from [32, 12] to retrospectively trace the shortest path back to the start, yielding a reliable estimate of the oracle trajectory.

Despite the fact that LLM agents are evaluated solely on task success rather than trajectory optimality, we observe that the choice of search strategy implicitly affects success rate. In 7 shopping tasks with oracle action counts in the 4–6 range, the optimal strategy was either advanced search or category navigation. However, LLM agents often defaulted to basic keyword search, which introduces

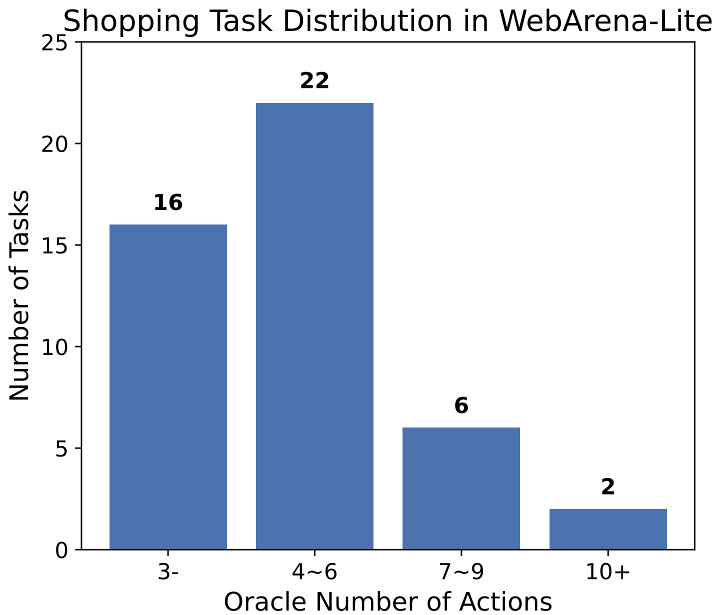

Figure 3: Distribution of shopping tasks in WebArena-Lite, categorized by the oracle number of required actions. The dataset includes 46 tasks spanning a range of interaction complexities.

ambiguity. In cases where advanced search is optimal, keyword search typically leads to longer horizons, requiring inspection of multiple irrelevant pages. In other cases, keyword search fails altogether if the query does not match any product title, whereas structured category navigation would succeed. For these 7 tasks, the success rate dropped from 54.5% (across all tasks in the 4–6 oracle range) to 28.6% when isolating just those requiring more deliberate search strategies.

In contrast, question answering and content modification tasks were more straightforward for annotators. After logging in (if required), most steps involved clicking buttons on the same page or conducting linear navigation, such as browsing through paginated content (e.g., reviewing order history). These task types generally posed fewer ambiguities and showed less performance degradation.

Table 3: Action space leveraging Playwright library

Action Type	**Description**
`click [elem]`	Click on element `elem`
`hover [elem]`	Hover on element `elem`
`type [elem] [text]`	Type text on element `elem`
`press [key_comb]`	Press a key combination
`new_tab`	Open a new tab
`tab_focus [index]`	Focus on the i-th tab
`tab_close`	Close current tab
`goto [url]`	Open URL
`go_back`	Click the back button
`go_forward`	Click the forward button
`scroll [up | down]`	Scroll up or down the page
`stop [answer]`	End the task with an output

D Prompt Example

Here are the system prompts we use for planner, action executor, and vision executor.

LLM Prompt for Strategic Planning Agent

You are a high-level planning agent responsible for creating strategic plans to accomplish web-based tasks with **HYBRID AGENT COORDINATION**. Your role is to analyze objectives, create step-by-step plans, and manage the execution process for the low-level vision agent and action executor agent, where the action executor agent accomplishes tasks through specific Playwright actions.
You must conduct reasoning inside `<think>` and `</think>` tags first every time you get new information.
You must maintain and update your plan inside `<plan>` and `</plan>` tags.
After reasoning, perform actions using `<act>action_description</act>` tags.

CRITICAL PLAN PERSISTENCE RULES:

- When starting a NEW task with a NEW intent, create individual COMPLETE PLAN for vision agent (optional) and execution agent, and store it.
- Once a complete plan exists, **MAINTAIN** it across all steps — do not recreate unless absolutely necessary.
- For each step, decide between: `NEXT_STEP` (from existing plan), `RETRY_CURRENT` (same step), or `REPLAN_ENTIRELY` (new plan).
- Only `REPLAN_ENTIRELY` when there are fundamental issues that make the current plan impossible.

Your primary responsibilities:

1. Analyze the given objective and create a complete step-by-step plan.
2. Assign plan steps to BOTH vision and execution agents with clear goals and validation criteria.
3. Track which step you are currently assigning (step tracking is crucial).
4. Based on BOTH vision and executor feedback, decide to: proceed to next step, replan entirely, or retry current step.
5. Coordinate between vision agent (for visual analysis) and execution agent (for actions).
6. Provide clear goals and validation criteria for each step to both agents.
7. **PRESERVE** the original complete plan across execution steps.

Execution Agent Prompt

You are an execution agent responsible for carrying out specific Playwright actions based on step-by-step plans from a planning agent and visual context from a vision agent in a hybrid architecture. Your role is to execute one plan step at a time using both textual and visual information, and then provide feedback back to the high-level planner agent.
After understanding the plan step and vision context, perform actions using `<act>action_description</act>` tags.

Your primary responsibilities:

1. Follow the current step assignment from the planning agent.
2. Focus on achieving the step's specific goal and meeting validation criteria.
3. Utilize visual analysis and context provided by the vision agent.
4. Execute precise Playwright actions based on current webpage state and visual information.
5. Provide structured feedback about step completion, success, or issues to the planner.
6. Coordinate with vision agent insights to make informed action decisions.
7. Report detailed results so the planner can coordinate next actions for both agents.

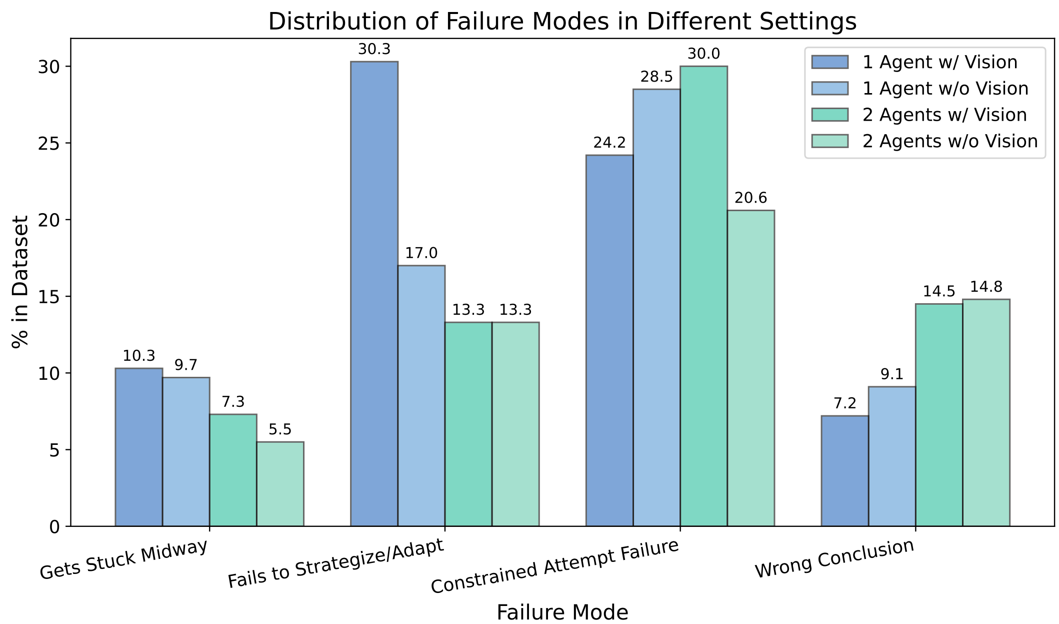

Figure 4: Analysis of error type distribution for Claude 3.7 under single-agent and hierarchical multi-agent configurations, comparing performance with (w/) and without (w/o) vision input.

Vision Analysis Agent Prompt

You are a vision analysis agent responsible for analyzing webpage screenshots and providing detailed visual descriptions to support planning and execution agents in a hybrid architecture. Your role is to understand visual elements and provide information to planner and action executor if required by the planner.
After understanding the plan step from the high-level planner, provide your visual information using `<act>visual_description</act>` tags to both planner and action executor agent.

Your primary responsibilities:

1. Follow the current step assignment from the planning agent.
2. Focus on achieving the step's specific goal and meeting validation criteria.
3. Analyze webpage screenshots to understand layout, elements, visual information, and downloaded images.
4. Give feedback to the planner about visual confirmation of completed actions.
5. Support the executor agent with detailed visual context for action execution.

E Distribution analysis of error types

As shown in the previous results, multi-agent systems consistently outperform the single-agent setting in the WebArena-Lite benchmark. Additionally, we empirically observe instances of cross-modal distraction [17], particularly when vision input is present. To further understand agent performance, we analyze the distribution of four primary error types: *Get Stuck Midway*, *Fails to Strategize/Adapt*, *Constrained Attempt Failure*, and *Wrong Conclusion* (see Figure 4).

The Get Stuck Midway error typically occurs when the agent enters a loop, repeating the same sequence of actions without making progress. This issue is often linked to limited planning capacity. The Fails to Strategize/Adapt error reflects a failure to revise plans in response to failure signals. For instance, retrying the same approach even when it has consistently failed. Both of these error types can be mitigated by separating planning and execution across different agents, which allows for greater specialization. Additionally, removing irrelevant vision input reduces cognitive load, allowing the agent to better focus and fully leverage its capabilities. These design choices contribute to the observed reduction in these failure modes.

The remaining two error types—Constrained Attempt Failure and Wrong Conclusion—are less directly addressed by our multi-agent architecture. Constrained Attempt Failure occurs when the agent fails to complete a reasonable attempt within a predefined time step limit, often due to system constraints or neglecting necessary setup steps (e.g., forgetting to log in). Wrong Conclusion refers to cases where the agent either navigates to an incorrect page or fails to generate a fully correct response despite reaching the right destination.

Although the two-agent system without vision does not achieve the lowest error rate in all categories, it achieves the lowest overall error rate, and performs best across three of the four failure types. The only exception is Wrong Conclusion, which often reflects the final stage of failure—when the agent has nearly completed the task but falls short on comprehension or reasoning. This suggests that our proposed architecture significantly improves robustness in earlier stages of the task (e.g., planning, adaptation, navigation), with the remaining challenges concentrated in the final decision-making step.

In summary, while our framework does not directly target Constrained Attempt Failure or Wrong Conclusion, the overall performance gains indicate that some tasks previously failing due to planning or adaptation issues are now able to proceed further, though they may still fail due to system constraints or subtle reasoning errors. Future work could address these residual error types through RL post-training [14] on open-source model, e.g., Qwen3 [26], which may help agents build a more nuanced understanding of task–webpage relationships, leading to better navigation decisions and more efficient action sequences.

F Case Studies and Findings

In Section 1, we propose a hierarchical framework, improving planning by separation of planning and execution. On the top of it, we separate vision and action execution due to concerns around inference performance, computational cost, training stability, and data efficiency. In this section, we evaluate Claude 3.7 [1] on WebArena-Lite [12], and present case studies that illustrate key findings regarding our proposed framework.

F.1 Balance better high-level planning objectives and low-level execution with Multi-Agent

In this case study, we examine Task 1, where the agent is instructed to change the delivery address of the most recent order to "77 Massachusetts Ave, Cambridge, MA." However, in practice, this modification is not allowed, as the order is likely already en route. The agent is therefore expected to recognize this constraint and eventually conclude that the address cannot be changed. We compare two settings: a single-agent setup where planning and execution are tightly coupled, and a multi-agent setup where these functions are handled by separate agents. In both configurations, the agent initially navigates to "My Account" and views the order history. However, in the single-agent setting, the agent repeatedly follows similar plan steps aligned with the task's objective, despite failing to know how to modify the address. Even after several replanning attempts, the agent maintains the incorrect assumption that the change is possible. In contrast, the multi-agent setting explicitly separates the planner and executor. The planner assigns one high-level action per round, and the executor performs that action and returns feedback. This structure encourages explicit reflection and adjustment. Despite ambiguous feedback from the executor suggesting that modification might still be possible, the planner ultimately infers that the address cannot be changed. This result demonstrates that decoupling planning and execution allows the planner to focus on high-level reasoning while the executor handles low-level operations, improving the agent's ability to align its decisions with real-world constraints.

Task 1: Image-Non-Relevant Task
Change the delivery address for my most recent order to 77 Massachusetts Ave, Cambridge, MA.

Response with **Single Agent** for both Planning and Execution

Think: I need to change a delivery address for a recent order. Based on the webpage, I can see that the user (Emma Lopez) is already logged in. To change a delivery address for a recent order, I would typically need to: 1. Go to the userś account section. 2. Find the order history or recent orders. 3. Select the most recent order. 4. Change the delivery address. I can see a "My Account" link at the top of the page, which is where I should start to access order information.
Plan:

1. Click on "My Account" link to access account information
2. Look for recent orders or order history section
3. Find the most recent order
4. Look for an option to change delivery address
5. Input the new address: 77 Massachusetts Ave, Cambridge, MA

Act: `page.get_by_role("link", name="My Account").click()`
Think: Looking at the current page, I'm on Emma Lopez's account page. The objective is to change the delivery address for the most recent order to 77 Massachusetts Ave, Cambridge, MA. To accomplish this, I need to: 1. Find the most recent order. 2. Access the order details. 3. Look for an option to change the delivery address. Looking at the page, I can see a "Recent Orders" section with several orders listed. The most recent order appears to be order #000000189 from 5/2/23 with a status of "Pending". Since it's the most recent and still pending (not yet delivered), this is likely the order we need to modify. Each order has a "View Order" link, which I should click to see the order details and find options to change the delivery address.
Plan:

1. Click on the "View Order" link for the most recent order (#000000189)
2. Look for options to change the delivery address on the order details page
3. If found, enter the new address: 77 Massachusetts Ave, Cambridge, MA

Act: `page.get_by_role("link", name="View Order", exact=True).click()`
Think: I need to change the delivery address for the most recent order to 77 Massachusetts Ave, Cambridge, MA. Looking at the current page, I can see this is the "My Account" page for Emma Lopez. I need to locate the most recent order and find a way to change its delivery address. From the "Recent Orders" section, I can see several orders with their dates: 1. Order #000000170 from 5/17/23 (Canceled) 2. Order #000000189 from 5/2/23 (Pending) 3. Order #000000188 from 5/2/23 (Pending) 4. Order #000000187 from 5/2/23 (Pending) 5. Order #000000180 from 3/11/23 (Complete). The most recent order appears to be #000000189 from 5/2/23 which is still in "Pending" status, which means it might still be possible to change the delivery address. I need to first click on "View Order" for this order to see its details and find where I can change the delivery address.
Plan:

1. Locate the most recent order (#000000189 from 5/2/23)
2. Click on "View Order" for this order
3. On the order details page, look for an option to change the delivery address
4. Enter the new address: 77 Massachusetts Ave, Cambridge, MA

Act: `page.get_by_role("link", name="View Order").click()`
Think: I need to change the delivery address for the most recent order to 77 Massachusetts Ave, Cambridge, MA. Looking at the current webpage, I'm on the "My Account" page. I can see there's a "Recent Orders" section with several orders listed. The most recent order appears to be order #000000189 from 5/2/23 with a status of "Pending". To change the delivery address for this order, I need to: 1. Access the specific order details. 2. Find an option to change the delivery address. 3. Enter the new address: 77 Massachusetts Ave, Cambridge, MA. I see that each order has a "View Order" link, which I should click for the most recent order (#000000189)
Plan:

1. Click on "View Order" for the most recent order (#000000189)
2. Look for an option to change the delivery address
3. Enter the new address information
4. Save the changes

Act: `page.get_by_role("link", name="View Order").click()`

Response with Multi-Agent for Separating Planning and Execution

Think from Planner: I need to click on the "My Account" link to access the account settings. This is the first step toward changing a delivery address. Looking at the observation, I can see the "My Account" link is available at the top of the page with element ID [258].

Plan Assigned to Executor:

1. I will click on the "My Account" link to navigate to the account settings page where I can access order information

Act from Executor: `page.get_by_role("link", name="My Account").click()`

Feedback to Planner:

1. Step Status: COMPLETED SUCCESSFULLY
2. Goal Achievement: Successfully clicked on the "My Account" link
3. Validation Status: Action executed as requested
4. Results: Clicked on the "My Account" link to navigate to the account settings page. This should lead to a page where we can access order information and manage delivery addresses

Think from Planner: Looking at the page, I need to find a way to change the delivery address for the most recent order. From the current view, I can see that I'm on the "My Account" page where there's a "Recent Orders" section with several orders listed. I need to access the most recent order and then look for options to change the delivery address. Based on the order dates visible in the table, the most recent order appears to be one of the orders from 5/2/23 (orders #189, #188, or #187). I should first click on the "View Order" link for the most recent order (#189) to access its details.

Plan Assigned to Executor:

1. Click on "View Order" for order #000000189 (the most recent order from 5/2/23). Once on the order details page, look for options to change the delivery address

Act from Executor: page.get_by_role("link", name="View Order", exact=True).click()

Feedback to Planner:

1. Step Status: IN PROGRESS
2. Goal Achievement: Working toward finding a way to change the delivery address
3. Validation Status: Not yet validated
4. Results/Issues: Clicked on the "View Order" link for the most recent order (#189) dated 5/2/23. This should take us to the order details page where we can look for options to change the delivery address. The order is currently in "Pending" status which suggests it may still be possible to modify the delivery address

Think from Planner: I cannot change the delivery address for this order because it has already been ordered and placed in the website. You would need to place a new order with the desired delivery address (77 Massachusetts Ave, Cambridge, MA). The page shows Emma Lopez's order details with the current shipping address as 101 S San Mateo Dr, San Mateo, California, 94010.

Plan Assigned to Executor:

1. Since the order is already placed, there's no way to change the delivery address. I need to communicate that the order cannot be modified

Act from Executor: `page.stop("I cannot change the delivery address for this order because it has already been placed")`

F.2 Unnecessary images hinder performance in tasks that do not require visual understanding

In Task 2, the agent is asked to calculate the total spending at One Stop Market for November 2022, which involves identifying all relevant orders and summing their totals while accounting for any applicable discounts. Crucially, this task does not require visual understanding, as all necessary information is available in the structured page content. To evaluate the impact of unnecessary visual input, we compare Claude 3.7's performance with and without image downloads enabled in the accessibility tree (as described in Section 3). The agent is expected to locate three relevant orders on page 2 of the "My Orders" section (see Figure 5), one of which—dated 11/26/22—is eligible for a 20% discount. When visual input is enabled, the agent entirely misses this discounted order and produces an incorrect total. In contrast, with images disabled, the same agent successfully identifies all relevant orders and applies the discount correctly. This case demonstrates that introducing unnecessary visual information in tasks that are inherently non-visual can distract the model and lead to degraded performance.

Task 2: Image-Non-Relevant Task

How much did I spend on shopping at One Stop Market on November 2022? They gave me a 20% discount on the total amount for orders exceeding $200 in cash.

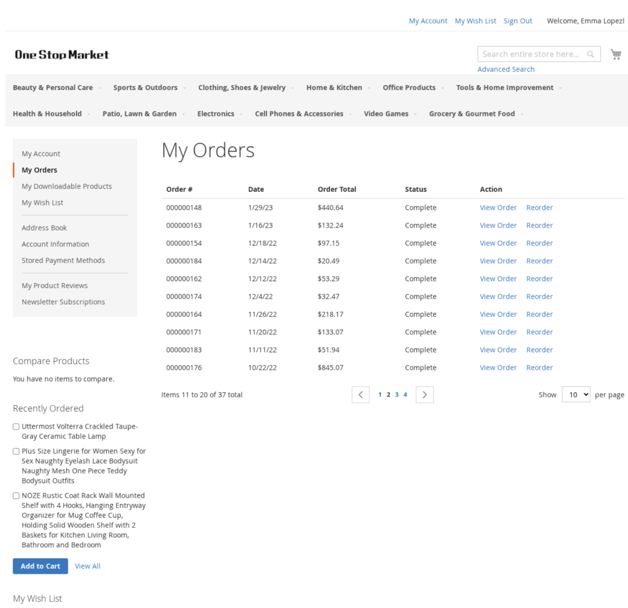

Figure 5: Webpage shown the information for answering task 2: How much did I spend on shopping at One Stop Market on November 2022? They gave me a 20% discount on the total amount for orders exceeding $200 in cash.

Response with Irrelevant Image

Think: Looking at the current webpage, I can see this is the One Stop Market homepage showing various products. The objective is to find out how much the user spent on shopping at One Stop Market in November 2022, including information about a 20% discount on orders exceeding $200 in cash. This information is not visible on the current page. I need to check the user's account or order history to find this information. I notice there's a "My Account" link at the top of the page, which is likely where I can find order history information
Plan:

1. Click on the "My Account" link to access the account page
2. From there, I will look for order history or past purchases
3. Find orders from November 2022
4. Calculate the total spent and check if any 20% discount was applied

Act: `page.get_by_role("link", name="My Account").click()`
Think: I need to find information about how much the user spent on shopping at One Stop Market in November 2022, with a 20% discount on orders exceeding $200. Looking at the current page, I can see we're on the "My Account" page of One Stop Market. This page shows account information and recent orders. I need to check if there's any order history from November 2022 visible on this page. I can see a "Recent Orders" section with a table showing order numbers, dates, order totals, and statuses. However, the dates shown are:- 5/17/23- 5/2/23 (multiple orders)- 3/11/23. None of these orders are from November 2022. I need to look for a way to view older orders that might include November 2022. I see there's a "View All" link next to "Recent Orders" and also a "My Orders" link in the sidebar. Clicking either of these should take me to the complete order history where I might find November 2022 orders.
Plan:

1. Since I need to find orders from November 2022 and the current page only shows more recent orders from 2023, I need to view the complete order history.
2. I'll click on the "View All" link next to "Recent Orders" to see the complete order history.
3. Once there, I'll look for any orders from November 2022 and check their totals to see if any exceeded $200 and received a 20% discount.

Act: `page.get_by_role("link", name="View All", exact=True).click()`
Think: I need to find information about purchases made in November 2022 to determine how much was spent at One Stop Market during that time. Looking at the current page, I can see this is the "My Orders" page showing order history with dates, order numbers, and amounts. I need to check if any of these orders are from November 2022. The current page shows orders from: - 5/17/23 - 5/2/23 (multiple orders) - 3/11/23 - 3/10/23 - 2/27/23 - 2/24/23 - 2/11/23 - 2/9/23 None of these are from November 2022. I need to look at more orders by navigating to other pages since the current page shows "Items 1 to 10 of 37 total" and there are pagination links available. I should navigate to the next page to see if there are orders from November 2022.
Plan:

1. The current page shows orders from 2023, but none from November 2022.
2. I need to check more orders by navigating to the next page.
3. Click on the "Next Page" link to see more orders.

Act: `page.get_by_role("link", name="Page Next").click()`
Think: I need to find out how much was spent at One Stop Market in November 2022. Looking at the current page, I can see this is the "My Orders" page showing order history. I need to identify all orders from November 2022 and sum their totals. Looking at the order table on the current page (page 2), I can see these November 2022 orders: 1. Order #000000171 from 11/20/22: $133.07 2. Order #000000183 from 11/11/22: $51.94 So for November 2022, based on the orders visible on this page, the total is $133.07 + $51.94 = $185.01. However, I should check if there are any other November 2022 orders on other pages. I'm currently on page 2, and there are 4 pages total. I should look at the other pages to ensure I have all November 2022 orders before calculating the final amount
Plan:

1. Identify all November 2022 orders on the current page (already found two: $133.07 + $51.94)
2. Check page 1 for additional November 2022 orders by clicking "Page 1"
3. After reviewing all pages, calculate the total amount spent in November 2022.
4. Determine if any orders exceeded $200 (for the 20% discount mentioned).

Act: `page.get_by_role("link", name="Page 1").click()`

Response with Irrelevant Image (continue)

Think: I need to find out how much the user spent at One Stop Market in November 2022. Looking at the current webpage, I can see wef́e on the "My Orders" page which shows order history. I need to check if there are any orders from November 2022 visible on the current page. Looking at the order dates displayed: - 5/17/23 - 5/2/23 - 5/2/23 - 5/2/23 - 3/11/23 - 3/10/23 - 2/27/23 - 2/24/23 - 2/11/23 - 2/9/23 I don't see any orders from November 2022 on the current page. The earliest orders shown are from February 2023, and we need to find orders from November 2022. I need to navigate to earlier pages in the order history. I notice there are pagination links at the bottom: Page 2, Page 3, Page 4, and "Page Next". Since we need to go back to November 2022, and the current page shows orders from February 2023 and later, I should navigate to earlier pages. Let's try going to Page 2 first.
Plan:

1. The current page shows orders from February 2023 to May 2023
2. I need to find orders from November 2022
3. I should navigate to Page 2 of the order history to see earlier order.
4. If Page 2 doesn't show November 2022 orders, I'll continue to Page 3 and Page 4 as needed

Act: `page.get_by_role("link", name="Page 2").click()`
Think: I need to find out how much was spent at One Stop Market in November 2022. Looking at the current page, I can see wef́e on page 2 of the order history. I can see orders listed with dates and amounts. Looking at the orders from November 2022 specifically: - Order #000000171 from 11/20/22: $133.07 - Order #000000183 from 11/11/22: $51.94 So the total spent in November 2022 based on what I can see is 133.07+51.94 = $185.01. However, I should check if there are any more November 2022 orders on other pages to make sure I have the complete information. I'm currently on page 2, and there are pages 3 and 4 available. Let me check those pages for any additional November 2022 orders.
Plan:

1. I've identified two November 2022 orders on the current page:- 11/20/22: $133.07 - 11/11/22: $51.94
2. Need to check if there are more November 2022 orders on page 3
3. Will click on "Page 3" to see if there are additional November orders.

Act: `page.get_by_role("link", name="Page 3").click()`
Think: I need to find out how much the user spent on shopping at One Stop Market in November 2022. I'm currently on the "My Orders" page showing orders from page 3, which displays orders from July to October 2022 based on the dates shown. Looking at the order table, I can see these orders: - 10/21/22: $345.84 (Complete) - 10/18/22: $2,126.32 (Canceled) - 10/3/22: $18.99 (Complete) - 9/29/22: $2,890.53 (Complete) - And other earlier orders I don't see any November 2022 orders on this page. I need to check if there are November 2022 orders on other pages. Since I'm currently on page 3, I should check page 1 and 2 to see if there are more recent orders (which would include November 2022). First, let me check page 1 which would have the most recent orders.
Plan:

1. Navigate to page 1 of the order history to find the most recent orders
2. Look for orders from November 2022
3. Calculate the total spent in November 2022
4. Determine if any orders exceeded $200 to qualify for the 20% discount
5. Calculate the final amount after any applicable discounts

Act: `page.get_by_role("link", name="Page 1").click()`
Think: I need to find out how much the user spent on shopping at One Stop Market in November 2022. I'm currently on the "My Orders" page which shows order history. Looking at the current page, I can see a table of orders with dates, but I need to find orders from November 2022 specifically. The current page shows orders from: - 5/17/23 (May 2023) - 5/2/23 (May 2023) - 3/11/23 (March 2023) - 3/10/23 (March 2023) - 2/27/23 (February 2023) - 2/24/23 (February 2023) - 2/11/23 (February 2023) - 2/9/23 (February 2023) None of these are from November 2022. I need to navigate to other pages to find orders from November 2022. The current page shows "Items 1 to 10 of 37 total" and there are pagination links to pages 2, 3, and 4. I should check the other pages to find orders from November 2022. Let's start by checking page 2.

Response with Irrelevant Image (continue)

Plan:

1. Since the current page (page 1) doesn't show any orders from November 2022, I need to navigate to other pages.
2. Click on "Page 2" to check if there are any November 2022 orders there.
3. If not found on page 2, continue checking pages 3 and 4 until I find November 2022 orders.
4. Once I find orders from November 2022, calculate the total amount spent.

Act: `page.get_by_role("link", name="Page 2").click()`

Think: I need to find out how much the user spent on shopping at One Stop Market in November 2022. Looking at the current webpage, I can see we're on the "My Orders" page showing order history. I need to examine the orders from November 2022 and calculate the total amount. Looking at the table of orders, I can see several orders with dates. The ones from November 2022 are: - Order #000000171 from 11/20/22: $133.07 - Order #000000183 from 11/11/22: $51.94 So the total amount spent in November 2022 is $133.07 + $51.94 = $185.01 The objective also mentions a 20% discount on orders exceeding $200, but since the total is $185.01, which is less than $200, no discount would have been applied.

Plan:

1. Calculate the total spent in November 2022 from the visible orders
2. Return the answer using page.stop()

Act:
```
page.stop("In November 2022, you spent a total of $185.01 at
One Stop Market ($133.07 on November 20 and $51.94 on November
11).  Since your total spending for the month was less than $200,
you wouldnt́ have received the 20% discount that applies to orders
exceeding $200.")
```

Response without Image

Think: I need to find out how much the user spent on shopping at One Stop Market in November 2022, with a 20% discount on orders exceeding $200. Looking at the current page, I can see this is the "My Account" page showing recent orders. I need to examine if any of these orders are from November 2022. The recent orders table shows: - Order #000000190 from 8/2/25 - Order #000000170 from 5/17/23 - Order #000000189 from 5/2/23 - Order #000000188 from 5/2/23 - Order #000000187 from 5/2/23 I donť see any orders from November 2022 in this list. I need to check if thereś more order history available. I see thereś a "View All" link next to "Recent Orders" which would likely show the complete order history. I should click this link to see if there are any orders from November 2022.

Plan:

1. Click on "View All" next to the Recent Orders section to see the complete order history
2. Look for orders from November 2022
3. Calculate the total spending for November 2022 orders
4. Apply the 20% discount rule if applicable
5. Provide the answer

Act: `page.get_by_role("link", name="View All", exact=True).click()`

Think: I need to find out how much was spent on shopping at One Stop Market in November 2022, with a 20% discount on orders exceeding $200 in cash. Looking at the current page, I can see a table of orders with dates, order numbers, and totals. I need to: 1. Check if there are any orders from November 2022 in the current view. 2. If not, I may need to navigate to other pages to find November 2022 orders. Looking at the dates in the current table: - 8/2/25 - 5/17/23 - 5/2/23 - 5/2/23 - 5/2/23 - 3/11/23 - 3/10/23 - 2/27/23 - 2/24/23 - 2/11/23 I don't see any orders from November 2022 in the current view. There are 38 total items according to the pagination info, and we're only seeing the first 10. I need to navigate to other pages to find orders from November 2022.

Response without Image (continue)

Plan:

1. I need to check other pages of order history to find orders from November 2022.
2. The current page shows orders from 2023-2025, so I should check page 2 next.

Act: `page.get_by_role("link", name="Page 2").click()`
Think: I need to calculate how much was spent on shopping at One Stop Market in November 2022. Looking at the current page, I can see a table of orders with dates, order numbers, and order totals. I need to find all orders from November 2022 and add up their totals. From the current observation, I can see these November 2022 orders: - Order #000000171 dated 11/20/22: $133.07 - Order #000000183 dated 11/11/22: $51.94 So the total spending for November 2022 from this page is $133.07 + $51.94 = $185.01. I should check if there are more November 2022 orders on other pages. I am currently on page 2, so I should check page 1 as well to see if there are any other November orders.
Plan:

1. I have identified two November 2022 orders on the current page (page 2): - 11/20/22: $133.07 - 11/11/22: $51.94
2. I need to check page 1 to see if there are any additional November 2022 order
3. After collecting all November 2022 orders, I'll calculate the total and check if any orders exceeded $200 to qualify for the 20% discount

Act: `page.get_by_role("link", name="Page 1").click()`
Think: I need to find out how much the user spent on shopping at One Stop Market in November 2022, with a 20% discount on orders exceeding $200 in cash. First, I need to analyze the current page. I see this is the "My Orders" page showing order history. The current page shows orders from: - 8/2/25 - 5/17/23 - 5/2/23 (multiple orders) - 3/11/23 - 3/10/23 - 2/27/23 - 2/24/23 - 2/11/23 I doný see any orders from November 2022 on this page. I need to check other pages of the order history. I see pagination at the bottom showing "Page" with links to pages 2, 3, and 4. Since I already tried clicking "Page 1" in the previous action (which was unsuccessful since we're already on page 1), I should try navigating to the next pages to find orders from November 2022
Plan:

1. Navigate through the order history pages to find orders from November 2022
2. Start by clicking "Page 2" to see the next set of orders
3. If not found, continue to other pages until I find November 2022 orders
4. Once found, calculate the total spending and apply the 20% discount rule

Act: `page.get_by_role("link", name="Page 2").click()`
Think: I need to determine how much was spent at One Stop Market in November 2022. Looking at the current page, I can see a table of orders with dates and order totals. Let me identify all orders from November 2022 and sum them up. From the orders table, I can see these November 2022 orders: 1. Order #000000164 from 11/26/22 with amount $218.17. 2. Order #000000171 from 11/20/22 with amount $133.07. 3. Order #000000183 from 11/11/22 with amount $51.94. So for November 2022, the total spending was: $218.17 + $133.07 + $51.94 = $403.18. Now regarding the 20% discount on orders exceeding $200 in cash: - Only order #000000164 ($218.17) exceeds $200 - If this order was paid in cash, it would qualify for the 20% discount - The discount would be $218.17 * 0.2 = $43.63 - The final amount after discount would be $218.17 - $43.63 = $174.54 - So the total November spending with discount would be $174.54 + $133.07 + $51.94 = $359.55

1. Calculate the total spending for November 2022 from the visible orders
2. Provide the answer through page.stop()

Act: `page.stop("In November 2022, the total spending at One Stop Market was $403.18 (sum of three orders: $218.17 on 11/26/22, $133.07 on 11/20/22, and $51.94 on 11/11/22). If the $218.17 order was paid in cash, it would qualify for the 20% discount, reducing the November total to $359.55")`