Does The Way You Plan Matter?
An Empirical Study of Planning Representations for LLM Web Agents

Anonymous ACL submission

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 representation remains unexplored. To address this, we introduce PLANAHEAD, a static planner-executor framework that evaluates the impact of plan representation in 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; accross different families of multimodal LLM powered agents (OpenAI, Alibaba, and Google). To account for stochastic variability, we introduce two novel 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 PLANAHEAD, 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 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. **We introduce 2 novel evaluation metrics for web agents.** *Achievement Rate (AR)*, a metric that goes beyond the binary notion of *Success Rate* (SR). AR treats tasks as achievable or not across multiple stochastic runs, acknowledging that agents may not succeed on every attempt. We also propose *Solved-Task Consistency (STC)*, a robustness metric that measures the reliability of performance across trials.

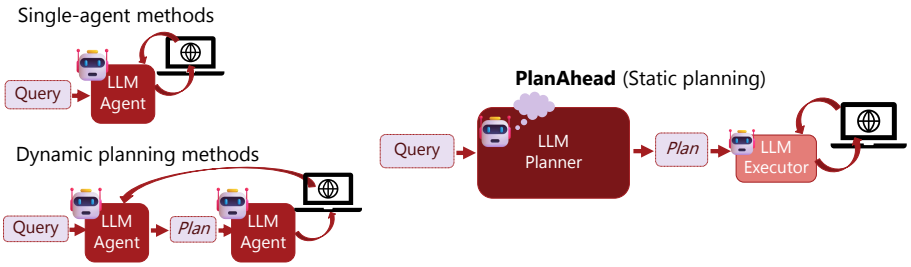


Figure 1: Overview of PLANAHEAD, in comparison to single-agent and proactive planning frameworks

2 Related Works

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 plan 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, remain 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).

3 Methodology

3.1 Benchmark Selection

Several web-agent benchmarks have been proposed since (Shi et al., 2017), but many introduce limitations for controlled evaluation. In particular, benchmarks relying on live websites (e.g. (Deng et al., 2023)) raise reproducibility and privacy concerns due to website updates and external dependencies. We therefore intentionally focus on *WebArena* (Zhou et al., 2023), which provides a closed and reproducible environment while still capturing diverse and realistic web interactions across multiple domains. While benchmarks such as *WorkArena* (Drouin et al., 2024) primarily target enterprise-oriented workflows, *WebArena* includes a broader range of consumer-facing websites, making it well suited for studying planning behavior representative of general public web use.

3.2 Task Difficulty Categorization

To identify challenging tasks in a reproducible manner, we start from the *WebArena* test split as provided in *BrowserGym* (Chezelles et al., 2025), which consists of 381 tasks. We partition these tasks into three difficulty levels: *Easy*, *Medium*, and *Hard*. Difficulty labels are assigned using BrowserGym's *GenericAgent*, a lightweight LLM-based web agent that receives structured browser signals (e.g AXTree, HTML, screenshots) and predicts the next action towards the goal. We evaluate *GenericAgent* using 5 different LLM backbones, running $N = 5$ independent trials per model with a decoding temperature of 0.4. This temperature balances stochasticity and stability, enabling variability across runs without degrading performance. Full results are reported in Appendix A (Table 2).

Tasks are labeled as *Easy* if all models succeed in every trial (100% success), *Hard* if all models fail in every trial (0% success), and *Medium* otherwise. This procedure yields 14 *Easy*, 209 *Medium* and 158 *Hard* tasks. Our analysis focuses on the *Hard* subset, as these tasks provide the most informative setting for evaluating the impact of planning under challenging conditions.

3.3 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

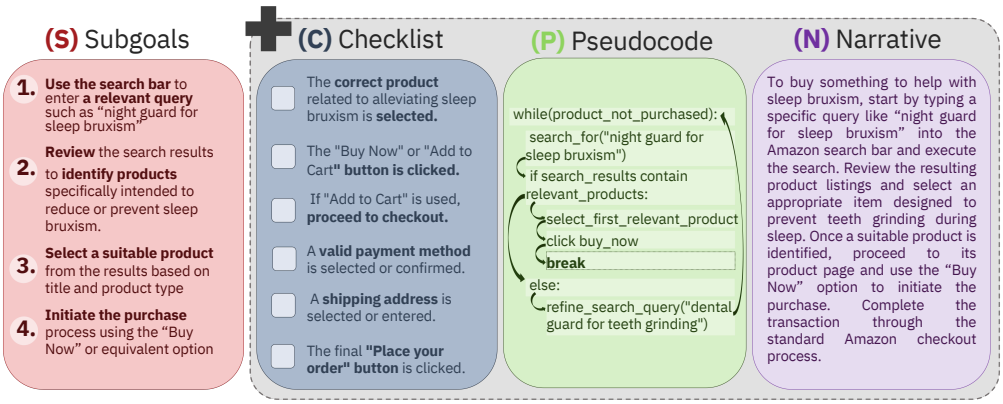


Figure 2: The four representations used in our evaluation of PLANAHEAD. On the left, we show the standard plan of sequential subgoals. In the box on the right, we show three new planning representations. All strategies were generated by GPT-4o given the task *"Buy something to alleviate sleep bruxism"*.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

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.

3.4 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.

3.5 Evaluation Metrics

Web-agent evaluations typically rely on *Success Rate (SR)*, a binary measure of whether a task is completed and their respective *Standard Error (SE)* (Yehudai et al., 2025). SR is averaged over tasks, but does not account for multiple stochastic runs which can obscure meaningful differences, as SR treats every trial across all tasks as independent. For instance, if 3 tasks are each run 5 times and only 2 runs of the first task succeed, SR would be computed as 13.3% $\frac{2}{3\times5}$). Given the stochastic nature of LLM-based agents, metrics that incorporate multiple runs are essential. To account for the non-deterministic nature of LLMs, therefore we propose two novel complementary metrics:

Achievement Rate (AR) extends SR to the multi-run setting by measuring task-level achievability in a set of T tasks. A task is considered achieved if at least one of N runs succeeds:

$$\text{AR} = \frac{1}{T}\sum_{t=1}^{T}\left[\bigvee_{i=1}^{N}\text{Reward}(t,i)\right]\times 100$$

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. $A = \{\, t \mid \exists i \in \{1,\ldots,N\} : \text{Reward}(t,i) = 1 \,\}$), STC measures performance consistency across runs:

$$\text{STC} = \frac{1}{|A|\cdot N}\sum_{t\in A}\sum_{i=1}^{N}\text{Reward}(t,i)\times 100$$

Taking the same example above, in contrast, $AR = \frac{1}{3} = 33.3\%$ since one of the two tasks is "achievable" with a $STC = \frac{1}{1*5}2 = 40\%$. AR and STC together capture more directly both achievability and robustness.

4 Results and Analysis

We evaluate planning strategies on the 158 *Hard* tasks. For each task and strategy, we run $N = 5$

Configuration		Dynamic		Static							
Planner	Executor	Sequential		Sequential		Checklist		Pseudocode		Narrative	
		AR	STC	AR	STC	AR	STC	AR	STC	AR	STC
4.1-mini	4.1-mini	5.1	42	5.1	42	4.4	35	4.4	34	**5.1**	**75**
Qwen	4.1-mini	-	-	5.4	31	4.2	40	5.4	31	**6.6**	**60**
Gemini	4.1 mini	-	-	7.0	45	3.2	36	4.4	51	5.7	53
Qwen	Qwen	5.1	55	2.5	85	**5.1**	**75**	3.2	80	3.8	63
4.1-mini	Qwen	-	-	5.1	35	**7.5**	**30**	5.0	45	5.1	45
Gemini	Qwen	-	-	3.2	32	**5.7**	**40**	3.8	40	4.4	40
Gemini	Gemini	9.5	39	5.7	53	7.6	58	8.2	43	**7.6**	**73**
4.1-mini	Gemini	-	-	**10.7**	**47**	8.9	44	7.0	53	9.5	43
Qwen	Gemini	-	-	**8.9**	**50**	7.0	58	4.4	77	**8.9**	**50**

Table 1: Comparison of our novel metric Achievement Rate (AR) and Solved-Task Consistency (STC) across planning strategies on the *Hard* set. The best result per configuration of planner executor are in bold and the best configuation overall is highlighted. A complementary sensitivity study can be found in Appendix B.

independent trials to observe execution variability and distinguish tasks that are occasionally solvable from those that are consistently solvable. Experiments use 3 multimodal LLM backbones GPT-4.1-mini (4.1 mini), Qwen-2.5-VL-72B (Qwen), and Gemini 2.5 Flash (Gemini) evaluated as both planner and executor, as well as all planner-executor combinations. Table 1 shows the performance measured using AR and STC. We also include a dynamic planning baseline (see Section 3) to compare static and dynamic planning under identical conditions. Key findings are summarized below:

LLMs Exhibit Distinct Preferences for Plan Representations. A central finding is that planning effectiveness varies substantially across LLMs (see Table 1), independent of the planner model used. GPT-4.1-mini consistently achieves its strongest performance with narrative-style plans. 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, which provide explicit and structured guidance. Gemini performs competitively across multiple representations and notably benefits from non-standard formats. Strikingly, pseudocode, despite not being designed for human readability yields one of the highest ARs observed when Gemini is used as both planner and executor. Overall, these results indicate that execution performance is strongly influenced by the plans representation.

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. Notably, GPT-4.1-mini as planner combined with Gemini as executor achieves the strongest overall performance in our evaluation. This pattern indicates that some models, such as GPT-4.1-mini, are more effective at plan generation, whereas others, like Gemini or Qwen, are 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 Often Outperforms Dynamic Single-Agent Planning. Even without the possibility of regenerating or updating plans during execution static plans are able to outperform 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 two novel metrics. Our experiments show that planning effectiveness depends not only on the models chosen but also subgoal quality and its representational form.

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 ($N = 5$). 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, Namyoung 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*.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, 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.

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

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*.

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 Hard Set Generation

To construct the *Hard* subset (Section 3), we evaluated four backbone models over $N = 5$ independent runs on all 381 WebArena test tasks. Table 2 reports the resulting Success Rates (SR). The two strongest backbones: GPT-4.1 and Qwen-2.5-VL achieve high success rate which motivated selecting these models for the main experiments.

We additionally examine how tasks in each difficulty tier distribute across WebArena domains. Figure 3 shows that while the *Hard* subset contains many multi-site tasks (as expected), a substantial portion also comes from the Shopping and GitLab environments. This aligns with prior observations of issues in the UI and evaluation methods (Thakkar et al., 2025).

To assess the robustness of our difficulty labels, we conduct a sensitivity analysis by recomputing the Hard set using fewer runs (from $N = 1$ to $N = 5$). Table 3 reports the overlap between these alternative partitions and our original $N = 5$ categories. Most tasks remain consistently labeled as *Hard* even with a single run, though approximately 16.5% transition from "unsolved" (5/5 failures) to "achieved at least once". This highlights both the intrinsic variability of LLM agents and the value of multiple runs for reliable difficulty grading.

Model	Min	Max	Average
4o-mini	16.3	20.7	17.9
4.1 mini	24.1	33.9	29.0
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 four single-agent models.

# Runs	Number of *Hard* tasks	Overlap with Final Set
1	188	84.0
2	178	88.8
3	168	94.0
4	165	95.7
5	158	100.0

Table 3: Overlap of tasks labeled as hard with final *Hard* set based on the number of runs.

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."

Config	Sequential	Checklist	Pseudocode	Narrative
4.1 mini	3.59–6.59	1.20–4.79	2.40–4.79	5.39–6.59
Qwen 2.5	2.40–2.99	2.40–4.79	3.59–3.59	2.99–4.19
Gemini	3.80-5.70	5.06-7.59	4.43-8.23	6.33-7.59
4.1 mini - Qwen	2.99–5.99	4.79–8.38	2.40–5.39	3.59–5.39
Qwen - 4.1 mini	2.99-5.99	1.80-4.19	1.80-5.3	4.19-6.59
Qwen - Gemini	5.70-8.86	5.70-6.96	3.80-4.43	5.70-8.86
Gemini - Qwen	1.27-3.16	3.16-5.70	1.90-3.80	2.52-4.43
Gemini - 4.1 mini	4.43-6.96	1.27-3.16	2.53-4.43	3.16-5.70
4.1 mini - Gemini	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 Strategy Promtps

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

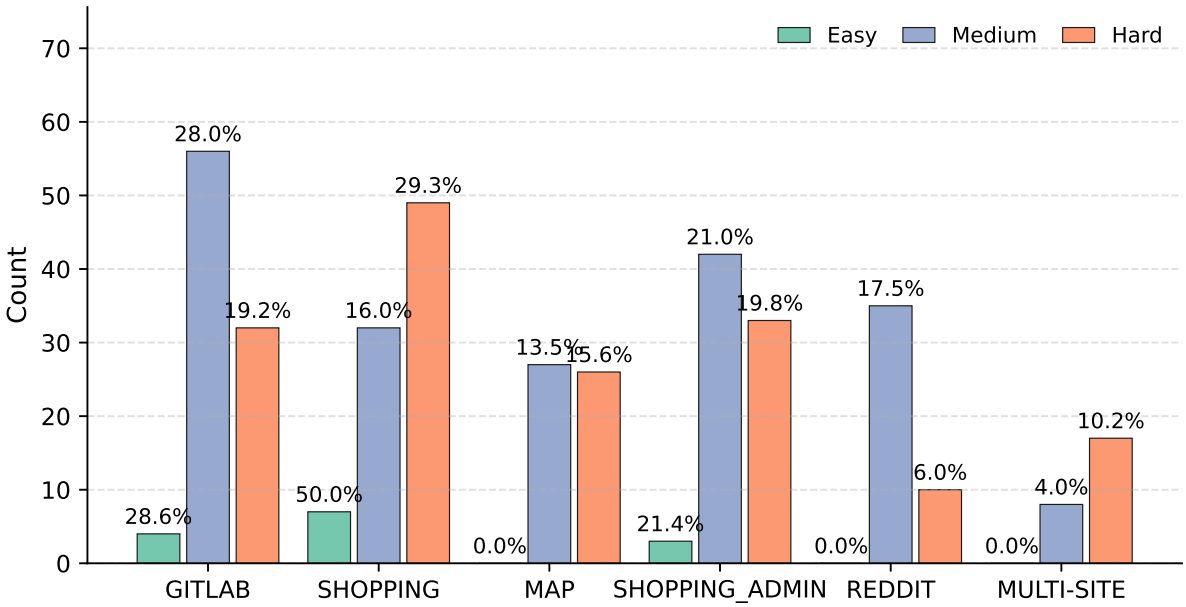


Figure 3: Distribution of *Easy*, *Medium*, and *Hard* tasks across WebArena domains. Percentages indicate each domain's share within a difficulty tier.

Config	Sequential	Checklist	Pseudocode	Narrative
4.1 mini	40.00–72.00	32.50–84.00	34.94–70.00	70.91–88.89
Qwen	68.00–96.00	30.00–75.00	70.00–93.33	57.14–88.00
Gemini	53.33-80.00	58.33-84.44	43.08-77.14	68.33-92.73
4.1 mini - Qwen	34.29–66.67	32.31–67.50	35.56–80.00	40.00–70.00
Qwen - 4.1 mini	28.00-62.86	40.00-76.00	30.00-69.00	54.55-87.50
Qwen - Gemini	49.18-74.00	49.18-74.00	68.57-94.29	50.00-76.00
Gemini - Qwen	26.67-70.00	37.78-75.00	36.67-76.00	40.00-73.33
Gemini - 4.1 mini	45.45-74.29	36.00-70.00	45.71-80.00	47.50-76.03
4.1 mini - Gemini	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

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 5.

Strategy 2, checklist shown in Figure 6.

Strategy 3, pseudocode shown in Figure 7.

Strategy 4, narrative shown in Figure 8.

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 4a.

At the next step the agent correctly calculates the 10% and reduces the price (see Figure 4b).

However, instead of saving the reduced priced, it proceeds to reduce the price again, see Figure 4c and again, see Figure 4d 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:

Configuration		Dynamic		Static							
Planner	**Executor**	**Sequential Plan**		**Sequential**		**Checklist**		**Pseudocode**		**Narrative**	
		SR	**SE**	**SR**	**SE**	**SR**	**SE**	**SR**	**SE**	**SR**	**SE**
4.1 mini	**4.1 mini**	2.0	0.5	2.6	0.6	1.7	0.4	1.9	0.5	**4.8**	**0.7**
Qwen-2.5-VL-72B	**4.1 mini**	-	-	1.8	0.5	1.7	0.4	1.7	0.4	**4.0**	**0.7**
Gemini 2.5 Flash	**4.1 mini**	-	-	**3.2**	**0.6**	1.1	0.4	2.3	0.5	2.9	0.6
Qwen-2.5-VL-72B	**Qwen-2.5-VL-72B**	2.5	0.6	2.4	0.5	1.7	0.4	**2.9**	**0.6**	2.6	0.6
4.1 mini	**Qwen-2.5-VL-72B**	-	-	2.3	0.5	**3.0**	**0.6**	2.3	0.5	2.4	0.5
Gemini 2.5 Flash	**Qwen-2.5-VL-72B**	-	-	1.0	0.4	**2.3**	**0.5**	1.5	0.4	1.8	0.5
Gemini 2.5 Flash	**Gemini 2.5 Flash**	4.4	0.8	3.0	0.6	4.4	0.7	3.5	0.7	**5.6**	**0.8**
4.1 mini	**Gemini 2.5 Flash**	-	-	**5.1**	**0.8**	3.9	0.7	3.6	0.7	4.1	0.7
Qwen-2.5-VL-72B	**Gemini 2.5 Flash**	-	-	**4.4**	**0.7**	4.1	0.7	3.4	0.6	**4.4**	**0.7**

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

```
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.

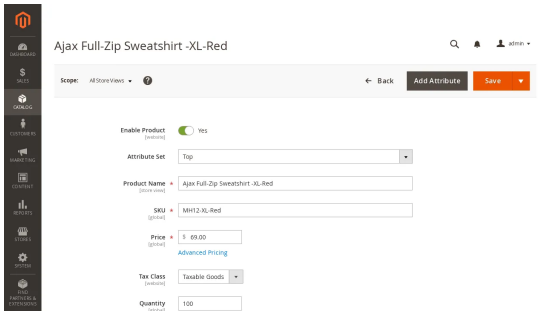
E Additional Experiments and Results

E.1 Easy Set Runs

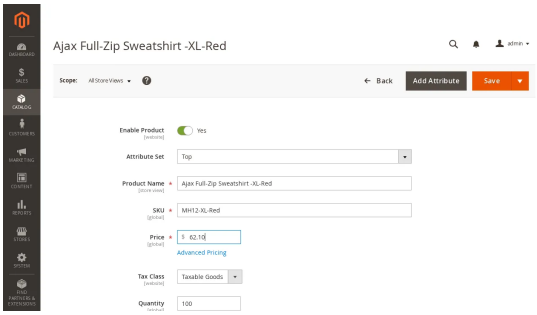
As a sanity check, we evaluate the *Easy* subset to confirm that planning does not degrade performance on tasks that are already solvable. Experiments on the *Easy* subset (see Table 7) revealed that low-level plans can limit exploration and generalization. Executors sometimes failed when constrained 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.

Planner	Executor	Sequential		Checklist		Pseudocode		Narrative	
		AR	STC	AR	STC	AR	STC	AR	STC
4.1-mini	4.1-mini	100	100	100	100	100	100	100	100
Qwen	Qwen	100	100	100	100	100	96	85.7	98
Gemini	Gemini	100	100	100	100	100	100	100	100

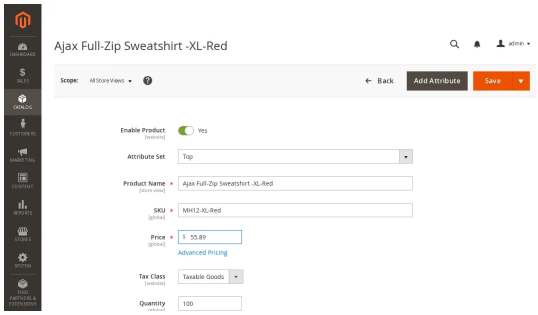
Table 7: AR and STC of different strategies on the *Easy* set by model.



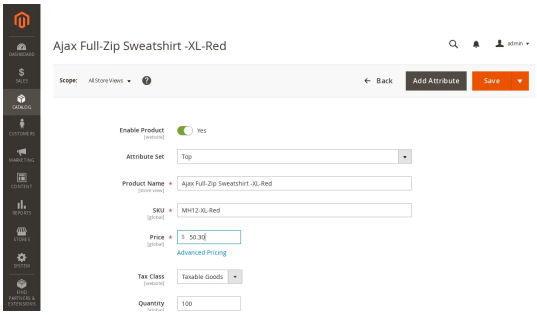
(a) Step 0



(b) Step 1



(c) Step 2



(d) Step 3

Figure 4: Examples of single-agent planner-executor behavior across different WebArena tasks. (a-c) show task webarena.459 where the agent incorrectly keeps reducing the price. (d) shows task webarena.54 where the agent correctly plans a route between universities.

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 5: 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 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 a 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 6: 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 7: 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 8: Prompt for narrative plans.