

Does The Way You Plan Matter?

An Empirical Study of Planning Representations for LLM Web Agents

Alejandra Zambrano^{1,2}, Sara Vera Marjanović^{2,3}, Imene Kerboua⁴, Xing Han Lù^{2,5},
Leila Kosseim¹,

¹Concordia University, ²Mila - Quebec AI Institute

³University of Copenhagen, ⁴Universite Claude Bernard Lyon, ⁵McGill University,

Correspondence: al_zambr@live.concordia.ca

Abstract

Despite recent advances, LLM-based web agents still struggle with limited exploration, omission of critical steps, and sensitivity to task constraints. Prior work suggests that many of these failures stem from weaknesses in planning, yet the impact of alternative natural language plan representations remains unexplored. To address this, we introduce PLANAHHEAD, a static planner-executor framework that evaluates the impact of plan representation on agent performance. We first automatically categorize WebArena tasks into 3 difficulty levels, enabling consistent difficulty grading without human annotation. Then we systematically evaluate 4 different plan representations on the tasks categorized as hard: sequential subgoals, narrative, pseudocode, and checklist; across different families of multimodal LLM-powered agents (OpenAI, Alibaba, and Google). To account for stochastic variability, we introduce a novel decomposition of agent Success Rate (SR) into 2 interpretable evaluation metrics: Achievement Rate (AR) and Solved-Task Consistency (STC). Our results show that both, the plan formulation and the underlying LLM generating the plan, significantly influence web-agent robustness and task success. Code and dataset are available [here](#).

1 Introduction

Recent advances in large language models (LLMs) have substantially enhanced the capabilities of web agents. Yet, web tasks, where agents must navigate the Internet to fulfill user-defined goals (Deng et al., 2023; Zhou et al., 2023), remain particularly challenging, as LLM-based web agents still struggle with adaptability, real-time interactions, and robustness in dynamic contexts (Chae et al., 2024; Xue et al., 2025).

Planning has emerged as a key strategy for improving web-agent performance, allowing complex tasks to be decomposed into simpler steps (Prasad

et al., 2023; Sodhi et al., 2023). While most existing approaches express plans as sequential lists of subgoals, recent work has explored alternative representations beyond natural language (Wei et al., 2025). However, alternative natural language (NL) representations of plans remain largely unexplored.

In this paper, we challenge both the definition of task difficulty and the representation of plans. Rather than proposing a new planning algorithm, we conduct an empirical study of how different plan representations affect web-agent performance and robustness. We introduce PLANAHHEAD, a framework that redefines difficulty grading and systematically evaluates diverse plan formats across multiple multimodal LLMs from different families (OpenAI, Alibaba, and Google). The contributions are threefold:

1. **Automated task difficulty grading.** We introduce a reproducible pipeline for partitioning benchmark tasks into 3 difficulty levels: *Easy*, *Medium*, and *Hard* without the need for human annotation. This categorization enables more systematic evaluation of planning strategies under varying levels of task complexity.
2. **Evaluation of novel planning representations.** We propose 3 alternative NL-based planning formats: narrative, pseudocode, and requirement checklists, and compare them against sequential subgoal decomposition. Our findings reveal that non-standard formats can, in specific conditions, be beneficial for web agents, particularly on *Hard* tasks.
3. **Novel interpretable decomposition of Success Rate (SR).** We decompose SR into *Achievement Rate* (AR) and *Solved-Task Consistency* (STC). AR, related to empirical pass@k, measures the proportion of tasks solved at least once across repeated runs, acknowledging that agents may not succeed on every attempt. STC measures the repeatability of success among those achieved tasks.

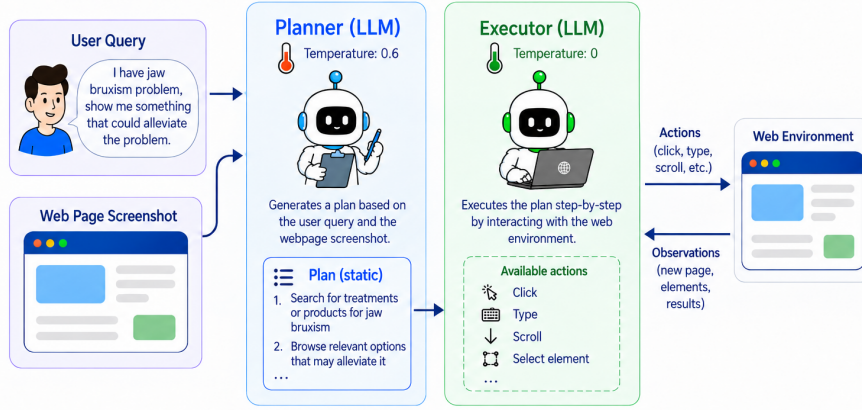


Figure 1: Overview of our framework PLANAHAD

2 Related Works

Planning in Web Agents. Planning through task decomposition, where complex tasks are broken into simpler subgoals (Huang et al., 2024), has emerged as a key capability for tackling complex, multi-step tasks in dynamic environments. Planning can be implemented within a single agent (Sodhi et al., 2023), but more often through a planner-executor separation, where one LLM generates plans and another executes them (Erdogan et al., 2025; Abuelsaad et al., 2024; Agashe et al., 2025; Gur et al., 2023; Zhang et al., 2025a; Marreed et al., 2025; Zhang et al., 2025b). This separation enables the planner to provide high-level structure, while the executor focuses on step-by-step execution. Planners normally follow two types of approaches: static or dynamic. Static plans are generated only once and are not reevaluated during execution, while dynamic plans can be reevaluated either when the executor finds an issue (Prasad et al., 2023) (reactive planning) or at every step of the task execution (Agashe et al., 2025) (proactive planning). Despite extensive research, studies show that LLMs still struggle to generate accurate and reliable plans (Valmeekam et al., 2023). In most existing approaches, plans are expressed as sequential lists of subgoals. However, exploring alternative plan representations, both different natural language forms and non-natural language formats, remains largely unexplored (Wei et al., 2025). Understanding this is particularly important given that planning capabilities have a significant impact on web-agent success (Shlomov et al., 2024).

Multi-Run Evaluation Metrics. Several metrics have been proposed for evaluating web agents, with *Success Rate (SR)* remaining the most widely

used (Yehudai et al., 2025). SR measures the proportion of tasks successfully completed out of the total number of evaluated tasks. However, most existing evaluations report performance from a single run per task, providing limited information about whether a task is consistently solvable or only occasionally achieved across different runs. A related idea appears in code-generation evaluation for LLMs through the *pass@k* metric, introduced by (Chen et al., 2021). In this setting, multiple code samples are generated for each programming problem. A problem is considered solved if at least one of the generated samples passes the corresponding unit tests. The underlying intuition of *pass@k* is highly relevant for stochastic agents: a task may be achievable even if it is not solved in every attempt. However, this perspective remains underexplored in web-agent evaluation.

Web-Agent Benchmarks. Several web-agent benchmarks have been proposed since (Shi et al., 2017), but many introduce limitations for controlled evaluation. In particular, offline benchmarks such as Mind2Web (Deng et al., 2023) and WebLINX (Lu et al., 2024) are built from real-world interactions, but evaluate next-action prediction over recorded trajectories rather than overall goal completion. Live-web benchmarks such as Online-Mind2Web (Xue et al., 2025) provide this interactivity, but introduce reproducibility concerns because of the website’s content and external dependencies may change over time. We therefore use WebArena (Zhou et al., 2023), which provides a dynamic yet controlled environment for web-agent evaluation. Its sandbox setup preserves interactive task execution while supporting reproducible evaluation across a diverse set of challenging tasks.

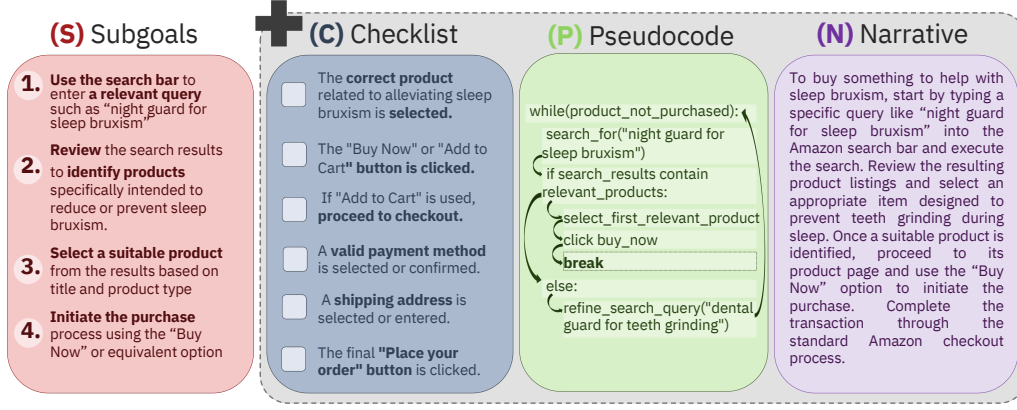


Figure 2: The four representations used in our evaluation of PLANAHEAD. On the left, we show the standard plan of sequential subgoals. In the box on the right, we show three new planning representations. All strategies were generated by GPT-4o given the task "Buy something to alleviate sleep bruxism". Details on prompt templates used to generate each representation are provided in Appendix C.

3 Methodology

3.1 Task Difficulty Categorization

Our primary interest is in tasks that are not achievable by a non-planning agent, as these provide the clearest setting for measuring the added value of planning. Re-evaluating tasks that the baseline agent can already solve offers limited insight into whether planning itself enables new capabilities. Instead, we focus on identifying tasks that the baseline consistently fails and ask whether introducing an explicit planning module can make them achievable. To identify these challenging tasks in a reproducible manner, we start from the *WebArena* test split provided through *BrowserGym* (Chezelles et al., 2025), which consists of 381 tasks. We partition these tasks into three empirical difficulty levels: *Easy*, *Medium*, and *Hard*. Difficulty is estimated using *BrowserGym*’s *GenericAgent* as our baseline agent, a lightweight LLM-based web agent that receives structured browser observations (e.g., AXTree, HTML, and screenshots) and predicts the next action toward the task goal.

To reduce the dependence of the resulting labels on a single model or execution trajectory, we evaluate *GenericAgent* with five different LLM backbones (GPT 4.1, GPT 4.1-mini, GPT 4o-mini, Gemini 2.5 Flash and Qwen 2.5 VL 72B) and run 5 independent trials per model. This results in 25 attempts per task. We set a temperature of 0.4 at each run to induce moderate variation across trajectories while maintaining stable agent behavior.

We assign a task to the *Easy* category only if **all five models succeed in all five trials** (25/25 successful runs), and to the *Hard* category only if

all five models fail in all five trials (0/25 successful runs). All remaining tasks are labeled *Medium*. This deliberately strict criterion makes the extreme categories conservative: *Easy* tasks are those that are consistently solvable across models and runs, whereas *Hard* tasks are those for which the baseline agent fails to find a successful trajectory across a diverse set of 25 attempts.

The resulting partition contains 14 *Easy*, 209 *Medium*, and 158 *Hard* tasks. Our subsequent analysis focuses on the *Hard* subset. Additional details and analyses of the task grading are provided in Appendix A.

3.2 Planner-Executor Modules

PLANAHEAD follows a static planner-executor framework (see Section 2). First, an LLM planner generates a plan from the *WebArena* task goal and a screenshot of the initial browser state. This step is performed once at the beginning of the task, producing a static plan. As shown in Figure 1, the plan is then provided to the executor LLM together with the current browser state, and the executor predicts low-level actions at each step. The planner is sampled with a temperature of 0.6 to encourage plan diversity, while the executor operates deterministically (temperature of 0) to ensure stable action selection. This setup ensures that performance differences arise from the plan representation rather than action-level stochasticity. The executor is allowed up to 30 actions per task, and success is evaluated using *WebArena*’s automatic reward.

As a dynamic planning baseline, we use the *GenericAgent* UsePlan capability, which combines plan generation and action prediction by producing

a new sequential plan at each step by a single agent.

3.3 Planning Representations

Using the 158 *Hard* tasks, we evaluate 4 plan representations for communicating high-level plans to the executor: a standard sequential list of subgoals and 3 novel representations: Requirement Checklist, Pseudocode, and Narrative. These representations differ in abstraction level, information ordering, and the degree of flexibility given to the executor. Figure 2 illustrates each representation.

3.4 Evaluation Metrics

As discussed in Section 2, the stochasticity of LLM-based agents provides an opportunity to explore multiple trajectories toward the same goal. However, standard SR is typically defined over a single run per task and therefore does not capture this potential. When evaluating multiple runs, we instead want to distinguish between whether a task can be achieved and how consistently it can be achieved. We therefore decompose SR into two complementary metrics:

Achievement Rate (AR) measures task-level achievability in a set of T tasks over R independent runs. A task is considered achieved if at least one run succeeds:

$$AR = \frac{1}{T} \sum_{t=1}^T \left[\bigvee_{i=1}^R \text{Reward}(t, i) \right] \quad (1)$$

Where $\text{Reward}(t, i)$ is a binary indicator of whether task t succeeds in run i .

Solved-Task Consistency (STC) For tasks that were achieved at least once (i.e. $A = \{t \mid \exists i \in \{1, \dots, R\} : \text{Reward}(t, i) = 1\}$), STC measures performance consistency across runs:

$$STC = \frac{1}{|A| \times R} \sum_{t \in A} \sum_{i=1}^R \text{Reward}(t, i) \quad (2)$$

Together, these metrics provide an exact decomposition of Success Rate $SR = AR \times STC$.

As an illustrative example, consider three web tasks, each evaluated over five runs. Suppose that only one task succeeds, and that it succeeds in two out of its five runs, while the other two tasks fail in all runs. Using the standard SR, the 15 task attempts are aggregated together, giving $SR = \frac{2}{3 \times 5} = 13.3\%$. In contrast, AR captures achievability at the task level giving an $AR = \frac{1}{3} = 33.3\%$

because one out of the three tasks is achieved. STC then measures reliability among the achieved tasks resulting in $STC = \frac{2}{1 \times 5} = 40\%$.

4 Results and Analysis

We evaluate planning strategies on the 158 *Hard* tasks. For each task and strategy, we run $R = 5$ independent trials to observe execution variability and distinguish tasks that are occasionally solvable from those that are consistently solvable. In our experiments we use a subset of the LLM backbones used for the task categorization: GPT 4.1-mini, Qwen-2.5-VL-72B, and Gemini 2.5 Flash, evaluated as both planner and executor, as well as all planner-executor combinations. Table 1 shows the performance measured using AR and STC percentages. We also include a dynamic planning baseline (see Section 3) to compare static and dynamic planning under identical conditions. Key findings are summarized below:

Planning makes unachievable tasks achievable.

Because the *Hard* subset consists exclusively of tasks that were not solved by any non-planning agent across all runs, any AR greater than zero reflects task completion attributable to the introduction of planning. Across all configurations and representations, $AR > 0$ (see Table 1). This demonstrates that changing the planning strategy (regardless of the representation or configuration) allows agents to solve tasks that were consistently unsolved under the baseline setting.

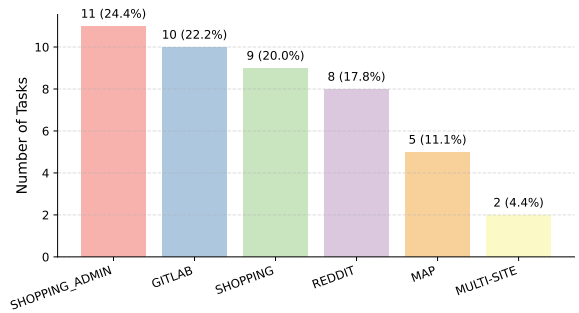


Figure 3: Task distribution of achieved tasks aggregating all types of planning.

Figure 3 further illustrates these gains by showing the distribution of achieved tasks across *WebArena* categories. Out of the 158 tasks classified as *Hard*, 45 tasks were successfully completed (approximately 28%) at least once by one or more of the planning configurations. These achieved tasks

#	Configuration		Dynamic		Static (Ours)							
	Planner	Executor	Sequential		Sequential		Checklist		Pseudocode		Narrative	
			AR	STC	AR	STC	AR	STC	AR	STC	AR	STC
1	GPT 4.1-mini	GPT 4.1-mini	5.1	42	5.1	42	4.4	35	4.4	34	5.1	75
2	Qwen 2.5-VL 72B	GPT 4.1-mini	-	-	5.4	31	4.2	40	5.4	31	6.6	60
3	Gemini 2.5 Flash	GPT 4.1-mini	-	-	7.0	45	3.2	36	4.4	51	5.7	53
4	Qwen 2.5-VL 72B	Qwen 2.5-VL 72B	5.1	55	2.5	85	5.1	75	3.2	80	3.8	63
5	GPT 4.1-mini	Qwen 2.5-VL 72B	-	-	5.1	35	7.5	30	5.0	45	5.1	45
6	Gemini 2.5 Flash	Qwen 2.5-VL 72B	-	-	3.2	32	5.7	40	3.8	40	4.4	40
7	Gemini 2.5 Flash	Gemini 2.5 Flash	9.5	39	5.7	53	7.6	58	8.2	43	7.6	73
8	GPT 4.1-mini	Gemini 2.5 Flash	-	-	10.7	47	8.9	44	7.0	53	9.5	43
9	Qwen 2.5-VL 72B	Gemini 2.5 Flash	-	-	8.9	50	7.0	58	4.4	77	8.9	50

Table 1: Comparison of percentage AR and STC across planning strategies on the *Hard* set. In bold are the best results (of AR and STC) per configuration and highlighted in grey is the best configuration overall

are distributed across all categories, suggesting that the benefits of planning are not limited to a single domain, but appear across a range of task types.

LLMs Benefit from Distinct Plan Representations. For example, when GPT 4.1-mini is used as executor, Narrative plans generally exhibit strong performance, achieving the highest AR in 2 configurations and particularly high STC across configurations (see rows 1,2 and 3 of Table 1). A plausible explanation is that GPT-style models, trained extensively on natural language text, may better leverage coherent descriptive instructions when interpreting plans. In contrast, Qwen-2.5-VL performs best with checklist-based representations (see rows 4,5,6), which provide explicit and structured guidance. Gemini performs competitively across multiple representations and notably benefits from non-standard formats (see rows 7,8,9). Interestingly, Pseudocode, a representation not designed for human readability, yields one of the highest ARs observed when Gemini is used as both planner and executor (see row 7). This suggests that structured control flow may provide useful information for LLM reasoning, even when expressed in simplified pseudo-programmatic form.

Taken together, these results demonstrate that the effectiveness of planning is influenced by their representational form and that the traditional format of a sequential numbered list might not always be optimal.

Planner-Executor Separation Reveals Complementary Model Strengths. Separating planning and execution exposes clear differences in model suitability for each role. Across configurations, mixed planner-executor pairs generally outperform homogeneous setups in AR. Notably, GPT 4.1-mini as planner combined with Gemini as ex-

ecutor achieves the strongest overall performance in our evaluation (see row 8 of Table 1). While we typically see a drop in STC given an increase in AR (given 2), this occurs *without* a substantial increase in AR (compare row 4 and 6). This pattern indicates that some models, such as GPT 4.1-mini, are more effective at plan generation, whereas others, like Gemini, may be better suited for low-level action execution. These findings support modular agent architectures that exploit complementary model strengths and enable flexible model combinations to balance performance and cost.

Static Planning is more Consistent. Even without the possibility of regenerating or updating plans during execution, static plans achieve higher consistency (STC) than single-agent dynamic plans. Qualitative trace analysis shows that dynamic agents often enter action loops, lose track of task progress, or prematurely mark subtasks as completed (see Appendix D). Static planner-executor is a simple setup that allows the executor to focus exclusively on action selection.

5 Conclusion

In this work, we revisited the role of planning in web agents, challenging the assumption that plans must take the form of sequential subgoals. We introduced PLANAhead, an automated and reproducible approach for identifying challenging web tasks, and used it to conduct a controlled empirical study of planning representations with a new decomposition of SR into two interpretable metrics. Our experiments show that planning effectiveness depends not only on the models chosen but also on the representational form of the plan. Overall, our findings position planning as a crucial but flexible design dimension.

Limitations

Despite our comprehensive list of planning strategies, our study has several limitations. We restricted our evaluation to a small set of LLM backends (GPT-4.1-mini, Qwen-2.5-VL and Gemini 2.5 Flash), a single benchmark (WebArena), and a limited number of runs ($R = 5$). Expanding to a broader range of models, tasks, and evaluation scales would provide a more comprehensive picture of planning effectiveness.

References

- Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. 2024. [Agent-e: From autonomous web navigation to foundational design principles in agentic systems](#). *arXiv preprint arXiv:2407.13032*.
- Saaket Agashe, Kyle Wong, Vincent Tu, Jiachen Yang, Ang Li, and Xin Eric Wang. 2025. [Agent S2: A compositional generalist-specialist framework for computer use agents](#). *arXiv preprint arXiv:2504.00906*.
- Hyungjoo Chae, Namyoun Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2024. [Web agents with world models: Learning and leveraging environment dynamics in web navigation](#). *arXiv preprint arXiv:2410.13232*.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. [Evaluating large language models trained on code](#). *arXiv preprint arXiv:2107.03374*.
- Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omid Shayan, Lawrence Keunho Jang, Xing Han Lu, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. 2025. [The BrowserGym ecosystem for web agent research](#).
- Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. [Mind2Web: Towards a generalist agent for the web](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 28091–28114.
- Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. 2025. [Plan-and-act: Improving planning of agents for long-horizon tasks](#). *arXiv preprint arXiv:2503.09572*.
- Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. [A real-world webagent with planning, long context understanding, and program synthesis](#). *arXiv preprint arXiv:2307.12856*.
- Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. [Understanding the planning of LLM agents: A survey](#). *arXiv preprint arXiv:2402.02716*.
- Xing Han Lu, Zdeněk Kasner, and Siva Reddy. 2024. [WebLINX: Real-World Website Navigation with Multi-Turn Dialogue](#). In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, volume 235 of *Proceedings of Machine Learning Research*, pages 33007–33056, Vienna, Austria. PMLR.
- Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Aviad Sela, Asaf Adi, and Nir Mashkif. 2025. [Towards enterprise-ready computer using generalist agent](#). *arXiv preprint arXiv:2503.01861*.
- Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2023. [ADAPT: As-needed decomposition and planning with language models](#). *arXiv preprint arXiv:2311.05772*.
- Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. [World of bits: An open-domain platform for web-based agents](#). In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3135–3144.
- Segev Shlomov, Aviad Sela, Ido Levy, Liane Galanti, Roy Abitbol, and 1 others. 2024. [From grounding to planning: Benchmarking bottlenecks in web agents](#). *arXiv preprint arXiv:2409.01927*.
- Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. 2023. [SteP: Stacked LLM policies for web actions](#). *arXiv preprint arXiv:2310.03720*.
- Megh Thakkar, Nicolas Chapados, Christopher Pal, and 1 others. 2025. [WebArena verified: Reliable evaluation for web agents](#). In *NeurIPS 2025 Posters*.
- Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. [On the planning abilities of large language models-a critical investigation](#). In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 75993–76005.
- Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. [PlanGenLLMs: A modern survey of LLM planning capabilities](#). *arXiv preprint arXiv:2502.11221*.
- Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. 2025. [An illusion of progress? assessing the current state of web agents](#). In *Proceedings of the Second Conference on Language Modeling (COLM)*.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2025. [Survey on evaluation of LLM-based agents](#). *arXiv preprint arXiv:2503.16416*.

Cong Zhang, Xin Deik Goh, Dexun Li, Hao Zhang, and Yong Liu. 2025a. [Planning with multi-constraints via collaborative language agents](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10054–10082.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. 2025b. [WebPilot: a versatile and autonomous multi-agent system for web task execution with strategic exploration](#). In *Proceedings of the Thirty-Ninth AAAI Conference on Artificial Intelligence (AAAI-25)*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2023. [WebArena: A realistic web environment for building autonomous agents](#). In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*.

A Results for Task Categorization

To construct the *Hard* subset (Section 3), we evaluated 5 backbone models over $R = 5$ independent runs on all 381 WebArena test tasks.

Model	Min	Max	Average
GPT 4o-mini	16.3	20.7	17.9
GPT 4.1 mini	24.1	33.9	29.0
GPT 4.1	31.5	34.1	32.9
Qwen 2.5-VL 72B	22.8	23.9	23.5
Gemini 2.5 Flash	29.4	32.0	30.7

Table 2: Success Rate (SR) on test set of *BrowserGym* on 5 baseline models.

Table 2 reports the resulting Success Rates (SR). The 3 strongest backbones: GPT-4.1, Qwen-2.5-VL and Gemini 2.5 achieve high success rate which motivated the selection done for the main experiments.

After running these experiments the division of task concluded in: 158 *Hard* task, 14 *Easy* tasks and 209 *Medium* tasks. To better understand the distribution of the *Hard*, *Medium* and *Easy* set we separated how the partition would be if only one model was considered.

In Table 3, we can further appreciate the reason for the small number of *Easy* tasks. Recall from Section 3, that our definition for *Easy* tasks require that all models solve the task in all runs R ; this

Year of release	Model	Hard	Medium	Easy
July, 2024	GPT 4o-mini	268	84	29
February, 2025	Qwen 2.5-VL 72B	240	87	54
April, 2025	GPT 4.1-mini	216	103	62
April, 2025	GPT 4.1	213	81	87
June, 2025	Gemini 2.5 Flash	219	88	74

Table 3: Number of *Hard*, *Medium* and *Easy* tasks per model.

strict definition results in only some of the tasks being in this category, meaning a task that is easy for one model is rarely also easy for another one. Additionally, this table shows that newer models (even with just a few months of difference) tend to perform better. This highlights that using a model as old as 4o-mini (released in July 2024) significantly lowers the number of easy tasks. For the interest of this project, studying the positive effects of planning, even an older model, is still interesting.

A.1 Hard Task Distribution

To better understand the nature of difficult tasks, Figure 4 presents the distribution of task difficulty across domains. Intuitively, the *Hard* subset contains most multi-site tasks, tasks that require cross-domain coordination and persistent memory across navigation steps. A substantial portion also comes from the Shopping and GitLab environments, this suggests that difficulty is closely tied to long-horizon reasoning as well as internal issues in these environments aligned to what is noted in (Thakkar et al., 2025).

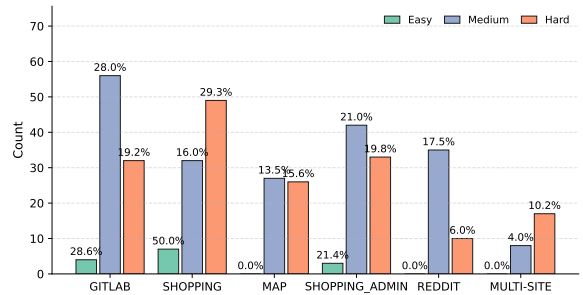


Figure 4: Distribution of *Easy*, *Medium*, and *Hard* tasks across WebArena domains. Percentages indicate each domain’s share within a difficulty tier.

A.2 Task Categorization Stability Analysis

To ensure that the *Hard* subset was not merely an artifact of “unlucky” runs, we performed a sensitivity analysis by recomputing the task partitions with different numbers of runs, from $R = 1$ to $R = 5$.

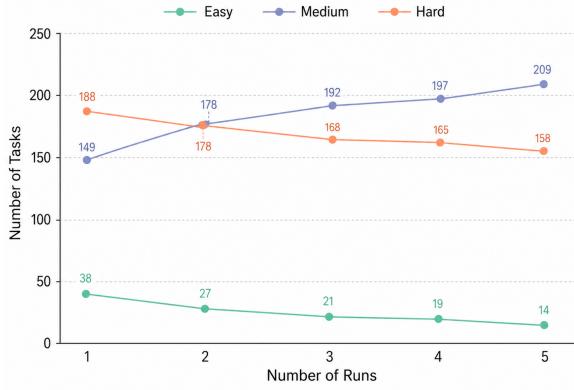


Figure 5: Task distribution across difficulty categories as the number of runs (R) increases.

Figure 5 shows how the number of tasks in each difficulty category changes as R increases. With a single run ($R = 1$) 188 tasks are classified as *Hard*, however, as the number of runs increases, the size of the *Hard* subset decreases to 158 tasks at $R = 5$, while the *Medium* subset increases from 149 to 209 tasks. This indicates that several tasks that appear unsolved under a single run become solvable at least once under repeated stochastic execution. This analysis leads to two main conclusions:

1. Single-run evaluation can underestimate both task achievability and variability. Approximately 16% of the tasks labeled *Hard* under a single run are no longer classified as *Hard* when evaluated over five runs. This suggests that some tasks are not consistently unsolvable (i.e. *Hard*), but instead require repeated attempts or favorable stochastic trajectories to be solved at least once (i.e. *Medium*). The opposite effect can also be observed in the *Easy* set when a task solved in a single run is not necessarily solved reliably across repeated runs. This is reflected in the decreasing number of *Easy* tasks as (R) increases, from 38 tasks at ($R=1$) to 14 tasks at ($R=5$). Together, these trends show that single-run evaluation can overestimate consistency while underestimating achievability.
2. Multi-run grading improves reliability but does not guarantee convergence. Although overlap with the final *Hard* set increases monotonically as the number of runs grows, Figure 5 does not demonstrate full convergence. The size of the *Hard* subset continues to decrease as R increases (from 188 at $R = 1$ to 158 at $R = 5$), suggesting that additional

runs could further reduce the number of tasks classified as *Hard*.

In other words, the *Hard* subset should be interpreted as conditionally hard given $R = 5$ runs and the selected model family. This subset represents tasks that are consistently unsolved under these specific conditions, but not necessarily tasks that are fundamentally unsolvable. Despite this limitation, $R = 5$ already substantially reduces stochastic misclassifications compared to single-run grading. The resulting subset provides a sufficiently conservative and reproducible evaluation setting for studying whether planning can unlock tasks that non-planning agents consistently fail to solve.

B Strategy Sensitivity Results

Although absolute AR values on the *Hard* set seem small, bootstrap analysis reveals that these estimates can be highly stable (see Table 4 and 5). It is important to note that the *Hard* subset is defined as tasks for which all baseline agents fail in all runs, any $AR > 0$ already corresponds to solving tasks that a single agent (even a powerful model like GPT 4.1) could not solve. The bootstrap intervals show that, for the strategies with observed AR improvements, the entire confidence interval remains above zero, confirming that these gains are not explained by stochastic noise.

Importantly, AR and STC capture different aspects of variability: STC measures within-task reliability, while bootstrap CIs reflect uncertainty in the metric itself due to finite sampling. Using both, provides a clearer picture of agent robustness than SR alone, which treats all runs as independent and cannot distinguish “rarely solvable tasks” from “completely unsolved tasks.”

Configuration						
#	Planner	Executor	Sequential	Checklist	Pseudocode	Narrative
1	GPT 4.1-mini	GPT 4.1-mini	3.59 – 6.59	1.20 – 4.79	2.40 – 4.79	5.39 – 6.59
2	Qwen 2.5-VL 72B	Qwen 2.5-VL 72B	2.40 – 2.99	2.40 – 4.79	3.59 – 3.59	2.99 – 4.19
3	Gemini 2.5 Flash	Gemini 2.5 Flash	3.80 – 5.70	5.06 – 7.59	4.43 – 8.23	6.33 – 7.59
4	GPT 4.1-mini	Qwen 2.5-VL 72B	2.99 – 5.99	4.79 – 8.38	2.40 – 5.39	3.59 – 5.39
5	Qwen 2.5-VL 72B	GPT 4.1-mini	2.99 – 5.99	1.80 – 4.19	1.80 – 5.30	4.19 – 6.59
6	Qwen 2.5-VL 72B	Gemini 2.5 Flash	5.70 – 8.86	5.70 – 6.96	3.80 – 4.43	5.70 – 8.86
7	Gemini 2.5 Flash	Qwen 2.5-VL 72B	1.27 – 3.16	3.16 – 5.70	1.90 – 3.80	2.52 – 4.43
8	Gemini 2.5 Flash	GPT 4.1-mini	4.43 – 6.96	1.27 – 3.16	2.53 – 4.43	3.16 – 5.70
9	GPT 4.1-mini	Gemini 2.5 Flash	6.96 – 10.76	5.06 – 8.86	4.43 – 6.96	5.70 – 9.49

Table 4: Bootstrap 95% confidence intervals for AR on the *Hard* subset with 1,000 resamples

C Plan Representation Prompts

The input of PLANAHHEAD’s planner consists of only a screenshot and the goal. Based on these two, the planner has to create a plan to complete the task.

Configuration		Sequential	Checklist	Pseudocode	Narrative
#	Planner				
1	GPT 4.1-mini	40.00 – 72.00	32.50 – 84.00	34.94 – 70.00	70.91 – 88.89
2	Qwen 2.5-VL 72B	68.00 – 96.00	30.00 – 75.00	70.00 – 93.33	57.14 – 88.00
3	Gemini 2.5 Flash	53.33 – 80.00	58.33 – 84.44	43.08 – 77.14	68.33 – 92.73
4	GPT 4.1-mini	34.29 – 66.67	32.31 – 67.50	35.56 – 80.00	40.00 – 70.00
5	Qwen 2.5-VL 72B	28.00 – 62.86	40.00 – 76.00	30.00 – 69.00	54.55 – 87.50
6	Qwen 2.5-VL 72B	49.18 – 74.00	49.18 – 74.00	68.57 – 94.29	50.00 – 76.00
7	Gemini 2.5 Flash	26.67 – 70.00	37.78 – 75.00	36.67 – 76.00	40.00 – 73.33
8	Gemini 2.5 Flash	45.45 – 74.29	36.00 – 70.00	45.71 – 80.00	47.50 – 76.03
9	GPT 4.1-mini	44.29 – 71.67	40.00 – 68.57	53.55 – 80.00	36.00 – 69.09

Table 5: Bootstrap 95% confidence intervals for STC on the *Hard* subset with 1,000 resamples

Each of the strategies proposed has their individual prompt but they all follow a similar format.

The original prompt was inspired from (Agashe et al., 2025) and adapted to each planning strategy, having these prompts as results.

Strategy 1, sequential plan shown in Figure 7.

Strategy 2, checklist shown in Figure 8.

Strategy 3, pseudocode shown in Figure 9.

Strategy 4, narrative shown in Figure 10.

D Single-Agent Plan Creation Failure Cases

When a single agent both produces the plan and predicts the next action it can sometimes get confused at the step it is currently at and start repeating the same action entering what we call an *action loop* or get over confident thinking a step was completed when it was not. Some examples are shown below.

D.1 Shopping Admin Case

The task webarena.459: "Reduce the price of this product by 10%". The task is initiated at the product page in the shopping admin website as shown in Figure 6a.

At the next step the agent correctly calculates the 10% and reduces the price (see Figure 6b).

However, instead of saving the reduced priced, it proceeds to reduce the price again, see Figure 6c and again, see Figure 6d and keeps reducing the price until step 30. A similar behaviour happens with task webarena.462 which is to increase the price.

D.2 Map Case

Task webarena.54 states "How long does it take to walk from Carnegie Mellon University to Univ of Pittsburgh?"

In this task the agent starts in the correct manner picking the right origin, destination and form of travel. However, at the 4th step when the agent is supposed to click go. The plan in the prompt is as following:

Plan:

You just executed step 3 of the previously proposed plan:

1. Change the mode of transportation to "Foot (OSRM)".
2. Click the "Go" button to get walking directions.
3. Extract the walking time from the results.

After reviewing the effect of your previous actions, verify if your plan is still relevant and update it if necessary.

As shown in the plan above, the step of 'Clicking Go' is ignored and is assumed completed. The agent proceeds to just "wait" for the results to appear instead of trying to click go again it proceeds to just wait (noop) for the remaining 30 turns.

D.3 Multiple Runs are Important

As shown in Tables 1 and 6, relying solely on Success Rate (SR) can obscure meaningful differences. SR can fail to indicate how many tasks an agent can actually solve, whereas AR and STC together capture more directly both achievability and robustness. Given the stochastic nature of LLM-based agents, metrics that incorporate multiple runs are essential.

E Additional Experiments and Results

E.1 Easy Set Runs

As a sanity check, we evaluate the *Easy* subset to confirm that planning does not degrade performance on tasks that are already solvable. Experiments on the *Easy* subset (see Table 7) revealed that low-level plans can limit exploration and generalization. Executors sometimes failed when constrained to search for exact words or sections, even though more flexible reasoning would have sufficed. High-level guidance appears crucial for balancing structure with adaptability.

#	Configuration		Dynamic		Static							
	Planner	Executor	SR	SE	Sequential	Sequential	Checklist		Pseudocode		Narrative	
			SR	SE	SR	SE	SR	SE	SR	SE	SR	SE
1	GPT 4.1-mini	GPT 4.1-mini	2.1	0.5	2.1	0.6	1.5	0.4	1.5	0.5	3.8	0.7
2	Qwen 2.5-VL 72B	GPT 4.1-mini	-	-	1.7	0.5	1.7	0.4	1.7	0.4	4.0	0.7
3	Gemini	GPT 4.1-mini	-	-	3.1	0.6	1.1	0.4	2.2	0.5	3.0	0.6
4	Qwen 2.5-VL 72B	Qwen 2.5-VL 72B	2.8	0.6	2.1	0.5	3.8	0.4	2.6	0.6	2.4	0.6
5	GPT 4.1-mini	Qwen 2.5-VL 72B	-	-	1.8	0.5	2.2	0.6	2.2	0.5	2.3	0.5
6	Gemini	Qwen 2.5-VL 72B	-	-	1.0	0.4	2.3	0.5	1.5	0.4	1.8	0.5
7	Gemini 2.5 Flash	Gemini 2.5 Flash	3.7	0.8	3.0	0.6	4.4	0.7	3.5	0.7	5.6	0.8
8	GPT 4.1-mini	Gemini 2.5 Flash	-	-	5.1	0.8	3.9	0.7	3.7	0.7	4.1	0.7
9	Qwen 2.5-VL 72B	Gemini 2.5 Flash	-	-	4.5	0.7	4.1	0.7	3.4	0.6	4.5	0.7

Table 6: Comparison of Success Rate (SR) and Standard Error (SE) across planning strategies on the *Hard* subset

#	Configuration		Sequential		Checklist		Pseudocode		Narrative	
	Planner	Executor	AR	STC	AR	STC	AR	STC	AR	STC
1	GPT 4.1-mini	GPT 4.1-mini	100	100	100	100	100	100	100	100
2	Qwen 2.5-VL 72B	Qwen 2.5-VL 72B	100	100	100	100	100	96.0	85.7	98.0
3	Gemini 2.5 Flash	Gemini 2.5 Flash	100	100	100	100	100	100	100	100

Table 7: Achievement Rate (AR) and Solved-Task Consistency (STC) of different strategies on the *Easy* subset by model

(a) Step 0

(b) Step 1

(c) Step 2

(d) Step 3

Figure 6: Example of single-agent planner-executor on task webarena.459. The agent incorrectly keeps reducing the price.

You are an expert planning agent for solving GUI navigation tasks. Using the plan that you output, another language model will execute the task.

You are provided with:

1. The state of the computer screen through a desktop screenshot Your responsibilities:
1. Generate a new plan to complete the task by decomposing the task into meaningful subtasks of logically ordered chunks.
2. Ensure the plan is concise and contains only necessary steps.
3. Carefully observe and understand the current state of the computer before generating your plan.
4. Avoid including steps in your plan that the task does not ask for.
5. Generate a thought process or explanation about the plan generated.

Below are important considerations when generating your plan:

1. Provide the plan in a step-by-step numbered format with sequential subtasks.
2. Provide high-level subtasks and avoid over-specifying actions—this isn't a low-level how-to (e.g. avoid specifying mouse clicks). This is a logical task breakdown that captures essential steps and decisions. The executor will identify the fine-grained actions to be taken whilst interacting with the GUI using your subgoals.
3. Do not include verification steps in your planning. Steps that confirm or validate other subtasks should not be included.
4. Do not include optional steps in your planning. Your plan must be as concise as possible.
5. Do not include unnecessary steps in your planning. If you are unsure if a step is necessary, do not include it in your plan.
6. Your plan ****must**** be end-to-end meaning it should entail instructions from the initial state till the goal state.

Below is an example of a plan created using subtask decomposition, given the request "Book a round-trip flight from New York to San Francisco for next weekend, departing Friday and returning Sunday, on the cheapest available airline.":

1. Open a flight booking website (e.g., Google Flights, Expedia, etc.)
 2. Input departure city as New York and destination as San Francisco
 3. Set departure date to next Friday and return date to next Sunday
 4. Search for round-trip flights
 5. Sort results by price
 6. Select the cheapest available round-trip option that fits the criteria
 7. Proceed to booking and complete the purchase
- Important formatting guidelines:
1. Make sure the formatting of your answer is correct. It should follow the following HTML tags format <observation> Description and understanding of current state </observation> <plan> Generated Plan </plan> <thought> Thought process </thought>
 2. Make sure to provide the correct start and end tags

Figure 7: Prompt for sequential plans.

You are an expert planning agent for solving GUI navigation tasks. Using the plan that you output, another language model will execute the task. You are provided with: 1. The state of the computer screen through a desktop screenshot

Your responsibilities:

1. Generate a checklist of all necessary requirements that need to be completed in order to achieve the goal.
2. Ensure this checklist is concise and contains only necessary requirements
3. Carefully observe and understand the current state of the computer before generating your checklist
4. Generate a thought process or explanation about the checklist generated

Below are important considerations when generating your checklist:

1. Provide the plan in a unordered checklist format. This is not necessarily sequential, but outlines requirements that must be met. The executor will identify the order of fine-grained actions to be taken whilst interacting with the GUI based on the provided checklist.
2. Do not include verification steps. Steps that confirm or validate other subtasks should not be included.
3. Do not include optional requirements. Your checklist must be as concise as possible.
4. Do not include unnecessary requirements. If you are unsure if a requirement is necessary, do not include it in your checklist.

Below is an example of a checklist given the request "Sign up for the weekly newsletter on the website.":

- ☐ The email input field is filled with a valid email address.
- ☐ The "Subscribe" or "Sign Up" button has been clicked.
- ☐ A confirmation message or success banner is visible after submission.
- ☐ The user is not required to solve a CAPTCHA.
- ☐ There are no visible error messages on the page.
- ☐ The checkbox for accepting terms (if present) is checked.

Important formatting guidelines:

1. Make sure the formatting of your answer is correct. It should follow the following HTML tags format <observation> Description and understanding of current state </observation> <plan> Generated Checklist </plan> <thought> Thought process </thought>
2. Make sure to provide the correct start and end tags

Figure 8: Prompt for checklist plans.

You are an expert agent in web navigation and you are able to foresee multiple possible trajectories to achieve a navigation task. Using the plan that you output, another language model will execute the task.

You are provided with:

1. The state of the computer screen through a desktop screenshot

Your responsibilities:

1. Generate an algorithm style plan in order to achieve the goal in a pseudocode style with abstract functions depicting high-level actions.
2. Ensure this pseudocode plan is as concise as possible avoiding complex nested if statements.
3. Carefully observe and understand the current state of the computer before generating your pseudocode
4. Generate a thought process or explanation about the pseudocode plan generated Below are important considerations when generating your pseudocode:

1. Provide the pseudocode plan in a readable format. Make function names intuitive and descriptive. The executor will then determine the fine-grained actions to take using these function names.

2. Consider potential obstacles or layout variations the executor may encounter whilst completing the task.

3. Do not include verification sections. Steps that confirm or validate other subtasks should not be included.

4. Do not include unnecessary parts in your pseudocode. If you are unsure if a requirement is necessary, do not include it.

Below is an example of the correct formatting and level of abstraction for the request "Add red coat product to cart and wish list":

```
while(red_coat_not_in_cart and red_coat_not_in_wish_list)
    search_for_red_coat
    if item found:
        click add to cart
        if item in cart:
            find_and_click_add_product_to_wishlist
            if item_added_to_wishlist:
                break
            else:
                find_add_to_wishlist_button
        else:
            find_add_to_cart_button
    else:
        update search query
```

Important formatting guidelines:

1. Make sure the formatting of your answer is correct. It should follow the following HTML tags format <observation> Description and understanding of current state </observation> <plan> Generated Pseudocode Plan </plan> <thought> Thought process </thought>
2. Make sure to provide the correct start and end tags

Figure 9: Prompt for algorithmic pseudocode plans.

You are an expert planning agent for solving GUI navigation tasks. Using the plan that you output, another language model will execute the task. You are provided with: 1. The state of the computer screen through a desktop screenshot Your responsibilities:

1. Generate a paragraph describing what the agent should do in order to achieve the given goal.
2. Ensure this paragraph is concise and contains only necessary information and suggestions to help the agent achieve the goal.
3. Carefully observe and understand the current state of the computer before generating your paragraph
4. Generate a thought process or explanation about the paragraph generated

Below are important considerations when generating your paragraph:

1. Provide the paragraph in plain text. There is no need for stylistic formatting.
2. Do not include verification steps. Steps that confirm or validate other subtasks should not be included.
3. Do not include unnecessary requirements. If you are unsure if a requirement is necessary, do not include it in your checklist.

Below is an example of a paragraph describing how an executor agent should fulfill the request

"Book a round-trip flight from New York to San Francisco for next weekend, departing Friday and returning Sunday, on the cheapest available airline.": To book a round-trip flight from New York to San Francisco for next weekend, you would start by opening a flight booking site like Google Flights or Expedia. Input New York as the departure city and San Francisco as the destination, then set the departure for Friday and the return for Sunday. After initiating the search, sort the results by price to find the cheapest option. Once identified, select the flight and proceed to the booking page to enter passenger and payment details. Finally, complete the purchase, ensuring no obstacles arise during the process.

Important formatting guidelines:

1. Make sure the formatting of your answer is correct. It should follow the following HTML tags format <observation> Description and understanding of current state </observation> <plan> Generated Paragraph </plan> <thought> Thought process </thought>
2. Make sure to provide the correct start and end tags

Figure 10: Prompt for narrative plans.