AGENTOCCAM: A SIMPLE YET STRONG BASELINE FOR LLM-BASED WEB AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Autonomy via agents based on large language models (LLMs) that can carry out personalized yet standardized tasks presents a significant opportunity to drive human efficiency. There is an emerging need and interest in automating web tasks (e.g., booking a hotel for a given date within a budget). Being a practical use case itself, the web agent also serves as an important proof-of-concept example for various agent grounding scenarios, with its success promising advancements in many future applications. Meanwhile, much prior research focuses on handcrafting their web agent strategies (e.g. agent's prompting templates, reflective workflow, role-play and multi-agent systems, search or sampling methods, etc.) and the corresponding in-context examples. However, these custom strategies often struggle with generalizability across all potential real-world applications. On the other hand, there has been limited study on the misalignment between a web agent's observation and action representation, and the data on which the agent's underlying LLM has been pre-trained. This is especially notable when LLMs are primarily trained for language completion rather than tasks involving embodied navigation actions and symbolic web elements. In our study, we enhance an LLM-based web agent by simply refining its observation and action space, aligning these more closely with the LLM's capabilities. This approach enables our base agent to significantly outperform previous methods on a wide variety of web tasks. Specifically, on WebArena, a benchmark featuring general-purpose web interaction tasks, our agent AGENTOCCAM surpasses the previous state-of-the-art and concurrent work by 9.8 (+29.4%) and 5.9 (+15.8%) absolute points respectively, and boosts the success rate by 26.6 points (+161%) over similar plain web agents with its observation and action space alignment. AGENTOCCAM's simple design highlights the LLMs' impressive zero-shot performance in web tasks, and underlines the critical role of carefully tuning observation and action spaces for LLM-based agents.

1 INTRODUCTION

AI agents leveraging large language models (LLMs) show great potential in automating repetitive and programmatic tasks and thereby alleviating human workloads (Gao et al., 2024; Xi et al., 2023; Yang et al., 2024). LLMs showcase remarkable capabilities in perception, reasoning and planning primarily due to their pre-training and post-training. However, their effectiveness is significantly constrained when task-specific observation and action representations diverge from the parametric knowledge encoded during training of LLMs. For instance, in web-based tasks, these agents perform notably below human levels (Zhou et al., 2023b; Koh et al., 2024a).

To improve web task performance by LLM-based agents, recent work focuses on designing better agent policies with either handcrafted prompting templates (Sodhi et al., 2024) or hard-coded auto-prompting strategies (Fu et al., 2024; Wang et al., 2024). While those pre-defined strategies can be effective for certain tasks, they struggle to generalize to diverse websites and varying skill requirements. Another emerging trend is to adopt sampling or search algorithms for a dynamic exploration of web navigation actions, which reduces dependence on pre-defined strategies but increases the cost on LLM inferences. (Koh et al., 2024b; Zhang et al., 2024; Pan et al., 2024).

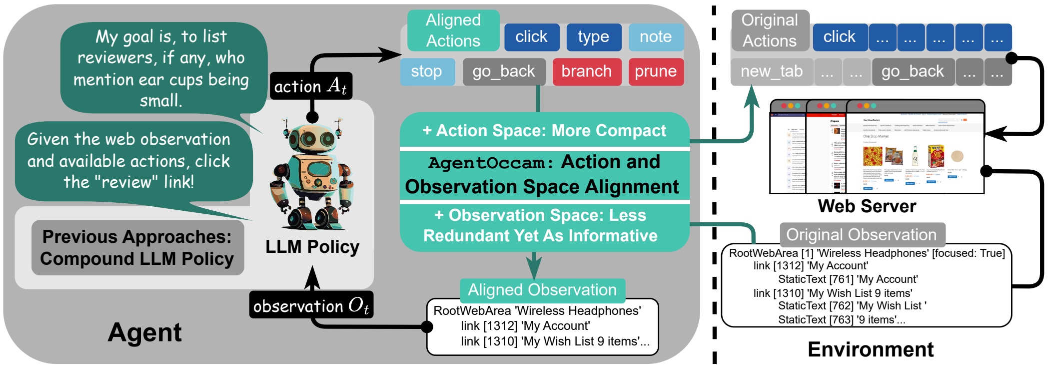


Figure 1: Overview of AGENTOCCAM. Unlike previous work that works intensively on designing compound LLM policies, we enhance the web agent simply by aligning the web interaction action and observation space with the functioning LLM's acquired knowledge and skills during its training.

In this work, we aim to enhance an LLM-based web agent's proficiency by optimizing the text-based task understanding and reasoning of existing LLMs, rather than refining the agent strategies. Automating web tasks is challenging, as the agent needs to *i)* accurately extract information from web pages with varying formats and encoded scripts, and *ii)* issue appropriate embodied actions, selecting from those defined merely on web (e.g. scrolling, clicking, or hovering over buttons). These web observation and action spaces are less common in both, the pre- and post-training data of LLMs, preventing the LLMs from fully realizing their potential in accomplishing general-purpose web tasks. Therefore, we study how to properly tune the observation and actions for LLM-based web agents, to align them with the functioning LLMs capacities learned during pre-training.

As shown in Figure 1, our proposed method comprises of three components: *i)* We reduce non-essential actions to minimize the agent's embodiment and trivial interaction needs; *ii)* We refine the observation by eliminating redundant and irrelevant web elements, and restructuring web content blocks for more succinct yet as informative representations; *iii)* We introduce two planning actions (`branch` and `prune`), which enables the agent to self-organize navigation workflow with a planning tree, and use the same structure to filter the previous traces for history replay. We implement these components by generic rules that applies to all types of markup-language-formatted web pages, without leveraging task-related information on the test benchmark.

By combining the three techniques mentioned above, our proposed agent AGENTOCCAM performs substantially better on web tasks across websites in the WebArena environments (Zhou et al., 2023b). AGENTOCCAM outperforms the previous state-of-the-art approach by 9.8 absolute points (+29.4%) and surpasses concurrent work by 5.9 absolute points (+15.8%). Notably, unlike most prior work, we do not use any in-context examples, additional online search or sampling, nor specialized prompting templates or agent roles to play well. In contrast, AGENTOCCAM delivers such strong performance with an unexpectedly simple approach: letting the LLM issue actions within the processed and augmented observation and action spaces. Compared with a similar plain web agent without these proposed observation and action space changes, AGENTOCCAM increases the success rate by 26.6 absolute points (+161%).

In summary, the primary contribution of this work are as follows. First, we develop a new state-of-the-art agent, AGENTOCCAM, for web tasks. On the WebArena benchmark consisting of 812 tasks across five diverse websites (e.g., shopping, searching on a forum), AGENTOCCAM outperforms previous and concurrent work significantly. Second, we shed light on the strong zero-shot performance of LLMs on web tasks with our simple agentic workflow, in sharp contrast to many more complex compound agent policies. Last, our work on aligning the observation and action spaces is orthogonal to agentic strategies and can be combined with future advances in that aspect.

[1]AWM supports two scenarios: in offline scenarios it directly leverage an offline dataset, and in online scenarios it relies on a domain-specific evaluator from Pan et al. (2024) which requires offline data to train.

Table 1: Comparison of essential components for different web agents.

Essential Components	Task-specific Strategies	Additional Module	In-context Examples	Offline Data	Online Search
AutoGuide (Fu et al., 2024)	NO	YES	YES	YES	NO
SteP (Sodhi et al., 2024)	YES	YES	YES	NO	NO
AutoRefine (Pan et al., 2024)	NO	YES	YES	YES	YES
LM-Tree Search (Koh et al., 2024b)	NO	YES	YES	YES	YES
AWM (Wang et al., 2024)	NO	YES	YES	YES[1]	NO
WebPilot (Zhang et al., 2024)	NO	YES	YES	NO	YES
AGENTOCCAM	NO	NO	NO	NO	NO

2 Related Work

LLM-based Web Agent Advances in large language and multi-modal foundation models have significantly boosted the development of autonomous agents to solve web tasks. Techniques translating LLMs to powerful decision-making agents (Yao et al., 2022b; Shinn et al., 2024) have led to progress in web agents, and have inspired many techniques that design inference time agent strategies. Many prior approaches improve the agent system by designing modular systems with specialized LLMs or roles, aiming to break down complex tasks (Sun et al., 2024; Prasad et al., 2024). Other works leverage LLMs to extract common patterns from examples or past experience (Zheng et al., 2023; Fu et al., 2024; Wang et al., 2024). However, this line of work often relies on pre-defined control hierarchy, prompt templates or examples to act accurately in the test environments. For example, SteP (Sodhi et al., 2024) utilizes a stack-based approach for dynamic multi-level control in the web tasks but relies on task-specific atomic policies with environment-related information hard-coded in prompt template. Another line of work focuses on improving web agents' performance by leveraging more online examples from the environments. Many of them (Zhou et al., 2023a; Zhang et al., 2024; Putta et al., 2024) adapt Monte Carlo Tree Search (MCTS) methods, expanding intermediate states (tree nodes) in one task repeatedly by multiple trials over that task. Among them, WebPilot (Zhang et al., 2024) also adds a global optimization layer for high-level planning. Koh et al. (2024b) use a trained value function to guide search and to back-trace on the task execution tree. Auto Eval and Refine (Pan et al., 2024) trains a separate evaluator, and improves the task execution using reflective thinking (Shinn et al., 2024) on past trials in the same task. However, sampling or resetting multiple times in the same task, not only increases the inference cost significantly, but also limits its applicability when failed task is not revocable. As a comparison, we highlight the simplicity of our method and its difference with related agent approaches in Table 1.

Fine-tuned or Trained Models for Web Tasks Fine-tuning language or multimodal models for web tasks is another effective approach to enhance decision-making capabilities on the web tasks (Yin et al., 2024; Hong et al., 2024; Lai et al., 2024; Putta et al., 2024). Although fine-tuning promises more adaptivity and optimization space, the size of task-specific fine-tuned models is often not comparable with the most powerful closed-source models. As for training models to follow natural language command on the computer or the web, there is also some early research before LLMs emerged, using semantic parsing (Artzi & Zettlemoyer, 2013), reinforcement learning (Branavan et al., 2009) and imitation learning (Liu et al., 2018; Humphreys et al., 2022). However, those fine-tuned agents, limited by the base model's capacities, fail to match those constructed with LLMs.

Simulated Web Agent Environments Web agent development has been supported by increasingly complex web simulators for training and evaluation. These range from basic platforms like MiniWoB (Shi et al., 2017) and its extension MiniWoB++ (Liu et al., 2018), to more sophisticated environments such as WebShop (Yao et al., 2022a), WebArena (Zhou et al., 2023b), and Visual-WebArena (Koh et al., 2024a). These simulators progressively incorporate real-world complexities, from simple form-filling to tasks across multiple full-featured websites. In this work, we focus only on the text modality, and use WebArena to evaluate our method's task success and generalizability as it contains different types of websites and task-intents in a single suite.

3 Problem Formulation

We formalize the web interaction process by a Partially Observable Markov Decision Process (POMDP, Littman (2009); Spaan (2012)): $\langle \mathcal{O}, \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma \rangle$. In POMDPs, an observation $o \in \mathcal{O}$ consist of information that the agent receives from the web environment, e.g. HTMLs, as well as any instructions and prompts. In this work, we consider the text modality only. A state $s \in \mathcal{S}$ denotes the whole underlying (unobserved) state of the agent and the environment such that the state transition is Markovian. An action $a \in \mathcal{A}$ is either a command recognized by the web environment, or any other unrecognized token sequence that will lead to staying in the current state. P denotes a deterministic state transition function that records the change in the webpage state given the current state and agent action. R is the reward function that decides the success or failure of the agent's sequence of actions. In the WebArena environment used in our work, the reward is only assigned at the end of an agent-web interaction episode. p_0 denotes the initial state distribution which is uniform over 812 tasks in WebArena and discounting factor γ is set to 1.

To solve POMDP, a common goal is to find a decision policy $\pi(a_t|h_t)$ maximizing the expected cumulative reward, where h_t denotes the observation history $\{o_0, o_1, ..., o_t\}$. In LLM-based web agent design, that is translated to designing a policy $\pi(a_t|h_t)$ with the help of one or more base LLM policy π_{LLM} and a set of algorithmic modules. In this work, we work on a special class of policies that can be expressed as: $\pi(g(a_t)|h_t) = \pi_{\text{LLM}}(a_t|f(h_t))$, where f and g are rule-based functions that process the observation (including action instructions) and actions for the LLM policy. We name it the observation and action space alignment problem. Notice that under such problem setting, all of our changes apply only to the observations and the actions. We emphasize not all agent strategies in previous approaches can be represented in this way. For example: search-based algorithms require a control program on the top to select actions and trigger back-tracing; methods with evaluators, reflective thinking or memory modules also necessitate a managing center to alternate between the main LLM and these helper segments or role-playing LLMs. In contrast, we aim to answer the following question in our work: **Can we build a strong web agent with the base LLM policy π_{LLM} by optimizing only the observation and action mapping f and g?**

4 Method

Rather than introducing any new modules or hierarchical structures on top of the base LLM, our method focuses on a simple web agent workflow that inputs the web observations to a general-purpose LLM-API and uses the LLM outputs as actions directly. In this section, we describe the process of aligning web navigation tasks, which necessitates embodiment knowledge, with the predominantly static and text-centric nature of LLM training. Section 4.1 discusses our strategies (summarized in Figure 2) for refining the action space to be more compact and reducing the need for the agent's embodiment capabilities. Section 4.2 outlines our methods (summarized in Figure 4) for condensing web content descriptions to be both brief and informative, and identifying key web elements and relevant steps for retention to organize the agent's memory in a pertinent manner.

4.1 Action Space Alignment

A web agent's action space defines the valid commands it can use to interact with web environment. Based on our observation of common failure modes in web agents, there are two key problems that need to be solve by editing the action space: *i)* removing irrelevant actions that LLMs struggle to understand and frequently misuse, and *ii)* improve the memorization and planning ability when the task execution requires navigating multiple potential paths to successfully execute. We propose that the first can be corrected simply by removing and combining actions. The second one was often solved in the previous work by handcrafted rules or strategies, making these approaches hard to generalize. In this work, we address the second problem by adding actions to allow the LLM to autonomously generate plans and manage the task workflow. Both of these proposed solutions are explained in details below and in Figure 2.

Simplifying the Action Space. First, we eliminate actions that can be replicated using similar actions or replace multiple actions by one action with the same expressiveness (illustrated in Figure 2 step 1). Specifically, we remove the `noop` action, signifying "no operation", as a distraction to the agent in most cases. Similarly, tab operations, which manage the focusing, opening, or closing of

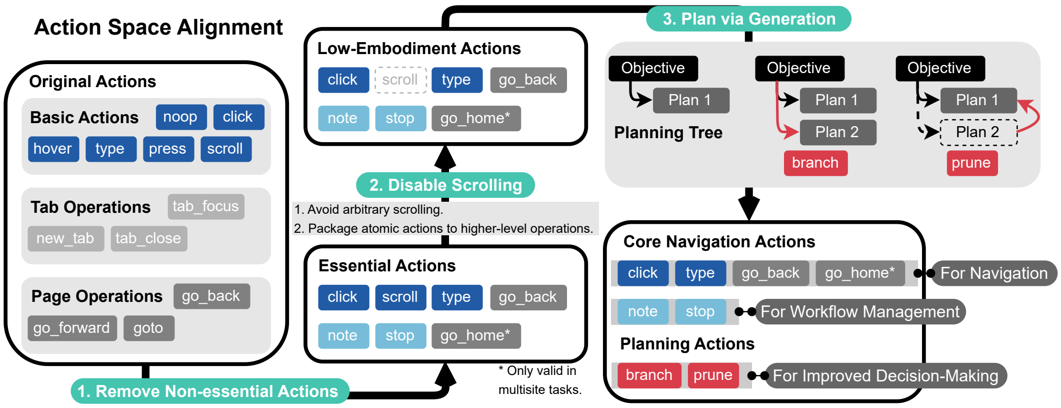


Figure 2: In aligning the action space with LLM pre-training, we only retain high-utility actions and lessen the demand for advanced embodiment skills (steps 1 and 2). Additionally, we incorporate planning steps, allowing the agent to *autonomously* manage task breakdown and execution (step 3).

tabs are removed because they are only needed in a limited cases of multi-site tasks requiring two tabs. Furthermore, we limit page navigation actions like `go_forward` and `goto`, as their utility is greatly constrained by the agent's poor memory of the relationship between a page's URL and its content. By eliminating these less effective actions, our goal is to minimize distractions and boost the agent's concentration on more meaningful operations. In addition, we introduce the `note` action, allowing the agent to record key observations for subsequent conclusions, and the `stop` action, enabling the agent to autonomously conclude the trajectory with answers. We also add a `go_home` command for multi-site tasks, enabling the agent to navigate directly to the homepage where all available sites are listed.

Second, we eliminate actions that heavily require embodiment knowledge and simplify low-level actions into more abstract operations as shown in Figure 2 step 2. In particular, we reduce commands that LLM-based agents struggle with unless provided with detailed context-specific guidance, like `hover` or `press` (the latter is for pressing key combinations, often shortcuts). To properly use these actions requires LLMs to have embodied thinking of the current scenario, especially regarding the mouse position, which it has not acquired during the training. Additionally, we remove the `scroll` action, opting instead to load the full page content as the web state. This change is in response to our observation that agents tend to engage in aimless and repetitive scrolling when an essential link is not visible at the top of the page, wasting steps without making progress. Furthermore, we streamline the agent's interaction with drop-down menus; instead of selecting the menu and then an option, a single `click` command with the ID of desired option now suffices. The list of all actions in original and reduced action space are shown in Table 3, together with the frequency they are taken in different agents.

Planning via Generation. Web tasks often requires solution that requires navigating multiple paths (e.g. extracting information from one page and submitting it to another page, like the task of creating a refund request on the contact us page for a broken product (task template 154), which requires parsing the order ID and refund amount from the order pages). We propose adding of two actions (`branch` and `prune`) to generate plans in a tree structure and save them for future observations. As Figure 2 step 3 shows, the LLM-generated plans starts with a root node being the objective of the task. The `branch` action will generate new subplans under the current node, decomposing high-level objectives into smaller, more manageable subgoals. Conversely, the `prune` action allows the agent to give up the current sub-plan (often after repetitive failed attempts) and seek for alternatives. Together with the branch and the prune actions, the LLM can edit the planning tree autonomously. Note that these two `planning` actions are of no difference from the native `navigation` actions in the web environment (e.g. click, type) and the LLM is free to choose when to take these actions to update the plan. The generated plan provides a context for future action generation and enhances the consistency of actions in one trajectory. This approach leverages the intrinsic planning ability in LLM itself. We argue that this increases the generalization performance as this design has minimum dependency on prior knowledge.

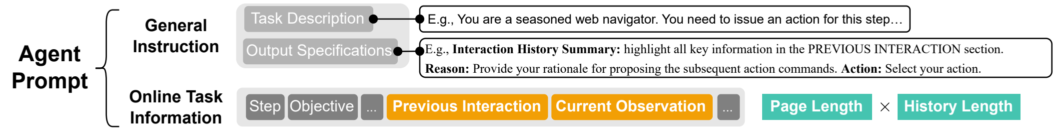


Figure 3: The components of our web navigation agent's prompt. It includes a general instruction outlining the task and the desired output, as well as online task information providing the current goal, the agent's past interactions, and the latest observations. Notably, the sections on previous interactions and current observation use the most tokens. These can be attributed to two main factors: the length of the pages and the extent of history span, with current observation primarily depending on page length and past interactions on both page length and history span.

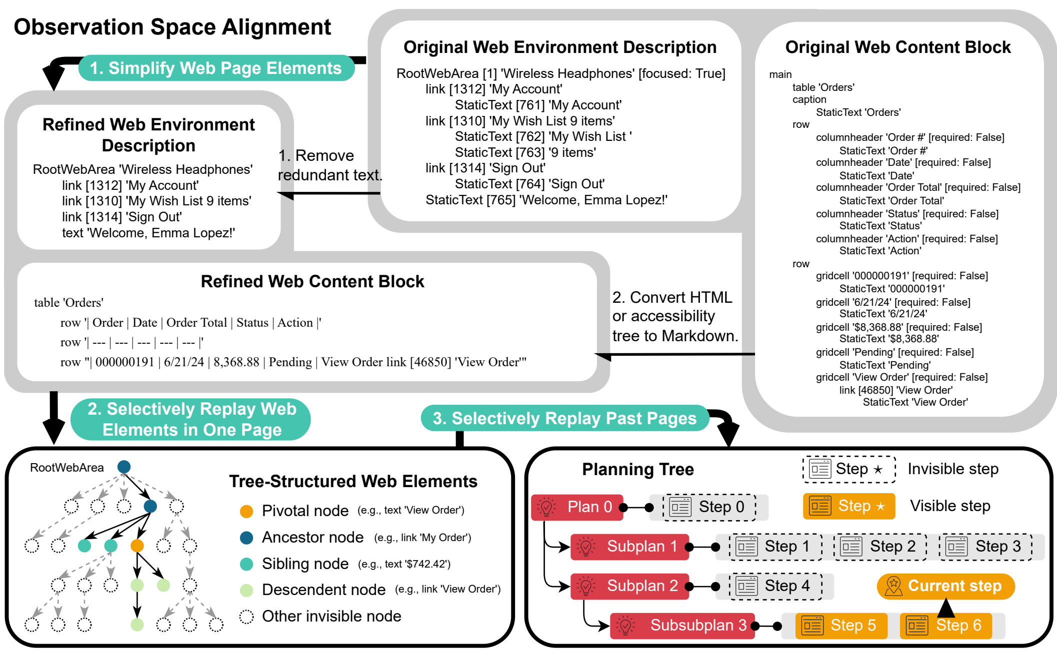


Figure 4: To align the task's observation space with the base model's pre-training, we condense a single-page length by removing unnecessary texts that repetitively describe the web page's functionality and layout (step 1), and by identifying page elements relevant to the task for the agent to remember (step 2). Additionally, we optimize the agent workflow memory through a stacked planning tree, viewing each new plan as a separate goal and excluding past steps' information dedicated to previous plans to enhance memory conciseness (step 3).

4.2 Observation Space Alignment

The observation space of web agents consists of task objectives, instructions, previous interaction, the current web text descriptions or screenshots (see Figure 3 and Appendix C for our agent). Among them, previous interactions and current web content consumes the most number of tokens, which scales with the length of a single page and the length of history. This often results in a long context window, which not only increases the LLM inference cost but also poses challenges for LLM to extract related information accurately. Therefore, our primary focus in refining the observation is to target these two aspects. Additionally, the alignment of observations is outlined in Figure 4.

Simplifying Web Page Observations. The content on web pages are represented in HTML or accessibility tree format in most text-only web agents. These formats are designed towards front-end loading and rendering, containing numerous formatting tokens making them lengthy and repetitive, as illustrated in Figure 4 Step 1. Our goal is to optimize the representation to make it more readable to LLMs in one single page. Specifically, we merge function-descriptive web elements (e.g., `StaticText [761] 'My Account'`) with interactive elements that share the same label (e.g., `link [1312] 'My Account'`). We then convert table and list blocks to Markdown, eliminating repetitive structural tokens (e.g., `columnheader`, `gridcell`). Consequently, we achieve a more concise representation while keeping the same information.

Replaying Observation History Selectively. Taking observation history as input is important for decision-making agents to act consistently for tasks needing long horizons, given the prior that the observation state only contains partial information about the environment's state. For web tasks, it is also important to include both observation and action history as some key information may not be displayed on the current page. However, the observation history will also significantly scale up the context length and increase reasoning difficulty as well as inference cost. We address this issue by only selecting the most important and related information on previous web page, according to two rules based on the "pivotal" nodes (defined later) and the planning tree.

First, we observe that only small amount of content on a web page is pertinent to a specific task among several steps and is worth to replay in future steps. For example, in tasks requiring the agent to find all reviews with in three months, it is unnecessary to keep other reviews or some unrelated links like `Contact Us` on the page. Thus we employ a simple rule to identify this small amount of content by leveraging the tree structure of web data (e.g. accessibility tree). To do this, we first instruct the agent to pinpoint the crucial web elements denoted as "pivotal" nodes, every time the agent generates an action. The agent is then programmed to include only the pivotal nodes' ancestor nodes (indicating their global hierarchy and position), sibling nodes (providing immediate context), and descendant nodes (offering detailed characteristics) in the future observations as illustrated in Figure 4 Step 2. This effectively narrows down the volume of data and level of noise passed to future context of LLM inference.

Second, we observe that not all previous steps' observation needs to be noted during the inference of future step. Thus we can leverage the planning tree generated by the agent itself to keep the agent's focus sharp. Specifically, when the agent initiates a `branch` action to develop a new plan, we treat this new plan as a separate goal. Steps taken for earlier plans and their observations will be dismissed in the current plan's observation window, as depicted in Figure 4 step 3. This allows the agent to focus only on information dedicated to the current plan for a sub-task.

5 Experimental resuts and analysis

Environment. We utilize WebArena (Zhou et al., 2023b), an interactive web simulator, as our benchmark. WebArena consists of fully functional websites from four common domains: e-commerce platforms (OneStopShop), social forums for idea and opinion exchange (Reddit), collaborative software development (*e.g.* GitLab), and content management for creation and management of online data (online store management). The platform additionally includes utility tools: a map, a calculator and a scratchpad, and Wikipedia to enable human-like task-solving. The benchmark consists of 812 tasks generated from 241 templates. A template here is a parametric form of a task intent, allowing for multiple instantiations with different values. Each task is accompanied by a specific evaluator/reward function that programmatically checks the correctness of the final information with respect to the desired ground truth information and the alignment of intermediate actions with the overall task objective [2]. We use `GPT-4-turbo-2024-04-09` (Achiam et al., 2023) to build our AgentOccam.

Baselines. We compare AgentOccam with the following prior and concurrent work: 1) WebArena agent: the Chain-of-Thought (CoT) prompted agent included in the WebArena benchmark (Zhou et al., 2023b). 2) SteP (Sodhi et al., 2024): a stack-based approach on top of 14 human-written atomic strategies tailored to solving WebArena. 3) WebPilot (Zhang et al., 2024): a multi-agent, multi-level MCTS based agent that reports state-of-the-art overall performance on WebArena. 4) Agent Workflow Memory (AWM) (Wang et al., 2024): a method automatically summarizing workflow from past experience. SteP has made their code and interaction trajectories public. Hence, we are able to fully replicate the agents from WebArena and SteP with `GPT-4-turbo` in identical web

[2] We identified and corrected errors in the original evaluators, with details discussed in Appendix A. Our approach outperforms the baseline methods with both original or corrected evaluators.

Table 2: Comparison of the success rate (SR) of AGENTOCCAM with baseline agents on WebArena.

Agent	Model	SR (%) (#812)	Shopping (#187)	Shopping Admin (#182)	GitLab (#180)	Map (#109)	Reddit (#106)	Multisite (#48)
WebArena-Replication	GPT-4-Turbo	16.5	16.6	15.9	10.0	22.9	21.7	16.7
SteP-Replication	GPT-4-Turbo	33.3	33.2	32.4	26.7	35.8	52.8	12.5
AWM	GPT-4	35.5	-	-	-	-	-	-
WebPilot	GPT-4o	37.2	-	-	-	-	-	-
AGENTOCCAM	GPT-4-Turbo	**43.1**	**40.6**	**45.6**	**37.8**	**46.8**	**61.3**	**14.6**

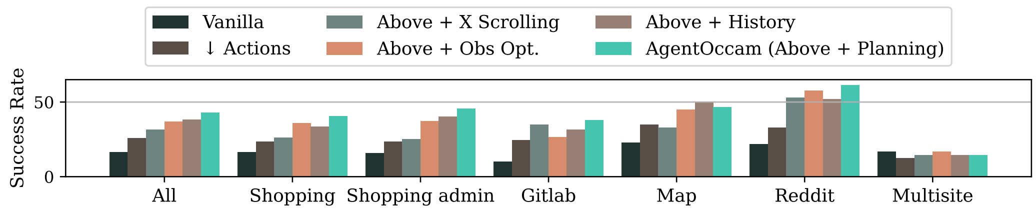


Figure 5: Ablation study of AGENTOCCAM's action and observation space refinement. We incrementally add refinement components and evaluate their marginal performance gains.

environments as our methods, for a fair comparison.[3] WebPilot and AWM, being concurrent works with this paper, do not yet provide source code or resulting trajectories, limiting our analysis of these works to just reporting the aggregated performance numbers included in their technical reports. Our analysis focuses on SteP as it is the most performant method prior to this work.

Question 1: How well does AGENTOCCAM perform? As seen from the results in Table 2, our agent AGENTOCCAM, which optimizes the action and observation space, now sets a new SOTA on the WebArena benchmark. It increases the overall success rate from 37.2% to 43.1%, a **15.8% relative improvement over best results among previous and concurrent work**. We observe that AGENTOCCAM not only accomplishes tasks in the template that is previously unsolvable, like updating personal information on OneStopShop (task template 165), but it also raises the success rate for templates with mixed results previously, such as setting a homepage URL on a GitLab profile (task template 331). This is further illustrated in Figure 6 in the appendix.

Question 2: How much does each observation and action space change contribute to AGENTOCCAM? We evaluate the contribution of each component in AGENTOCCAM described in Section 4 to its overall success by incrementally integrating them into the vanilla agent (WebArena-Replication) and assessing the marginal performance gain shown in Figure 5. The details of each incremental experiment are as follows:

i) **Removal of non-essential actions (↓ `Actions`)**: Narrowing the action space can reduce the level of distraction for LLM policies and significantly improves performance across all tested websites as shown in Figure 5. By removing rarely used actions like `tab_focus`,`go_forward`, `hover` and `press`, the agent spends less steps wandering around and explores more efficiently using actions such as `click` and `type`. Table 3 shows it reduces hundreds of `hover` and `goto` actions while significantly increase the number of `click` and `type`.

ii) **Disabling scrolling (`Above + X Scrolling`)**: We observe that LLM policies tend to use `scroll` up and down often when they do not know what to do (since these action are revertible). Consequently, it significantly delays the task execution and causes looping over in certain tasks. As a result, disabling the scrolling action and passing the entire page to agent proves advantageous, especially for GitLab and Reddit tasks. However, this strategy increases the number of observation tokens, which will be addressed by subsequent refinements.

[3]In our experiments, we note that all agents can occasionally fails due to errors from the WebArena simulator, such as exceeding posting rate limits in Reddit or the login expires. In that case, we restart the experiments.

[4]We remove `stop` in the statistics for the vanilla WebArena agent as this action is excluded in their officially defined action space. However, their agent is allowed by code to generate `stop` to end the trajectory.

Table 3: Action statistics for the ablation study of AGENTOCCAM's components. Each number in the table represents the frequency of an action across all the tasks within the experiment setting. Actions `noop`, `go_forward`, `tab_focus` and `tab_close` are not included since they are not used even once in vanilla agent and removed in our method.

Exp.	click	hover	type	press	scroll	new_tab	go_back	goto	note	stop	go_home	branch	prune
Vanilla	2328	126	1024	7	132	20	71	511	-	-[4]	-	-	-
↓ `Actions`	7119	-	2531	-	370	-	52	-	194	512	36	-	-
`Above + X Scrolling`	7033	-	2390	-	-	-	100	-	219	536	42	-	-
`Above + Obs Opt.`	6890	-	2040	-	-	-	56	-	201	571	23	-	-
`Above + History`	4625	-	1286	-	-	-	94	-	112	801	54	-	-
AGENTOCCAM	4720	-	1159	-	-	-	339	-	197	769	42	34	47

Table 4: Average observation tokens per step across WebArena sites.

Exp.	All	Shopping	Shopping Admin	GitLab	Map	Reddit	Multisite
Vanilla	2210.2	2272.1	2460.2	2199.1	1883.2	2132.4	1751.0
↓ `Actions`	1652.0	1644.7	2133.1	1981.3	912.0	1081.2	1296.8
`Above + X Scrolling`	3376.2	3148.0	5403.7	3364.9	1378.1	2603.6	1975.5
`Above + Obs Opt.`	2891.1	1722.5	4791.7	2560.8	1476.4	3332.3	1619.4
`Above + History`	3051.3	1802.6	5140.2	3153.3	862.1	3156.1	2030.3
AGENTOCCAM	2930.9	1634.2	4920.7	3126.8	1056.0	3697.8	1282.5

iii) **Simplifying web page elements (`Above + Obs Opt.`)**: We remove redundant text and web format as show in Figure 4 Step 1. This results in fewer tokens in the context window, as outlined in Table 4. It helps the agent focus on web elements crucial to task success across all websites and boosts the performance on all task types, except on Gitlab, where this sometimes leads the agent to overlook simpler solutions (task id 394).

iv) **Selective replay of web elements in one page (`Above + History`)**: In this experiment, we follow step 2 shown in Figure 4 to add a subset of elements from previous web pages as history. We observe that it allows the agent to avoid repetitive actions in tasks, significantly decreasing the steps needed for task completion as demonstrated in Table 5. However, this addition slightly hurts performance in tasks with dense single-page content or those requiring navigation across multiple pages, as shopping and Reddit tasks success rate drops by 3.2 and 6.0 points.

v) **Planning via generation and selective replay of past pages (AGENTOCCAM; `Above + Planning`)**: We introduce actions `branch` and `prune` to generate actions and exclude historical steps not in the current sub-plan from the current prompt context. This results in performance gains in tasks across nearly all websites, alongside a reduction in the required observation tokens. The actions `branch` and `prune` are both primarily used in correcting a failed strategy and trying an alternative path. For example, in the task of identifying the nearest national park to Boston (task id 265), the agent employs a `branch` action to adopt an alternative search strategy after a failed search attempt. In a GitLab task (id 563) the agent after multiple failed attempts uses the `Create project` button opts for a `prune` action to explore other methods.

Question 3: Could the power of AGENTOCCAM be combined with other agentic strategies? A natural question to ask next is if we can combine these changes with other common agent strategies or prior work, since the changes in observation and action space are orthogonal and complementary to them. We showcase two example studies to answer this question: one with the SteP method (Sodhi et al., 2024) and another action selection method with LLM-as-a-judge.

The judge method is motivated by our observation of the high variation from the agent's behavior. In some key steps, the agent has certain probability of generating the correct action but often failing to do so, making it hard for the agent to recover from later pages. For instance, when tasked with identifying the most suitable subreddit for posting (task template 6100), the AGENTOCCAM agent tends to hastily choose less relevant subreddits and gets stuck there. To address this, we direct the AGENTOCCAM to generate all possible suitable actions instead of one action at each step. These action candidates are then evaluated by another LLM (`GPT-4-turbo` as well) prompted to be play the role of a judge and select the best action. The prompts for the judge are included in Appendix C.

Table 6 shows that a AGENTOCCAM + SteP agent, enhanced with task strategies, outperforms the standalone SteP method but doesn't match AGENTOCCAM's base performance. Additionally, com-

Table 5: Average number of steps per task across all WebArena sites.

Exp.	All	Shopping	Shopping Admin	GitLab	Map	Reddit	Multisite
Vanilla	6.2	6.2	6.6	5.9	5.7	7.4	4.4
↓ `Actions`	13.3	10.6	14.3	14.8	11.9	15.2	13.7
`Above + X Scrolling`	12.7	9.0	14.0	14.8	12.7	13.0	14.0
`Above + Obs Opt.`	12.0	8.5	13.2	15.4	10.2	12.1	13.2
`Above + History`	8.6	5.6	9.6	10.3	8.3	7.6	12.9
AGENTOCCAM	9.0	6.7	9.2	10.8	8.5	8.6	13.4

Table 6: Success rate (SR) of AGENTOCCAM combined with agent strategies on WebArena.

Agent	Model	SR (%) (#812)	Shopping (#187)	Shopping Admin (#182)	GitLab (#180)	Map (#109)	Reddit (#106)	Multisite (#48)
AGENTOCCAM	GPT-4-Turbo	43.1	40.6	45.6	37.8	46.8	61.3	14.6
AGENTOCCAM + SteP	GPT-4-Turbo	41.1	**46.5**	36.3	36.7	47.7	50.9	**18.8**
AGENTOCCAM + Judge	GPT-4-Turbo	**45.7**	43.3	**46.2**	**38.9**	**52.3**	**67.0**	16.7

bining AGENTOCCAM with a judge role through an action prediction and selection pipeline rectifies some of the base agent's behavioral misconduct.

By analyzing the trajectories of each method, we observe that the task-specific strategy like SteP can help when the strategy fits the task requirement. For example, in the task of "`Draft an email to the shop owner via their contact us function for a coupon as {reason}`" (task template 163), the AGENTOCCAM + SteP and SteP agents excel by prompting the agent explicitly not to click the submit button after drafting, where AGENTOCCAM fails to follow. However, for tasks outside the designed strategies, these hints can mislead the agent, leading to 2 points drop in overall success rate of AGENTOCCAM + SteP compared to AGENTOCCAM only. An example is task 639, where the agent, guided by SteP's instruction "`Under forums, you will see only a subset of subreddits. To get the full list of subreddits, you need to navigate to the Alphabetical option.`", repetitively navigates away from the appropriate subreddit, and generates reasons for its action selection that "`Clicking on the 'Alphabetical' link will help us access a more comprehensive Reddit list.`", demonstrating how hard-coded strategies can distract the agent and hurt generalizability.

The AGENTOCCAM + Judge agent, combining the AGENTOCCAM's generated action list with the second opinion from a LLM judge increases its overall success rate by 2.6%, by completing tasks where it may well fail due to intermediate decision flaws. For example, in choosing the right subreddit for a post (task template 6100), the base AGENTOCCAM might hastily pick from an initial list, whereas the AGENTOCCAM + Judge agent conducts a thorough search using post keywords or explores the entire forum list before drafting the post. This approach minimizes errors due to rushed decisions, increasing the likelihood of successfully completing task series.

6 CONCLUSION

In this paper, we proposed a simple but efficient LLM-based web navigation agent AGENTOCCAM that refines its observation and action space by making them more readable and friendly to the LLMs trained on language. Compared with other agent methods, AGENTOCCAM shows a surprising simplicity in its policy workflow design with no additional modules, LLM calls or in-context example requirements. Despite its simplicity, AGENTOCCAM outperform prior and concurrent work on WebArena by 9.8 (SteP) and 5.9 (WebPilot) absolute points respectively. Our result underlines that it is important to keep the agent architecture simple for its generalizability, unless an additional module is necessary, echoing the principle of Occam's razor. In summary, we hope AGENTOCCAM provides both strong groundwork as well as insights for the future research and development of web agents.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Yoav Artzi and Luke Zettlemoyer. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the association for computational linguistics*, 1:49–62, 2013.

Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pp. 82–90, 2009.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents, 2024. URL https://arxiv.org/abs/2403.08978.

Chen Gao, Xiaochong Lan, Nian Li, Yuan Yuan, Jingtao Ding, Zhilun Zhou, Fengli Xu, and Yong Li. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, 11(1):1–24, 2024.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024.

Peter C Humphreys, David Raposo, Tobias Pohlen, Gregory Thornton, Rachita Chhaparia, Alistair Muldal, Josh Abramson, Petko Georgiev, Adam Santoro, and Timothy Lillicrap. A data-driven approach for learning to control computers. In *International Conference on Machine Learning*, pp. 9466–9482. PMLR, 2022.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.50. URL https://aclanthology.org/2024.acl-long.50.

Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents, 2024b. URL https://arxiv.org/abs/2407.01476.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, et al. Autowebglm: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5295–5306, 2024.

Michael L Littman. A tutorial on partially observable markov decision processes. *Journal of Mathematical Psychology*, 53(3):119–125, 2009.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*, 2018.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. In *First Conference on Language Modeling*, 2024.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. ADaPT: As-needed decomposition and planning with language models. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 4226–4252, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-naacl.264. URL https://aclanthology.org/2024.findings-naacl.264.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*, 2024.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, pp. 3135–3144. PMLR, 2017.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. Step: Stacked llm policies for web actions. In *First Conference on Language Modeling*, 2024.

Matthijs TJ Spaan. Partially observable markov decision processes. In *Reinforcement learning: State-of-the-art*, pp. 387–414. Springer, 2012.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024. URL https://arxiv.org/abs/2409.07429.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*, 2023.

Ke Yang, Jiateng Liu, John Wu, Chaoqi Yang, Yi R Fung, Sha Li, Zixuan Huang, Xu Cao, Xingyao Wang, Yiquan Wang, et al. If llm is the wizard, then code is the wand: A survey on how code empowers large language models to serve as intelligent agents. *arXiv preprint arXiv:2401.00812*, 2024.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022b.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent lumos: Unified and modular training for open-source language agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12380–12403, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.670. URL https://aclanthology.org/2024.acl-long.670.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration, 2024. URL https://arxiv.org/abs/2408.15978.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In *The Twelfth International Conference on Learning Representations*, 2023.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023a.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023b.

A EVALUATOR RECTIFICATIONS

We only modify the evaluator when it's deemed erroneous due to the wrong task labels or misuse of evaluating functions. When the task definition and corresponding evaluation metric match to some extent but might be misleading to most agents and even to human, we still keep the original ones to ensure the slightest reasonable changes. **We emphasize that we re-implement WebArena's base agent SteP's agent with the same web environment and modified evaluators as AGENTOCCAM for a fair comparison.** For example, we keep the evaluators of shopping tasks defined with template 163, requiring the agent to `"Draft an email to the shop owner via their contact us function for a coupon as {reason}"`, which doesn't explicitly specify whether to submit the drafted email. However, the evaluator is defined to assess the not yet submitted email. All capable LLM-based agents we have tested, which have been aligned to be helpful, will for sure submit the email if not directly prompted to behave in the way the evaluator desires, leading the email field to be blank and thus failing those tasks. Another example of this kind is the Reddit task asking the agent to find the most appropriate subreddit to post (task template 6100), where the assessment of appropriation is very subjective. In all those tasks, we follow the original evaluators, though their evaluation outcomes are arguably questionable.

We categorize our evaluator modification into two classes, namely label errors and improper evaluation function selection, raise representative examples for each class, and list all the changes made.

Label errors: We find there exist evaluator definition errors and some typos in the correct answers. In the later cases, the tasks always require exact matching, where any well-aligned LLM-agent would correct those typos in their generation. We thus rectify those errors:

i) Evaluator definitions contain errors. For example, in the Reddit task 584, the evaluator would open up the wrong page for the evaluation. Another case in point is the shopping task 261, where the `url_match` evaluator is constrained to identifying one correct url (`<server_host>:7770/electronics/headphones.html`), misjudging the same page (of the identical content) with a different url (`<server_host>:7770/electronics.html?cat=60`). Tasks fall in this category include: **261-264, 707-709, 584**.

ii) The answer contains typos or grammatical errors. For example the `is car necessary in NYC` in task 601, or the `budge` in task 603. More tasks of this kind include: **task id 489, 583, 601, 603, 606, 608**.

Improper evaluation function selection: Evaluator problems are more obvious in this case with the following types:

i) Use the `exact_match` function that compares whether the answer given by a human label-er and the answer returned by the agent is identically the same. Errors occur when the agent returns a full-form or a more complete answer, where the evaluators' labels cannot match. For example, in Reddit task 644 that requires the agent to post a meeting notice with the meeting date, where the keyword match for such date is exactly the `Dec 15th`, where the evaluator would judge other answers like `December 15th` as incorrect, where we change the keyword matching to one that could match both `Dec 15th` and `December 15th`. (In other cases with a single answer, we simply replace `exact_match` with `fuzzy_match`, which for instance in task 254 it could match `4125785000` with the agent's answer `The phone number is 4125785000`; or replace `exact_match` with `must_include`, which for instance in task 363 it could match `778m` with the agent's answer `778 m`.) It also demands that the answer should strictly include expressions like `virtual meetup`, where the agent might add other words in the `virtual` and `meetup`. In that sense, we also split the keyword `virtual meetup` into two separate keywords, i.e., `virtual` and `meetup`. Tasks of this kind include: **task id 146, 178-182, 254, 308-312, 330, 363-367, 415-418, 528-532, 640-649, 653-657, 679**.

ii) Use the poorly defined `fuzzy_match` function, that would view the answer returned as unqualified for the missing-from-expression answer exploration process, or assess answers with more detailed answers as partially correct (reward=0). We thus shift our prompt for the `fuzzy_match` function from: *"Help a teacher to grade the answer of a student given a question. Keep in mind that the student may use different phrasing or wording to answer the question. The goal is to evaluate whether the answer is semantically equivalent to the reference answer."* to *"Help a teacher to grade* *the answer of a student given a question.* ***Keep in mind that the student has executed the actions to get the answer.*** *They are allowed to use different phrasing or wording to answer the question. The goal is to evaluate whether the key points in the reference answer are included in the student's answer.* ***We allow answers with additional information that doesn't contradict the reference answer and review them as fully (not partially) correct.****"*

iii) Misuse the `fuzzy_match` function by splitting the keyword list for matching into a list, where each of the keyword and the entire answer, would be evaluated as partially correct (reward=0). In other words, no answer would be assessed as the correct answer (even the gloden-standard answer itself) due to such evaluator function misuse. This could be inferred from the function and the evaluator's definition. Tasks of this type include: **task id 16-20,** In such tasks, we simply merge the list of keywords into a string, concatenated with `"; "`. For instance, for task 16, the previous `fuzzy_match` field is `["driving: 2min", "walking: 16min"]`, and we modify it to `["driving: 2min; walking: 16min"]`.

Table 7: Action statistics.

Exp.	click	hover	type	scroll	go_back	goto	note	stop	go_home	branch	prune
AgentOccam	4715	-	1159	-	339	-	197	769	42	34	47
AgentOccam + SteP	5235	198	1407	11	25	132	124	1733	-	-	-
AgentOccam + Judge	4893	-	1297	-	261	-	127	726	94	220	41

Table 8: Average number of steps per task across all WebArena sites.

Exp.	All	Shopping	Shopping Admin	GitLab	Map	Reddit	Multisite
AgentOccam	9.0	6.7	9.2	10.8	8.5	8.6	13.3
AgentOccam + SteP	11.6	10.3	12.0	10.6	12.0	14.6	11.0
AgentOccam + Judge	9.4	6.7	10.5	10.6	9.6	8.4	13.5

Table 9: Average observation tokens per step across WebArena sites.

Exp.	All	Shopping	Shopping Admin	GitLab	Map	Reddit	Multisite
AgentOccam	2932.1	1634.2	4920.7	3126.8	1056.0	3697.8	1282.9
AgentOccam + SteP	2601.1	1675.2	3833.3	2983.8	1196.4	3071.4	1581.9
AgentOccam + Judge	2646.4	1773.8	4181.2	2848.4	729.7	3285.4	1433.2

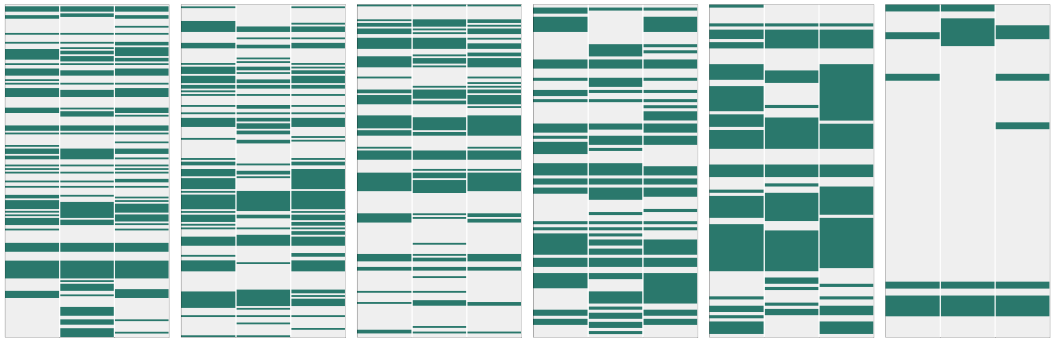
B Additional Experiment Details

We include the trial statistics for experiments that combine AgentOccam with other compound agent policies like SteP's strategies and our newly proposed Judge agent. Specifically, 7 shows these well performing agent are equally open to web environment exploration, actively issuing environment-changing actions like `click` and `type`. Not surprisingly, the AgentOccam + SteP agent frequently issuing un-interactive actions like `hover`. From Table 8, we can observe that AgentOccam finish the task with the fewest steps, often yielding a task result with 9 steps. Last, from Table 9, those three agents' token consumptions are of comparative orders of magnitude.

As shown in Figure 6, agents that combing AgentOccam with compound agent policies possess different behavioral success patterns. For AgentOccam + SteP, it benefits in tasks where the agent could easily be guided with detailed instructions, such as shopping tasks, with more success (green) blocks and denser success rate in tasks defined with the identical templates. However, it falls short in tasks that require generalizable skills like shopping admin tasks, and in tasks where task-specific strategies distract, like reddit tasks. On the contrary, AgentOccam + Judge agent shares similar patterns with the AgentOccam agent except that some of the success blocks are denser, thanks to the behavior rectification enabled by the action generation and selection pipeline.

In addition, we add the success rate figures of the ablation studies in Table 10, which has been visually represented in Figure 5.

We attach the trajectory logs for the experiments of AgentOccam, AgentOccam +SteP, AgentOccam +Judge, and ablation study of simplifying web page elements (`Above + Obs Opt.`) in supplementary, since agent behaviors in some tasks were referred to in the main text. We cannot attach logs of other experiments due to the space limit of supplementary material.



(a) Shopping. (b) Shopping admin. (c) GitLab. (d) Map. (e) Reddit. (f) Multisite.

Figure 6: Success patterns of AGENTOCCAM (leftmost in each sub figure), AGENTOCCAM + SteP, and AGENTOCCAM + Judge (rightmost) across different sites on WebArena. The y-axis represents task ids, with **green** indicating successful trials and **grey** indicating unsuccessful trials. Notably, tasks defined with the same templates are clustered together.

Table 10: Comparison of the success rate (SR) of AGENTOCCAM with baseline agents on WebArena.

Agent	Model	SR (%) (#812)	Shopping (#187)	Shopping Admin (#182)	GitLab (#180)	Map (#109)	Reddit (#106)	Multisite (#48)
Vanillar	GPT-4-Turbo	16.5	16.6	15.9	10.0	22.9	21.7	**16.7**
↓ `Actions`	GPT-4-Turbo	25.9	23.5	23.6	24.4	34.9	33.0	12.5
`Above + X Scrolling`	GPT-4-Turbo	31.7	26.2	25.3	35.0	33.0	52.8	14.6
`Above + Obs Opt.`	GPT-4	37.1	35.8	37.4	26.7	45.0	57.5	**16.7**
`Above + History`	GPT-4	38.2	33.7	40.1	31.7	**50.5**	51.9	14.6
AGENTOCCAM	GPT-4-Turbo	**43.1**	**40.6**	**45.6**	**37.8**	46.8	**61.3**	14.6

C AGENT PROMPTS

We list all agent prompts.

C.1 AGENTOCCAM

The general prompt template:

- **With planning**

```
You are an AI assistant performing tasks on a web browser.
You will be provided with task objective, current step, web page observations, previous plans,
and interaction history.
You need to issue an action for this step.

Generate the response in the following format:
{output_instructions}

You are ONLY allowed to use the following action commands.
Strictly adheres to the given format. Only issue one single action.
If you think you should refine the plan, use the following actions:
{planning_instructions}
Otherwise, use the following actions:
{navigation_instructions}
```

- **Without planning**

```
You are an AI assistant performing tasks on a web browser.
You will be provided with task objective, current step, web page observations,
and other relevant information.
You need to issue an action for this step.

Generate the response in the following format:
{output_instructions}
```

```
You are ONLY allowed to use the following action commands.
Strictly adheres to the given format. Only issue one single action.
{navigation_instructions}
```

Output specifications:

- **Interaction history summary**: Emphasize all important details in the INTERACTION HISTORY section.
- **Observation description**: Describe information in the CURRENT OBSERVATION section. Emphasize elements and features that are relevant or potentially helpful for fulfilling the objective in detail.
- **Reason**: Provide your rationale for proposing the subsequent action commands here.
- **Action**: Select your action here.
- **Observation Highlight**: List the numerical ids of elements on the current webpage based on which you would issue your action. Also include elements on the current webpage you would attend to if you fail in the future and have to restore to this step. Don't include elements from the previous pages. Select elements at a higher hierarchical level if most their children nodes are considered crucial. Sort by relevance and potential values from high to low, and separate the ids with commas. E.g., "1321, 52, 756, 838".

Action space specifications:

- **click**: n element with its numerical ID on the webpage. E.g., "click [7]" If clicking on a specific element doesn't trigger the transition to your desired web state, this is due to the element's lack of interactivity or GUI visibility. In such cases, move on to interact with OTHER similar or relevant elements INSTEAD.
- **type**: type [id] [content] [press_enter_after=0|1]: To type content into a field with a specific ID. By default, the "Enter" key is pressed after typing unless "press_enter_after" is set to 0. E.g., "type [15] [Carnegie Mellon University] [1]" If you can't find what you're looking for on your first attempt, consider refining your search keywords by breaking them down or trying related terms.
- **go_back**: go_back: To return to the previously viewed page.
- **note**: note [content]: To take note of all important info w.r.t. completing the task to enable reviewing it later. E.g., "note [Spent $10 on 4/1/2024]"
- **stop**: stop [answer]: To stop interaction and return response. Present your answer within the brackets. If the task doesn't require a textual answer or appears insurmountable, indicate "N/A" and additional reasons and all relevant information you gather as the answer. E.g., "stop [5h 47min]"
- **branch**: branch [parent_plan_id] [new_subplan_intent]: To create a new subplan based on PREVIOUS PLANS. Ensure the new subplan is connected to the appropriate parent plan by using its ID. E.g., "branch [12] [Navigate to the "Issue" page to check all the issues.]"
- **prune**: prune [resume_plan_id] [reason]: To return to a previous plan state when the current plan is deemed impractical. Enter the ID of the plan state you want to resume. E.g., "prune [5] [The current page lacks items "black speaker," prompting a return to the initial page to restart the item search.]"
- **go_home**: go_home: To return to the homepage where you can find other websites.

Observation space example:

```
RootWebArea [1] 'Dashboard / Magento Admin'
    link [178] 'Magento Admin Panel'
    menubar [85]
            link [87] 'DASHBOARD'
            link [90] 'SALES'
            link [96] 'CATALOG'
            link [102] 'CUSTOMERS'
            link [108] 'MARKETING'
            link [114] 'CONTENT'
            link [120] 'REPORTS'
            link [138] 'STORES'
            link [144] 'SYSTEM'
```

```
            link [150] 'FIND PARTNERS & EXTENSIONS'
    heading 'Dashboard'
    link [254] 'admin'
    link [256]
    textbox [894] [required: False]
    main
            text 'Scope:'
            button [262] 'All Store Views'
            link [265] 'What is this?'
            button [240] 'Reload Data'
            HeaderAsNonLandmark [898] 'Advanced Reporting'
            text "Gain new insights and take command of your business' performance, using our dynamic product, order,...
            link [902] 'Go to Advanced Reporting'
            text 'Chart is disabled. To enable the chart, click'
            link [906] 'here'
            text 'Revenue'
            text 'Tax'
            text 'Shipping'
            text 'Quantity'
            tablist [57]
                    tab [59] 'The information in this tab has been changed. This tab contains invalid data...
                            link [67] 'The information in this tab has been changed. This tab contains invalid data...
                                    text 'The information in this tab has been changed.'
                                    text 'This tab contains invalid data. Please resolve this before saving.'
                                    text 'Loading...'
                    tab [61] 'The information in this tab has been changed. This tab contains invalid data...
                            link [69] 'The information in this tab has been changed. This tab contains invalid data...
                                    text 'The information in this tab has been changed.'
                                    text 'This tab contains invalid data. Please resolve this before saving.'
                                    text 'Loading...'
                    tab [63] 'The information in this tab has been changed. This tab contains invalid data...
                            link [71] 'The information in this tab has been changed. This tab contains invalid data...
                                    text 'The information in this tab has been changed.'
                                    text 'This tab contains invalid data. Please resolve this before saving.'
                                    text 'Loading...'
                    tab [65] 'The information in this tab has been changed. This tab contains invalid data...
                            link [73] 'The information in this tab has been changed. This tab contains invalid data...
                                    text 'The information in this tab has been changed.'
                                    text 'This tab contains invalid data. Please resolve this before saving.'
                                    text 'Loading...'
            tabpanel 'The information in this tab has been changed. This tab contains invalid data...
                    table
                            row '| Product | Price | Quantity |'
                            row '| --- | --- | --- |'
                            row '| Sprite Stasis Ball 65 cm | 27.00 | 6 |'
                            row '| Quest Lumaflex Band | 19.00 | 6 |'
                            row '| Sprite Yoga Strap 6 foot | 14.00 | 6 |'
                            row '| Sprite Stasis Ball 55 cm | 23.00 | 5 |'
                            row '| Overnight Duffle | 45.00 | 5 |'
            text 'Lifetime Sales'
            text 'Average Order'
            text 'Last Orders'
            table
                    row '| Customer | Items | Total |'
                    row '| --- | --- | --- |'
                    row '| Sarah Miller | 5 | 194.40 |'
                    row '| Grace Nguyen | 4 | 190.00 |'
                    row '| Matt Baker | 3 | 151.40 |'
                    row '| Lily Potter | 4 | 188.20 |'
                    row '| Ava Brown | 2 | 83.40 |'
            text 'Last Search Terms'
            table
                    row '| Search Term | Results | Uses |'
                    row '| --- | --- | --- |'
                    row '| tanks | 23 | 1 |'
                    row '| nike | 0 | 3 |'
                    row '| Joust Bag | 10 | 4 |'
                    row '| hollister | 1 | 19 |'
                    row '| Antonia Racer Tank | 23 | 2 |'
            text 'Top Search Terms'
            table
                    row '| Search Term | Results | Uses |'
                    row '| --- | --- | --- |'
                    row '| hollister | 1 | 19 |'
                    row '| Joust Bag | 10 | 4 |'
                    row '| Antonia Racer Tank | 23 | 2 |'
                    row '| tanks | 23 | 1 |'
                    row '| WP10 | 1 | 1 |'
    contentinfo
            link [244]
            text 'Copyright 2024 Magento Commerce Inc. All rights reserved.'
            text 'ver. 2.4.6'
            link [247] 'Privacy Policy'
            link [249] 'Account Activity'
            link [251] 'Report an Issue'
```

C.2 JUDGE USED IN AGENTOCCAM + JUDGE EXPERIMENTS

The general prompt template:

```
You are a seasoned web navigator.
You now assess the value and risk of serveral web navigation actions based on the objective,
the previous interaction history and the web's current state.
Then, you select the action with the most value and least risk
with which you would earn the maximum objective fulfillment reward in the future.

Adhere to the following output format:
{output_instructions}

Note that `branch` and `prune` are planning actions that will modify the PREVIOUS PLAN section
and won't interact with the web environment.
```

Output specifications:

- **Plan progress assessment**: Review critically why the plans have not been fulfilled or the objective achieved. Justify your assessment with detailed evidence drawn from the objective, observations, and actions taken. Itemize the assessment using this format: "- plan [{plan_id}]\ n\t[{step_ids_taken_for_this_milestone}] [{concrete_proof_from_observation}] [{why_milestone_a_not_successful}]\ n\t[{step_ids_taken_for_this_milestone}] [{concrete_proof_from_observation}] [{why_milestone_b_not_successful}]\ n\t...".
- **Action assessment**: Assess the value and risk of each action. Consider both the best-case and worst-case outcomes resulting from its implementation. Itemize the assessment using this format: "- action [action_id]: [action value, including but not limited to what outcomes you can expect by executing the action, or whether the note is of the most correct and comprehensive content] [action risk, including but not limited to whether the note/stop content is correct, and whether you can gather more information by continuing playing rather than ending the trial] [best_case] [worst_case]".
- **Action selection**: List the numerical id of your selected action here. You can only choose one action. E.g., "1".