

---

# SIMURA: Towards General Goal-Oriented Agent via Simulative Reasoning Architecture with LLM-Based World Model

---

Mingkai Deng<sup>◊,†,\*</sup> Jinyu Hou<sup>◊,†,\*</sup> Yilin Shen<sup>‡</sup> Hongxia Jin<sup>‡</sup>  
 Graham Neubig<sup>†</sup> Zhiting Hu<sup>◊,▲</sup> Eric Xing<sup>◊,†</sup>

<sup>◊</sup>Institute of Foundation Models, Mohamed bin Zayed University of Artificial Intelligence

<sup>†</sup> School of Computer Science, Carnegie Mellon University

<sup>‡</sup> Samsung Research

<sup>▲</sup>Halicioğlu Data Science Institute, UC San Diego

{mingkaid, jinyuhou}@cs.cmu.edu, eric.xing@mbzuai.ac.ae

## Abstract

AI agents built on large language models (LLMs) hold enormous promise, but current practice focuses on a one-task-one-agent approach, which not only falls short of scalability and generality, but also suffers from the fundamental limitations of autoregressive LLMs. On the other hand, humans are general agents who reason by mentally simulating the outcomes of their actions and plans. Moving towards a more general and powerful AI agent, we introduce SIMURA, a goal-oriented architecture for generalized agentic reasoning. Based on a principled formulation of optimal agent in any environment, SIMURA overcomes the limitations of autoregressive reasoning by introducing a world model for planning via simulation. The generalized world model is implemented using LLM, which can flexibly plan in a wide range of environments using the concept-rich latent space of natural language. Experiments on difficult web browsing tasks show that SIMURA improves the success of flight search from 0% to 32.2%. World-model-based planning, in particular, shows consistent advantage of up to 124% over autoregressive planning, demonstrating the advantage of world model simulation as a reasoning paradigm. We are excited about the possibility for training a single, general agent model based on LLMs that can act superintelligently in all environments. To start, we make REASONERAGENT-WEB, a web-browsing agent built on SIMURA with pretrained LLMs, available as a research demo for public testing.

## 1 Introduction

AI agents powered by large language models (LLMs) hold tremendous potential for handling tasks that require flexible decision making. Recently, there have been great advancements in agents specialized in web and computer automation [1, 2, 3, 4], internet research [5, 6, 7], social simulation [8], software development [9, 10], scientific research [11, 12], and so on. Despite the promise, current LLMs often prove insufficient for solving complex agentic tasks, suffering from issues such as hallucination, repetitions, or failure at complex planning [13, 14]. To address these issues, many approaches focus on creating agents tailored to specific tasks like the above examples. However, this strategy have some inherent drawbacks. Economically, redesigning custom agents for every task is

---

\*Co-first author

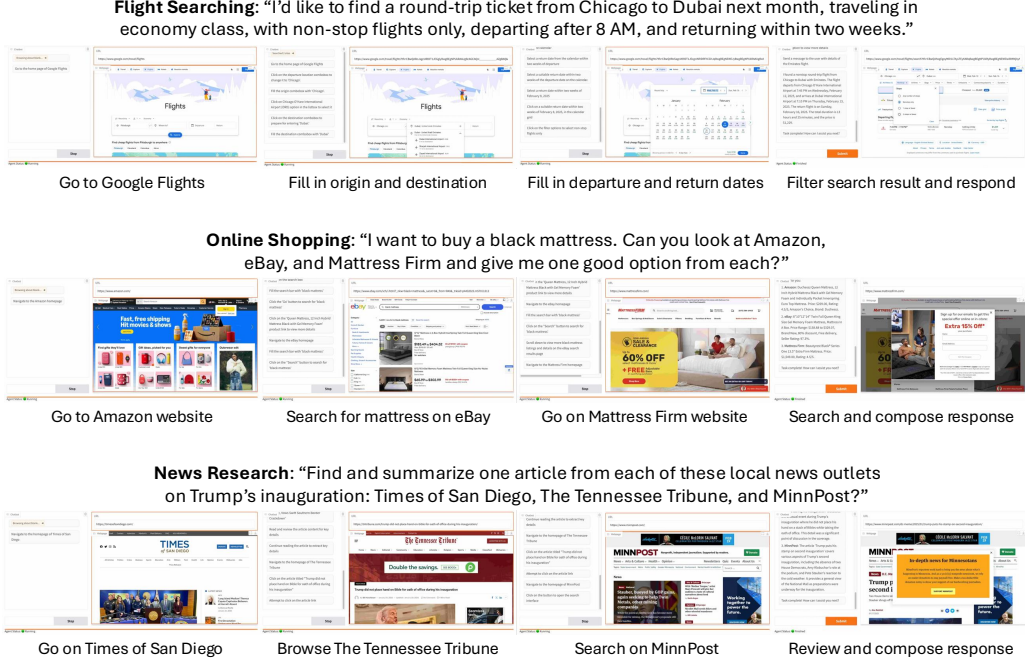


Figure 1: Demo of tasks performed using a web-browsing agent built on SIMURA with simulative planning using a LLM-based world model.

costly and not scalable from a business standpoint. Intellectually, narrowly focused solutions offer no clear path towards general and transferrable intelligence. [15] Technically, autoregressive LLMs rely on linear, step-by-step reasoning that often leads to errors that propagate through their thought trajectories [16, 17]. Humans, in contrast, are generalist problem-solvers that can reason and plan to achieve goals in diverse environments. Using a single cognitive system, we adapt to different tasks not only by linear reasoning, but also by imagining potential outcomes, simulating possibilities using a mental world model, and planning accordingly. [18]

Moving towards a more general and powerful AI agent, we introduce SIMURA (Simulative Reasoning Architecture), a goal-oriented architecture for generalized agentic reasoning. SIMURA mitigates the limitations of LLM autoregressive reasoning by introducing *world model* as the engine for planning via simulation. Specifically, a *policy* module first proposes a few potential actions, aimed at achieving specific goals based on agent identity and environment. Then, the *world model* simulates the outcomes of those proposed actions. Finally, a *critic* module evaluates these outcomes against the initial goals in order to select the best action from the candidates. Because simulating the full details of the world is infeasible and unnecessary for planning, we extract only the relevant information using natural language as a compact but complete representation, and simulate the next world in this latent space. To ensure robustness from observation noise and distracting execution details, we further propose a hierarchical architecture that isolates perception, simulative planning, and action selection which ensures adaptability and consistency across diverse tasks. Experiments on a range of web browsing tasks show SIMURA improving substantially compared to baselines, increasing the success rate of flight search from **0%** to **32.2%**, with reasoning by WM simulation outperforming LLM autoregressive reasoning by up to **124%**. Figure 1 shows examples of the agent performing multi-website, long-range task such as flight searching, online shopping, and news research.

For evaluation and demonstration purposes, we implemented SIMURA as an open-source library available via LLM Reasoners [19].<sup>2</sup> The resulting web agent, REASONERAGENT-WEB [20], is

<sup>2</sup><https://github.com/matrix-org/llm-reasoners/tree/main/examples/ReasonerAgent-Web>

available as a research preview.<sup>3</sup> We are actively expanding the system to address broader challenges and to further demonstrate its generality across a wider range of task domains.

## 2 Related Work

**LLM-Based Agents** LLM-based agents have rapidly evolved into versatile systems capable of autonomous behavior across a range of environments. One major approach to build such systems focuses on data collection in the targeted environment followed by model training. Notable examples include AutoWebGLM [21], AgentQ [22], UI-TARS [23], etc. Prompt-based workflows, on the other hand, have also shown strong potential when equipped with carefully designed modules, as demonstrated by recent work such as AWM [24], VOYAGER [25], and so on. SIMURA is built on prompt-based workflows but can leverage observation data for targeted improvement of its world model [26], leading to reduced reliance on human demonstration and strong generalizability to new tasks [18], which is an exciting next step.

**World-Model-Based Agents** Model-based planning for agents have long been frequently discussed and studied. Early work demonstrated the success of this approach by testing in classic games like go, chess, shogi and Atari. [27, 28]. Later on, world model was used for policy optimization and experimented on control tasks. [29, 30] In recent years, with the boost in foundation model’s capabilities, world-model-based planning was applied to more complex problems like math reasoning [31], playing Minecraft [32], and web browsing [33]. However, these world models typically represent and predict the world states using holistic continuous embeddings, which suffer from noise and high variability which detracts from robust and stable decision-making [34]. SIMURA instead adopts natural language as a discrete, concept-based latent space for consistent representation and prediction, which shows more general applicability across tasks in practice.

**Web Browsing Agents** Web browsing and navigation were chosen to evaluate SIMURA due to their realism and the complex decision-making they demand across diverse, dynamic interfaces. Recent years have seen the emergence of several prominent web-browsing agents, from proprietary ones such as OpenAI’s Operator [1], Anthropic’s Computer Use [6], and Google-DeepMind’s Project Mariner [2], and open-source ones including OpenHand’s BrowsingAgent [35], WebVoyager [36], CogAgent [37] and WebAgent [38]. These agents are typically built on simple ReAct-based [17] autoregressive reasoning which have difficulty recovering from previous mistakes; their often specialized design also preclude these approaches from generalizing to other task domains like social interactions and the physical world. Numerous benchmarks have been introduced to evaluate these web agents, including WebArena [3], WebVoyager [36] MiniWoB++ [39], Mind2Web [40], and WebShop [41]. Despite wide adoption, these benchmarks are usually either built in simulated and simplified environments, based on outdated questions, or lacks convincing method of measuring task completion, which detract from the goal of evaluating practically useful web agents. To address these challenges, we build FlightQA, an new dataset for evaluating agent ability in real-time complex website navigation. More details are included in Section 4.1.

**Generalist Agents** There have been various attempts of building generalist agents recently. One major approach focuses on creating a multi-agent system that consists of a unified interface on top of a few specialist agents that collaborates to decompose and complete complex tasks. [42, 43, 44, 45] Although this approach could lead to impressive performance on benchmarks, it has a few inherent limitations. First of all, tasks in reality could be versatile and may constantly require new specialist agents to be added to the system to achieve optimal performance, which is not efficient. Moreover, independently trained specialist agents for different domains are unable to leverage shared experience in the way that world model training enables. Finally, error propagation along the interaction trajectory remains an open challenge and is further complicated by the presence of multiple agents. Another popular approach utilizes framework similar to the CodeActAgent [46]. These agents [35, 47, 48] suffer from inaccurate code plans and have limited ability to revise or correct prior errors as well. SIMURA, on the other hand, is able to avoid these limitations by working as a monolithic architecture in which world model act as a central planning component.

---

<sup>3</sup><https://easyweb.maitrix.org/>

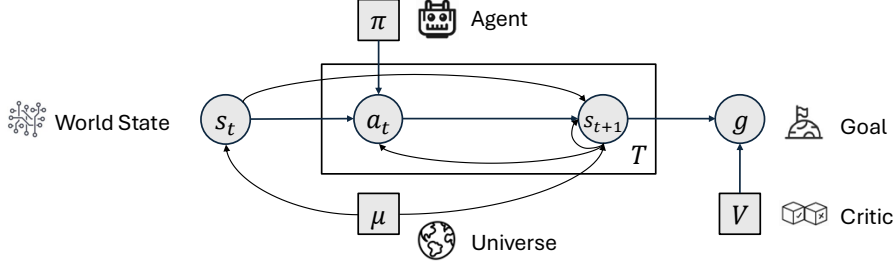


Figure 2: A possible definition of an optimal agent

### 3 SIMURA: Generalized Architecture for Optimal Goal-Oriented Agent

#### 3.1 Formulation of Agent-Environment Model

We first present our formulation of an optimal goal-oriented agent following the agent-environment model presented in [49]: We consider an agent  $\pi$  with identity  $i$  (e.g., name, description) and goal  $g$  acting in environment  $\mu$  (e.g., web browser, physical world, the entire universe) with action space  $\mathcal{A}$  and state space  $\mathcal{S}$ . Formally, at each time step  $t$ , the **agent**  $\pi$  takes the current state  $s_t \in \mathcal{S}$  and outputs the next action  $a_t \in \mathcal{A}$  following a policy distribution  $p_\pi(a_t | s_t)$ , while the **environment**  $\mu$  takes the current state  $s_t$  and action  $a_t$ , and outputs the next state  $s_{t+1} \in \mathcal{S}$  based on the distribution  $p_\mu(s_{t+1} | s_t, a_t)$ . We can thus denote the distribution of the interaction trajectory up to timestep  $T$ , or  $(a_t, s_{t+1}, \dots, a_{T-1}, s_T)$  given the current state  $s_t$  as below:

$$p_\mu^\pi(a_t, s_{t+1}, \dots, a_{T-1}, s_T | s_t) = \prod_{k=t}^{T-1} \underbrace{p_\pi(a_k | s_k)}_{\text{agent}} \underbrace{p_\mu(s_{k+1} | s_k, a_k)}_{\text{environment}} \quad (1)$$

In each state  $s_t$ , the agent also receives a reward  $r(g, s_t)$  based on its goal  $g$ . We evaluate the agent by its discounted cumulative reward, denoted as  $\sum_{k=t}^{\infty} \gamma_k r(g, s_k)$  (with the discount parameter  $\gamma_t$  decaying to zero with time, i.e.,  $\lim_{t \rightarrow \infty} \gamma_t = 0$ ). Note that this reward function can be dense (e.g., gaming scores), but perhaps frequently sparse (e.g., curing a disease). The agent's long-term success can thus be measured by its expected future discounted reward, also known as **value function** [50], which satisfies the following recurrence:

$$\begin{aligned} V_{\pi, \mu}^g(s_t) &:= \mathbb{E}_{\pi, \mu} \left[ \sum_{k=t}^{\infty} \gamma_k r(g, s_k) \mid s_t \right] \\ &= \lim_{T \rightarrow \infty} \sum_{(a_t, s_{t+1}, \dots, s_T)} \sum_{k=t}^T \gamma_k r(g, s_k) p_\mu^\pi(a_t, s_{t+1}, \dots, s_T | s_t) \\ &= \sum_{(a_t, s_{t+1}, \dots, s_T)} \underbrace{\left( \sum_{k=t}^{T-1} \gamma_k r(g, s_k) + \gamma_T V_{\pi, \mu}^g(s_T) \right)}_{\text{goal progress}} \underbrace{p_\mu^\pi(a_t, s_{t+1}, \dots, s_T | s_t)}_{\text{trajectory}}, \quad (2) \end{aligned}$$

Which indicates that the value function in state  $s_t$  can be expressed in terms of the value function at possible future states  $s_T$  weighted by their probabilities.

#### 3.2 Definition of Optimal Agent

Based on Equations 1 and 2, we can define the optimal agent  $\pi_\mu^*$  in environment  $\mu$  as one that maximizes the value function, written formally as below:

$$\pi_\mu^* := \arg \max_{\pi} V_{\pi, \mu}^g. \quad (3)$$

Some simple derivation will show that the optimal agent in state  $s_t$  will follow the following decision rule  $\pi_\mu^*$  when planning for actions  $a_{t:T-1}$ :

$$\pi_\mu^*(s_t) = \arg \max_{\underbrace{a_{t:T-1}}_{\text{possible actions}}} \sum_{s_{t+1:T}} \underbrace{\left( \sum_{k=t}^{T-1} \gamma_k r(g, s_k) + \gamma_T V_{\pi, \mu}^g(s_T) \right)}_{\text{goal progress}} \prod_{i=t}^{T-1} \underbrace{p_\mu(s_{i+1} | s_i, a_i)}_{\text{universe response}} \quad (4)$$

In practice, agents often samples promising action candidates using a policy function  $\tilde{\pi}$  through the distribution  $p_{\tilde{\pi}}(a_t | s_t)$ . Building the optimal agent thus requires capabilities for proposing possible actions ( $\tilde{\pi}$ ), predicting their outcomes ( $\mu$ ), and evaluating goal progress ( $r, V$ ), respectively. Note that typical reactive agents that output the next action directly can be seen as taking the first sample from  $\tilde{\pi}$  (similar to “System 1” in humans which makes fast, instinctive reactions [51]), without simulating and evaluating the outcomes using  $\mu$  and  $V$  (similar to “System 2” responsible for deliberate decision-making). In terms of LLM-based agents, this can also be seen as the agent generating a plan using autoregressive LLMs, which has no way of correcting errors during the sampling process.

### 3.3 World Model for Generalized Simulative Reasoning

Note that the optimal decision-making process defined in Equation 4 requires the agent to have access to the ground-truth world state  $s$  and the environment  $\mu$  to experience and optimize over. However, these are often not available aside from simple scenarios like Go and Chess games [52, 53] – imagine building a spacecraft to land on Mars, or simply a humanoid robot relying on noisy sensors in daily environments. World Model (WM) thus arises as a crucial component for predicting any environment’s response to a general agent. Specifically, a WM  $f$  operates on an internal representation of the world state, denoted as a *belief state*  $\hat{s}_t$ , which is derived from sensory inputs  $o_t$  via an *Encoder*  $h$  (unlike the optimal agent described in §3.2 which has direct access to the true world state  $s_t$ ). Given proposed action  $a_t$ , the WM predicts the next belief state  $\hat{s}_{t+1}$  according to the distribution  $p_f(\hat{s}_{t+1} | \hat{s}_t, a_t)$ . The predicted belief state then allows the agent to propose the next action, continuing the cycle of prediction and action up to the desired time horizon  $T$ . Thus, a WM here essentially functions as a generative model of possible future world states, which enables simulative reasoning, or “thought experiments”. Formally, for the optimal agent  $\pi_f^*$  equipped with WM  $f$  in belief state  $\hat{s}_t$ , we define the simulation-based decision rule in Equation 6 as follows:

$$\pi_f^*(\hat{s}_t) = \arg \max_{\underbrace{a_{t:T-1}}_{\text{possible actions}}} \sum_{\hat{s}_{t+1:T}} \underbrace{\left( \sum_{k=t}^{T-1} \gamma_k r(g, \hat{s}_k) + \gamma_T V_{\pi, f}^g(\hat{s}_T) \right)}_{\text{goal progress}} \prod_{i=t}^{T-1} \underbrace{p_f(\hat{s}_{i+1} | \hat{s}_i, a_i)}_{\text{simulation with world model}} \quad (5)$$

A general-purpose WM  $f$  here enables simulation of diverse possibilities across a wide range of domains, enabling agents to reason about outcomes without direct interaction with the environment.

### 3.4 Design of Simulative Reasoning Agent Using LLM-Based World Model

In this subsection, we present our design of a generalizable simulative reasoning agent using large language models (LLMs) as building blocks due to the latter’s strength in a wide range of capabilities such as summarization, commonsense knowledge, reflection, and tool use, which are gained from large-scale pretraining and instruction tuning. In particular, we provide detailed discussion on design decisions that enable robust and general applicability across environments and tasks.

**Discrete, Hierarchical State Representation via Natural Language** The dominant approach to encoding observation  $o_t$  (e.g., webpages, video streams) has been to directly pass all input tokens into an LLM to form continuous embeddings  $\hat{z}_t^z$ . While technically preserving all information, real-world sensory readings often suffer from inherent noise and high variability (e.g., ads on a webpage, varying weather and lighting conditions in video), which can make them brittle for reasoning over. Human cognition, on the other hand, has evolved to counter this variability by categorizing raw perception into *discrete concepts* [34], which are often encoded in language, symbols or structured thoughts. Indeed, natural language is inherently hierarchical, capable of encoding concepts from concrete ones (e.g., apple) to highly abstract ones (e.g., religion). Discrete representations are also complete in general [49], which ensures no information is necessarily lost in the compression process.



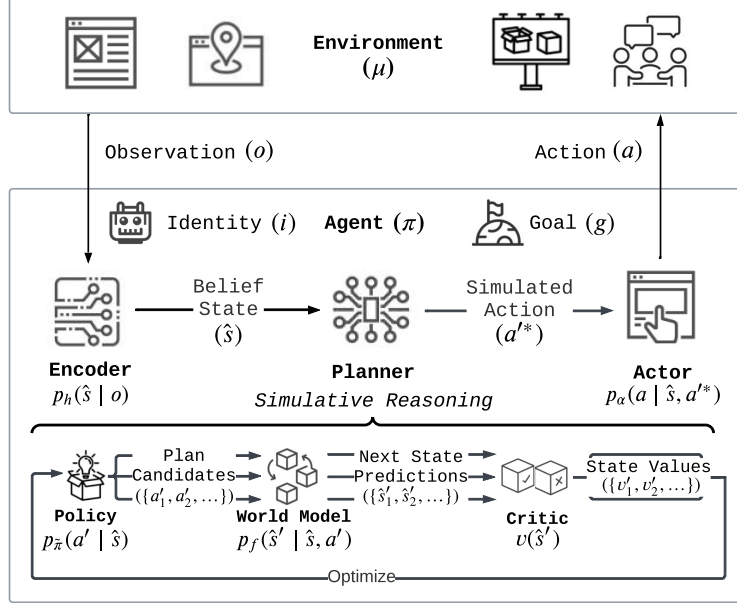


Figure 4: Optimal agent architecture design with conditional probability annotation

best simulated action  $a_t^{I*}$  through the planner. Inside the planner, the architecture performs simulative reasoning by proposing actions  $a_t'$  using policy  $\tilde{\pi}$  and predicting the next state  $\hat{s}_{t+1}$  using the world model  $f$ , and evaluating goal progress  $\sum_{k=t}^{T'-1} \gamma_k r(g, \hat{s}_k) + \gamma_{T'} V_{\pi, f}^g(\hat{s}_{T'})$  using critic  $v$  upon reaching state  $\hat{s}_{T'}$  at the planning horizon  $T'$ . This can repeat multiple times until the planner selects the action sequence  $a_{t:T'-1}^{I*}$  with the highest expected success and passes the first step  $a_t^*$  to actor  $\nu$  which finally outputs the concrete action  $a_t$ . Formally, SIMURA can be seen as solving the following multi-level optimization problem:

$$\begin{aligned}
 \hat{s}_t &= \arg \max_{\hat{s}} \underbrace{p_h(\hat{s} | o_t)}_{\text{encoder}} & (\text{Perception}) \\
 a_{t:T'-1}^{I*} &= \arg \max_{\substack{a_{t:T'-1} \\ \text{sampled from policy } \tilde{\pi}}} \sum_{\hat{s}_{t+1:T'}} \underbrace{v(\hat{s}_{T'})}_{\text{critic}} \prod_{k=t}^{T'-1} \underbrace{p_f(\hat{s}_{k+1} | \hat{s}_k, a'_k)}_{\text{world model}} & (\text{Planning}) \quad (8) \\
 a_t &= \arg \max_a \underbrace{p_\nu(a | \hat{s}_t, a_t^{I*})}_{\text{actor}} & (\text{Acting})
 \end{aligned}$$

In practice, we implement each of these components by zero-shot prompting pretrained LLMs. While these LLMs alone are often insufficient for many complex agentic tasks, SIMURA's divide-and-conquer approach combines existing LLM strengths like instruction-following, summarization, reflection, and tool use to allow agentic behavior to emerge. Benefiting from massive web-scale pretraining on next-token prediction  $p(x_t | x_{<t})$ , which is formally akin to world modeling, LLMs possess significant potential to serve as world models with natural-language state and action spaces [31, 55]. We approximately infer the world state  $\hat{s}_t$  and action  $a_t$  by sampling from the LLM-based encoder and actor distributions  $p_h$  and  $p_\nu$ , respectively. For planning, we optimize over the sampled actions  $a_{t:T'-1}^{I*}$  using readily available tree search algorithms like Depth-First Search (DFS) and Monte-Carlo Tree Search (MCTS).

## 4 Experiments

Our proposed SIMURA architecture is generally applicable to various environments and tasks. As our first step, we evaluate our implementation on **web browsing** as an example due to both its practical value and its technical challenge. Web browser is an indispensable portal for individuals to perform

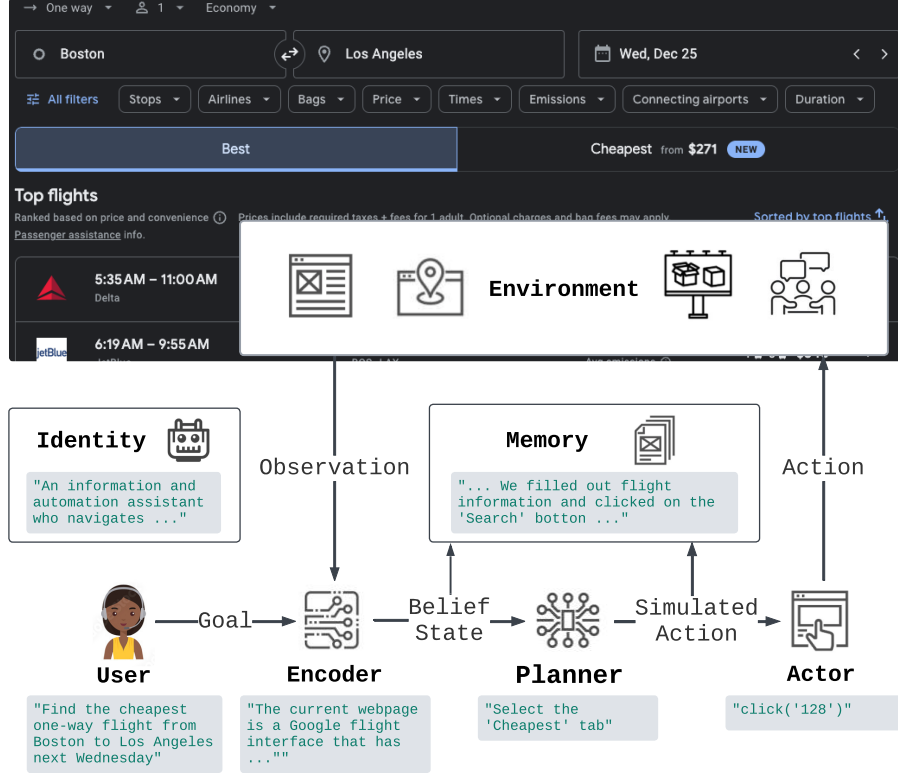


Figure 5: LLM-based implementation of our proposed agent model for web-related tasks (e.g. multi-website QA, flight search, etc). Planner is where we implement our proposed world-model-based planning. We also implement a baseline that simply samples the plan from a language model (i.e., autoregressive planning).

many tasks in real life (e.g., gather information, book travels, and submit applications). Whereas many existing products do access the internet [56, 57, etc.], they typically use specialized tools (e.g., search engines and data APIs) to capture a subset of web browser capabilities (i.e., reading) while falling short of the full functionality (e.g., access content not exposed to search engines or predefined APIs like flight and hotel databases). We argue that an agent that takes advantage of the full browser will push the envelope in AI’s abilities to serve human needs.

Despite the richness and flexibility, the web browser is a highly challenging environment for agentic reasoning due to its immense complexity, long-horizon nature, partial observability, and multimodality [3, 58]. We evaluate our architecture in 3 types of web browsing tasks: 1) complex website navigation, 2) multi-hop, multi-website QA, and 3) general web automation. For the baselines, we compare against:

1. BrowsingAgent from OpenHands [35], a representative open-web agent which generates chain-of-thought before selecting an action
2. SIMURA (our architecture) with autoregressive planning (i.e., commit to the first sample from our policy module) instead of our proposed simulation-based planning with world model. Formally, the planning process is simplified to the following:

$$a_t'^* = \arg \max_{a_t'} p_{\pi}(a_t' | \hat{s}_t)$$

**Implementation for Web Browsing** Figure 5 presents our implementation when applied to web browsing. We use prompts tailored to the web environments in this example, but plan to extend to other environments and move towards training a single agent model that can act optimally in all environments, which is an exciting next step. At each step  $t$ , the agent receives the observation  $o_t$  as the HTML-based accessibility tree visible through the browser’s viewport (an example is provided in



Web Task Performance vs. Agent Architecture

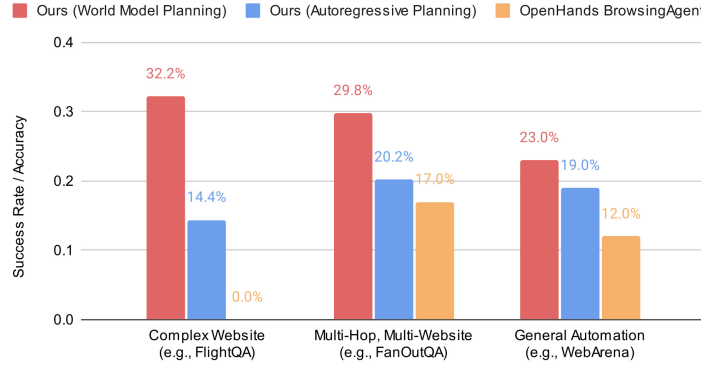


Figure 6: Overview of performance comparison between SIMURA and baselines. The full architecture shows clear advantage over the baseline BrowsingAgent, improving the performance on complex website navigation from 0% to 32.2%. Our proposed world model reasoning for planning also consistently improves over simple planning with autoregressive LLM by up to 124%.

Appendix A). The agent then uses encoder LLM  $h$  to summarize the observation as  $\tilde{s}_t \sim p_h(\cdot | o_t)$ , and then add it to a selective memory of past summaries and simulated actions  $\{m(\tilde{s}_k, a'_k)\}_{k=1}^{t-1}$  to form the estimated world state  $\hat{s}_t = [m(\tilde{s}_1, a'_1), \dots, m(\tilde{s}_{t-1}, a'_{t-1}), \tilde{s}_t]$  for planning. During planning, we sample  $M$  simulated actions  $a'_t$  from the policy  $\tilde{\pi}$ , cluster them into distinct actions, and use the world model  $f$  to predict the next summary as  $\tilde{s}_{t+1} \sim p_f(\cdot | \hat{s}_t, a'_t)$  to form the next state  $\hat{s}_{t+1} = [m(\tilde{s}_1, a'_1), \dots, m(\tilde{s}_t, a'_t), \tilde{s}_{t+1}]$ ; this repeats until the planning horizon  $T$ . To evaluate the terminal state  $\hat{s}_T$  with critic  $v$ , we prompt the LLM to generate qualitative answers and convert them into numerical scores (e.g., “success” receives a score of 1), and repeat for  $N$  times to capture the fine-grained differences between states. Following previous work [59, 33], we set  $M = N = 20$  and  $T = t + 1$ , and use DFS as the search algorithm. We implement the planning process using LLM Reasoners [19], a library for LLM-based complex reasoning using advanced algorithms. After the planner selects the simulated action  $a'_t$ , we update the memory with  $m(\tilde{s}_t, a'_t)$ . For the actor  $\nu$ , we additionally include the observation text  $o_t$  in the prompt to ensure the action grounding. All the prompts are included in Appendix B.

**Overview of Results** An overview of our results is presented in Figure 6. Across all 3 categories of tasks, our architecture shows a clear advantage over the baseline BrowsingAgent, specifically increase the success rate on complex website navigation from 0% to 32.2%. Our proposed world model reasoning for planning also consistently improves over simple planning with autoregressive LLMs by up to 124%. In the subsections below, we will introduce the evaluation settings and discuss the results in more detail.

#### 4.1 Complex Website Navigation

A distinguishing feature of web agents is the ability to gather live information (e.g., flights, stock prices, social media) not present in the training data of foundation models due to its rapid update [60]. For many questions (e.g., the price of the earliest flight tomorrow), LLMs without such grounding often result in hallucinations (see Figure 7 for an example). In practice, however, obtaining the information is challenging, as many websites are very complex and difficult to navigate (e.g., execute a flight search query on a travel website and filter through the results), which calls for superb reasoning skills on the part of the agent.

**Dataset** Due to the lack of benchmarks for complex website navigation on the open internet, we created our own dataset using flight search as an example, since it is an important part of travel planning, a major use case for web agents. We formulate flight search as the user asking for a flight satisfying a list of constraints (e.g., one-way, from New York to Los Angeles), and the agent operating a flight search engine to find such a flight and returning it to the user. To evaluate the agent’s reasoning

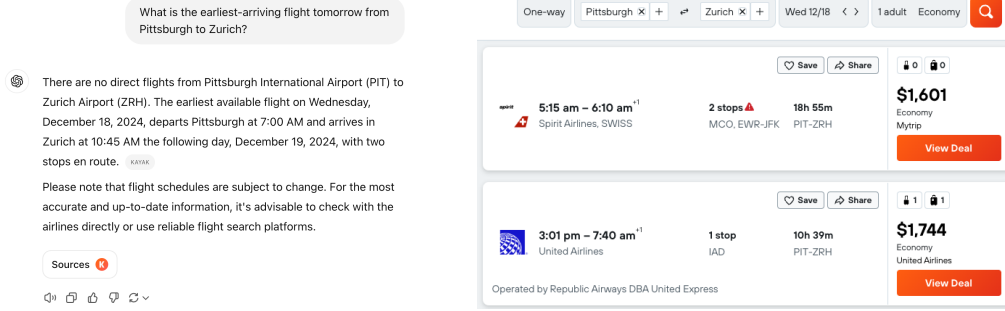


Figure 7: Faced with the question “What is the earliest-arriving flight tomorrow from Pittsburgh to Zurich?” ChatGPT-4o browsed the frontpage of Kayak.com and hallucinated a flight that arrives at 10:45am on the following day as the answer (left). Performing the search on Kayak.com, however, shows that the earliest-arriving flight lands in Zurich at 6:10am on the next day (right). The question was asked on December 17th, 2024.

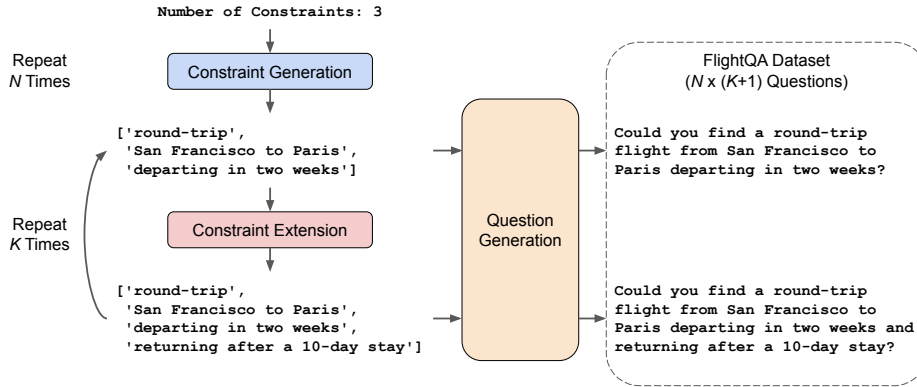


Figure 8: Illustration of the data generation process for the FlightQA dataset. We first prompt a LLM to generate  $N$  lists of  $C$  starting constraints (Constraint Generation). Then, we prompt the LLM to iteratively add constraints to the lists one by one, repeating for  $K$  times (Constraint Extension). Finally, we prompt the LLM to convert each constraint list into a question in natural language (Question Generation).

ability, we further produce questions with varying number of constraints by iteratively adding to the list, which enables a counterfactual analysis that controls for the confounding effect of specific constraint configurations (e.g., an agent with perfect reasoning should still be able to answer the same question with one more constraint; an agent of rote memorization will likely fail when the question changes slightly).

We illustrate our data collection process in Figure 8. To ensure scalability and controllability, we prompt a LLM to first generate a list of  $C$  starting constraints, repeating for  $N$  times. After that, we prompt the LLM to iteratively add constraints to the lists one at a time, repeating for  $K$  times. Finally, we prompt the LLM to convert each constraint list into a question in natural language. In practice, we set  $C = 3$ ,  $N = 15$ , and  $K = 5$ , which results in FlightQA, a dataset consisting of 90 questions with 15 sequences of constraint lists where the number of constraints increases from 3 to 8. We use gpt-4o to perform all the data generation steps. The initial question generation and question expansion prompts are included in Appendix C

**Evaluation** Because FlightQA involves querying live information from the open internet, it is impossible to establish ground truth answers due to the constantly evolving flight pricing and availability. Inspired by previous work on evaluation for generated text [61], we propose to evaluate the agent response based on two quality aspects: **groundedness** for whether the response is supported by the interaction history and **relevance** for whether the response satisfies user constraints to the

Method	Performance (%)			Outcomes (%)				
	Correct	Grounded	Relevant	Response Returned	Browser Crashed	Max Steps Reached	Repetitive Actions	Action Errors
OpenHands BrowsingAgent	0.0	0.0	0.0	0.0	3.3	3.3	0.0	93.3
SIMURA ( <i>Ours</i> )								
Autoregressive Planning	14.4	15.6	14.4	16.7	0.0	37.8	44.4	1.1
– with o1 <sup>†</sup>	1.1	1.1	1.1	1.1	11.1	40.0	37.8	10.0
– with o3-mini <sup>†</sup>	3.3	4.4	3.3	4.4	3.3	51.1	32.2	8.9
World Model Planning	<b>32.2<sup>**</sup></b>	<b>36.7</b>	<b>32.2</b>	<b>38.9</b>	1.1	40.0	18.9	1.1

Table 1: Performance and outcome statistics for the FlightQA dataset. Our architecture increases the correct rate from 0% in OpenHands BrowsingAgent to 32.2%. Reasoning by world model simulation also clearly outperforms autoregressive reasoning by 124%. \*\* indicates being significantly higher than the second-best method at the statistical significance level of 0.01 ( $p < 0.01$ ) based on pairwise t-test. <sup>†</sup>We implement the autoregressive planner with o1 and o3-mini, respectively.

extent allowed by the results (e.g., if the search results do not include any flight that satisfies all user constraints). Due to the strong ability of LLMs in evaluating generated text [62], we prompt LLMs to assess the two quality aspects of the agent response. Specifically, we include all browser observations in the agent’s trajectory over  $T$  steps ( $o_1, o_2, \dots, o_T$ ), the constraint list, the question, and the agent response, and ask the LLM to provide judgment on the groundedness and relevance of the response. We further define an answer to be **correct** when it is both grounded and relevant. We also include the evaluation prompt in Appendix C.

**Experiment Setup** We ran the experiments and evaluation using gpt-4o between November 24th, 2024 and December 9th, 2024. For the environment, we use BrowserGym [63], a popular open-source browser sandbox. We stop each run when the agent provides a response or after the agent takes 30 actions, whichever comes first. We also mark the run as failed when the agent repeats the same action for 3 times consecutively or when the agent causes more than 3 errors while interacting with the browser.

**Results** We present our Complex Website Navigation results in Table 1. Compared to BrowsingAgent which fails completely in this task, our full architecture improves the correct rate from 0% to 32.2%. Within our architecture, our proposed world-model-based planning shows superior performance over autoregressive reasoning with a 124% improvement (significant at the 0.01 level). The other components in our architecture, which communicate using the concept-based latent space of model-generated language (e.g., observation summary and selective memory), also result in more coherent behavior by reducing the action error rate in BrowsingAgent from 93.3% to 1.1%. However, the autoregressive reasoning still results in frequent repetitions, which is mitigated by world model-based planning (44.4%  $\rightarrow$  18.9%).

**Analysis of Reasoning Ability** To compare the reasoning abilities of autoregressive and world-model planners within our architecture, we visualize the percentage of correct responses vs number of constraints in Figure 9. As the questions in FlightQA are generated based on iteratively expanded constraint lists, this analysis should faithfully reflect the effect of increasing constraints while controlling for other confounders such as specific constraint sets. Based on our data samples, world model planning shows consistent advantage over autoregressive planning as we increase the number of constraints, showing signs of improved reasoning ability. The performance for both methods decreases with more constraints initially but then increases sharply at 7 constraints before dropping again, which may reflect memorization in the backend LLM or implicit constraints in questions with fewer explicit constraints.

## 4.2 Multi-Hop, Multi-Website QA

Another type of challenging questions for web agents is those that require gathering information about multiple entities over multiple websites. For instance, given the question “*What are the*

% Correct vs Number of Constraints

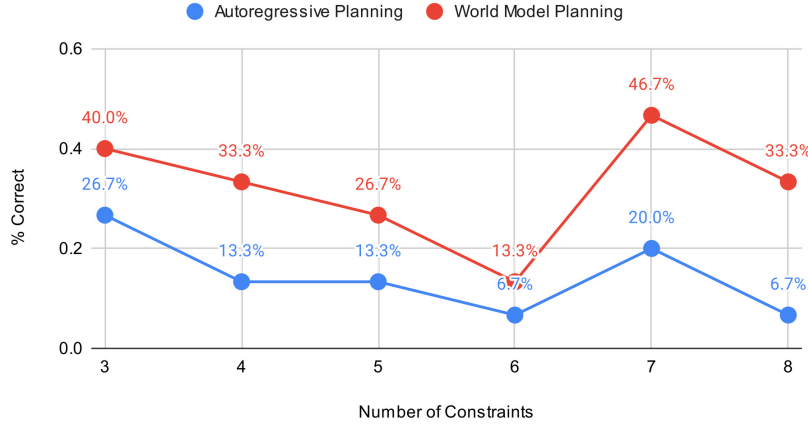


Figure 9: % correct and % response returned vs. number of constraints for FlightQA. Based on our data samples, world model planning consistently outperforms autoregressive planning as we increase the number of constraints, showing signs of improved reasoning ability.

*availabilities of the top-10 restaurants in Paris for a dinner next week?”*, an agent must first find the top-10 restaurants in Paris, then look up the availability of each restaurant, and finally compile the information into a response to the user. Whereas complex website navigation stresses the depth of individual websites, multi-hop, multi-website QA concerns the breadth of websites to navigate over long-horizon interactions.

**Dataset** To evaluate agent abilities for multi-hop, multi-website QA, we adopt the FanOutQA [64] dataset, which consists of questions of exactly this nature. Due to resource constraints, we evaluate on the first 100 examples of the dev set. As the results show, however, the smaller sample size is sufficient to show statistically significant differences between methods.

**Experiment Setup** We ran the experiments using gpt-4o-2024-05-13 between November 10th, 2024 and December 8th, 2024. We noticed that our architecture with world-model-based planning deteriorates in performance when using the newer versions of gpt-4o, which may be due to additional training which changed the model’s response patterns to the same prompts. We operate the browser using the same rules as in experiments for Complex Website Navigation.

**Results** We present our results on Multi-Hop, Multi-Website QA in Table 2. Again, our method increases the accuracy from 17.0% to 29.8% and world model planning improves over autoregressive planning by 48.6% (p-value = 0.011). BrowsingAgent achieves fair performance even though it cannot memorize information from different websites, often due to some questions in the dataset being answerable based on information from a single Wikipedia page (e.g., *What are the publication dates for all of the Harry Potter books?*). Despite this, our architecture improves over BrowsingAgent even without world model planning by dramatically reducing action errors (43% → 10%). Browser crashes make a sizable contribution to agent failures (24% for our architecture), indicating room for improvement in the tooling for open-web navigation.

### 4.3 General Web Automation

Last but not least, web agents are often tasked with performing various work tedious to human users (e.g., online shopping, managing social media). These tasks often require the ability to interact with a range of websites of moderate complexity. As an example, given the question “*Summarize customer reviews for Amazon Echo Dot 3rd generation,*” the agent should navigate a shopping website to locate and go over all the customer reviews of said product before summarizing the content for the user.

Method	Performance (%)		Outcomes (%)					
	Acc.	Acc. (Strict)	Response Returned	Browser Crashed	Max Steps Reached	Repetitive Actions	Action Error	Parsing Error
OpenHands BrowsingAgent	17.0	4.0	32.0	17.0	8.0	0.0	43.0	0.0
SIMURA ( <i>Ours</i> )								
Autoregressive Planning	20.2	3.0	37.0	24.0	10.0	18.0	10.0	1.0
World Model Planning	<b>29.8*</b>	4.0	<b>55.0</b>	24.0	12.0	8.0	1.0	0.0

Table 2: Performance and outcome statistics for the FanOutQA dataset. Acc. (Strict) refers to the percentage of responses that exactly match the groundtruth. Our architecture clearly outperforms the baseline BrowsingAgent. Reasoning by world model increases the response rate and fact-level accuracy vs. autoregressive planning by 48.6% and 47.5%, respectively. \* indicates being significantly higher than the second-best method at the 0.05 level based on pairwise t-test.

**Dataset** To evaluate general web automation capabilities, we adopt the WebArena [3] benchmark, a standard environment for testing web agents which features a range of simulated websites including a Reddit-like social forum, a shopping site, a GitLab-based code management platform, a map, and a Wikipedia-like encyclopedia. Following the evaluation for Multi-Hop, Multi-Website QA, we take a random subset of 100 examples.

**Experiment Setup** We run the experiments using gpt-4o over BrowserGym accessed via the OpenHands platform which provides a uniform evaluation procedure. Because WebArena demands a specific response format for evaluation, we rewrote the agent description to steer the agent answer format accordingly (Appendix B.1). We keep all other environment rules the same as previous experiments, except for setting the maximum allowed steps to 15 which is consistent with the default setting of WebArena.

**Results** We present our results on General Web Automation in Table 3. Continuing the patterns from previous experiments, our proposed architecture improves over BrowsingAgent by up to 91.7%, while within our architecture, world model reasoning improves over autoregressive reasoning by 21.1%, highlighting the comparative advantage under the given experimental setup.

Method	Success Rate (%)
OpenHands BrowsingAgent	12.0
SIMURA ( <i>Ours</i> )	
Autoregressive Planning	19.0
Ours (World Model Planning)	<b>23.0</b>

Table 3: Results on a random 100-sample subset of WebArena. Our architecture improves over BrowsingAgent by up to 91.7%, while world model planning improves over autoregressive planning by 21.1%.

## 5 Limitations

Due to the modular pipeline and thorough exploration of multiple plans in world model planning, the current agent takes longer than typical LLM agents to run. Speeding up world-model-based reasoning with appropriate caching and parallelization strategies is an important part of our future work. Agent capabilities can be limited by the tooling. For example, with open-source browser environments, web agents are often blocked by Captcha or anti-scraping tools from certain websites. Deeper integration with user browser can help solve this issue. As agent-based automation become more integrated into browsing and computer-use workflows, we also encourage conversations around fair use and protocols around agent access of certain websites. We are currently only using the text portion of the webpage observations, which can miss information like images and layout information (e.g., occlusions). While existing work are experimenting with visual-based web browsing, it is still

challenging to combine multimodal perception and planning, which we are excited to keep working on.

## 6 Conclusion

In this paper, we have presented SIMURA, a general goal-oriented architecture for optimal agent decision-making. Empowered by simulation-based planning using world model and modeling of agent internal belief space activities using natural language as latent representation, we see significant and strong improvements on a range of tasks in web browsing experiments, with world model-based reasoning showing improved reasoning capacity compared to LLM autoregressive reasoning.

We are very excited about the possibilities for a single, general, superintelligent agent, but are also keenly aware of the risks for individuals and societies. On the capability side, we aim to test on more types of environments (e.g., software development) and continue developing functional components that strengthen the agent (e.g., multi-agent interaction and long-term memory). On the safety / alignment side, we look forward to engaging the community in discussions about how to ensure such an agent stays aligned with our shared values, priorities, and welfare.

## Acknowledgement

This work was supported in part by the Samsung GRO Project “Efficient Designs for Generative and Agent LLM Development.” We thank Zhoujun Cheng, Shibo Hao, and Xinyu Pi from MixLab; Han Guo, Nicholas Ho, and Bowen Tan from SAILING Lab; Li Erran Li from AWS, Zora Wang from NeuLab; and Sarah Cheah and Hector Ren from MBZUAI for their insightful feedback and discussions. We are also grateful for their helpful suggestions throughout the project. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Samsung.

## References

- [1] OpenAI. Computer-using agent (cua). <https://openai.com/index/computer-using-agent/>, January 2025. Research preview of “Operator”, published January 23, 2025.
- [2] DeepMind. Project mariner. <https://deepmind.google/models/project-mariner/>, 2024. Accessed: 2025-07-16.
- [3] 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*, 2023.
- [4] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.
- [5] OpenAI. Introducing Deep Research. <https://openai.com/index/introducing-deep-research/>, February 2025. Deep Research agent release announcement.
- [6] Anthropic. Introducing computer use, a new claude 3.5 sonnet, and claude 3.5 haiku. <https://www.anthropic.com/news/3-5-models-and-computer-use>, October 22 2024. Public beta “computer use” feature for Claude 3.5 Sonnet.
- [7] Google. Gemini Deep Research: Your Personal Research Assistant. <https://gemini.google/overview/deep-research/?hl=en>, December 2024. Overview of Gemini Deep Research agent feature.
- [8] Joon Sung Park, Joseph C. O’Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

- [9] Anysphere Inc. Cursor: The ai code editor. <https://cursor.com>, 2025. Accessed: 2025-07-16.
- [10] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents, 2025.
- [11] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, Yossi Matias, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yunhan Xu, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Towards an ai co-scientist. *arXiv preprint arXiv:2502.18864*, February 2025. URL: [https://storage.googleapis.com/coscientist\\_paper/ai\\_coscientist.pdf](https://storage.googleapis.com/coscientist_paper/ai_coscientist.pdf).
- [12] Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search, 2025.
- [13] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, Yuheng Cheng, Suyuchen Wang, Xiaoqiang Wang, Yuyu Luo, Haibo Jin, Peiyan Zhang, Ollie Liu, Jiaqi Chen, Huan Zhang, Zhaoyang Yu, Haochen Shi, Boyan Li, Dekun Wu, Fengwei Teng, Xiaojun Jia, Jiawei Xu, Jinyu Xiang, Yizhang Lin, Tianming Liu, Tongliang Liu, Yu Su, Huan Sun, Glen Berseth, Jianyun Nie, Ian Foster, Logan Ward, Qingyun Wu, Yu Gu, Mingchen Zhuge, Xiangru Tang, Haohan Wang, Jiaxuan You, Chi Wang, Jian Pei, Qiang Yang, Xiaoliang Qi, and Chenglin Wu. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems, 2025.
- [14] Trishna Chakraborty, Udit Ghosh, Xiaopan Zhang, Fahim Faisal Niloy, Yue Dong, Jiachen Li, Amit K. Roy-Chowdhury, and Chengyu Song. Heal: An empirical study on hallucinations in embodied agents driven by large language models, 2025.
- [15] Cewu Lu and Shiquan Wang. The general-purpose intelligent agent. *Engineering*, 6(3):221–226, 2020.
- [16] Jacob Andreas. Language models as agent models, 2022.
- [17] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023.
- [18] Yann LeCun. A path towards autonomous machine intelligence. OpenReview preprint, June 2022. Version 0.9.2, June 27 2022.
- [19] Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, et al. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*, 2024.
- [20] Brandon Chiou, Mason Choey, Mingkai Deng, Jinyu Hou, Jackie Wang, Ariel Wu, Frank Xu, Zhiting Hu, Hongxia Jin, Li Erran Li, Graham Neubig, Yilin Shen, and Eric P. Xing. Reasoneragent: A fully open source, ready-to-run agent that does research in a web browser and answers your queries, February 2025.
- [21] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. Autowebglm: A large language model-based web navigating agent, 2024.
- [22] 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, 2024.

- [23] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents, 2025.
- [24] Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory, 2024.
- [25] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.
- [26] Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. Web agents with world models: Learning and leveraging environment dynamics in web navigation. *arXiv preprint arXiv:2410.13232*, 2024.
- [27] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games, 2015.
- [28] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, December 2020.
- [29] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization, 2021.
- [30] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control, 2022.
- [31] Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model, 2023.
- [32] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024.
- [33] Yu Gu, Kai Zhang, Yuting Ning, Boyuan Zheng, Boyu Gou, Tianci Xue, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your llm secretly a world model of the internet? model-based planning for web agents, 2025.
- [34] Lisa Feldman Barrett. *How emotions are made: The secret life of the brain*. Pan Macmillan, 2017.
- [35] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An Open Platform for AI Software Developers as Generalist Agents, 2024.
- [36] Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models, 2024.
- [37] Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. Cogagent: A visual language model for gui agents, 2024.
- [38] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. A real-world webagent with planning, long context understanding, and program synthesis, 2024.



- [39] Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. In *International Conference on Learning Representations (ICLR)*, 2018.
- [40] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web, 2023.
- [41] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents, 2023.
- [42] Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, Zeyu Zhang, Yifeng Wang, Qianshuo Ye, Bernard Ghanem, Ping Luo, and Guohao Li. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation, 2025.
- [43] Xun Jiang, Feng Li, Han Zhao, Jiahao Qiu, Jiaying Wang, Jun Shao, Shihao Xu, Shu Zhang, Weiling Chen, Xavier Tang, Yize Chen, Mengyue Wu, Weizhi Ma, Mengdi Wang, and Tianqiao Chen. Long term memory: The foundation of ai self-evolution, 2025.
- [44] Wentao Zhang, Ce Cui, Yilei Zhao, Rui Hu, Yang Liu, Yahui Zhou, and Bo An. Agentorchestra: A hierarchical multi-agent framework for general-purpose task solving, 2025.
- [45] Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang, Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. Magentic-one: A generalist multi-agent system for solving complex tasks, 2024.
- [46] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024.
- [47] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. ‘smolagents’: a smol library to build great agentic systems. <https://github.com/huggingface/smolagents>, 2025.
- [48] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, Xing Zhou, Dongrui Liu, Ling Yang, Yue Wu, Kaixuan Huang, Shilong Liu, Hongru Wang, and Mengdi Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025.
- [49] Eric Xing, Mingkai Deng, Jinyu Hou, and Zhiting Hu. Critiques of world models. *arXiv preprint arXiv:2507.05169*, 2025.
- [50] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [51] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.
- [52] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [53] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.
- [54] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.
- [55] Zhiting Hu and Tianmin Shu. Language models, agent models, and world models: The law for machine reasoning and planning. *arXiv preprint arXiv:2312.05230*, 2023.

- [56] OpenAI. Introducing chatgpt search. <https://openai.com/index/introducing-chatgpt-search/>, 2024. Accessed: 2024-12-19.
- [57] Perplexity. Getting started with perplexity. <https://www.perplexity.ai/hub/blog/getting-started-with-perplexity>, 2024. Accessed: 2024-12-19.
- [58] Yu Gu, Boyuan Zheng, Boyu Gou, Kai Zhang, Cheng Chang, Sanjari Srivastava, Yanan Xie, Peng Qi, Huan Sun, and Yu Su. Is your llm secretly a world model of the internet? model-based planning for web agents. *arXiv preprint arXiv:2411.06559*, 2024.
- [59] Jing Yu Koh, Stephen McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *arXiv preprint arXiv:2407.01476*, 2024.
- [60] Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. Assistantbench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*, 2024.
- [61] Mingkai Deng, Bowen Tan, Zhengzhong Liu, Eric P Xing, and Zhiting Hu. Compression, transduction, and creation: A unified framework for evaluating natural language generation. *arXiv preprint arXiv:2109.06379*, 2021.
- [62] Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522, Singapore, December 2023. Association for Computational Linguistics.
- [63] Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR, 21–27 Jul 2024.
- [64] Andrew Zhu, Alyssa Hwang, Liam Dugan, and Chris Callison-Burch. FanOutQA: A multi-hop, multi-document question answering benchmark for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 18–37, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

## A Details on Web Browsing Environment

### Example Observation

```
URL https://www.google.com/travel/flights
Scroll Position: 0, Window Height: 720, Webpage Height: 3024, Remaining Pixels: 2304, Scrolling
Progress: 23.8%
RootWebArea 'Google Flights - Find Cheap Flight Options & Track Prices'
[149] banner ''
  [160] button 'Main menu', clickable, expanded=False
  [161] image ''
  [168] link 'Google', clickable
  StaticText 'Skip to main content'
  StaticText 'Accessibility feedback'
  [186] navigation ''
    [189] link 'Travel'
    [193] image ''
    [197] link 'Explore'
    [201] image ''
    [205] link 'Flights'
    [209] image ''
    [213] link 'Hotels'
    [217] image ''
    [221] link 'Vacation rentals'
    [225] image ''
  [235] button 'Change appearance', hasPopup='menu', expanded=False
  [240] image ''
  [249] button 'Google apps', clickable, expanded=False
  [250] image ''
  [251] link 'Sign in', clickable
[342] image ''
StaticText 'Flights'
[346] search 'Flight'
  [355] combobox 'Change ticket type. \u200bRound trip', live='polite', relevant='additions
text', hasPopup='listbox', expanded=False, controls='i9'
  [364] image ''
  [399] button '1 passenger, change number of passengers.', hasPopup='dialog'
  [404] image ''
  [406] image ''
  [522] combobox 'Change seating class. \u200bEconomy', live='polite', relevant='additions
text', hasPopup='listbox', expanded=False, controls='i22'
  [529] image ''
  [576] combobox 'Where from?' value='Pittsburgh', clickable, autocomplete='inline',
hasPopup='menu', expanded=False
  [580] image ''
  [628] button 'Swap origin and destination.', disabled=True
  [631] image ''
  [638] combobox 'Where to?', clickable, focused, autocomplete='inline', hasPopup='menu',
expanded=False
  [641] image ''
  generic '', hidden=True
  [690] image ''
  [691] textbox 'Departure', clickable, describedby='i32'
  [712] textbox 'Return', clickable, describedby='i32'
  generic '', hidden=True
  [857] button 'Explore destinations'
[866] Section ''
  [867] heading 'Find cheap flights from Pittsburgh to anywhereMore information on suggested
flights.'
  [871] button 'More information on suggested flights.', hasPopup='menu'
  [873] image ''
  [904] list '', clickable
  [905] listitem ''
    StaticText 'Pittsburgh'
  [907] listitem ''
    [908] button 'Cleveland'
  [909] listitem ''
    [910] button 'Columbus'
  [911] listitem ''
    [912] button 'Akron'
  StaticText 'San Francisco'
  StaticText '$128'
  StaticText 'Jan 9 - Jan 16'
  StaticText '1 stop'
  StaticText ''
  StaticText '10 hr 30 min'
  StaticText 'New York'
  StaticText '$68'
  StaticText 'Dec 7 - Dec 14'
```

## B Prompts for Web Browsing Implementation

### Prompt for Agent Identity

# Name:

Web Browsing Agent

# Description:

An information and automation assistant who responds to user instructions by browsing the internet. The assistant strives to answer each question accurately, thoroughly, efficiently, and politely, and to be forthright when it is impossible to answer the question or carry out the instruction. The assistant will end the task once it sends a message to the user.

# Observation Space:

The text representation and screenshot of the part of webpage visible in the viewport of a browser. Here is an abstract description of the information available in the webpage text representation:

- Identification Information:

- URL: The web address that specifies the location of the webpage.
- Document Properties: Attributes such as scroll position and viewport dimensions that describe the current viewing context.

- Structural Hierarchy:

- Root Element: The primary container for the webpage, indicating its overall theme or purpose.
- Nested Elements: A hierarchy of sections, containers, and components that organize content logically (e.g., headers, footers, sidebars).

- Interactive Components:

- Links: Elements that can be clicked to navigate to other pages or sections, often labeled descriptively.
- Buttons: Interactive controls that trigger actions (e.g., submitting forms, opening menus).

- Content Types:

- Text: Main content, headings, and subheadings that provide information and context.
- Images and Media: Visual elements that enhance the understanding or appeal of the content.
- Forms and Inputs: Fields for user input, including text boxes, dropdowns, and checkboxes.

- Functional Areas:

- Navigation Menus: Organized sets of links that allow users to explore different sections of the site.
- Search Interface: Components that enable users to search for content within the site, including input fields and associated buttons.

- State Information:

- Visibility and Expand/Collapse States: Indicators showing whether certain elements are active, visible, or in a collapsed state, impacting user interaction.
- Focus States: Information on which elements are currently focused, important for keyboard navigation and accessibility.

## Prompt for Agent Identity (Continued)

### - Accessibility Features:

- Role and Description Information: Metadata that provides context about the purpose of elements, useful for screen readers and assistive technologies.

### - General User Considerations:

- Navigation: Recognizing how to traverse the webpage using links and buttons.
- Interactivity: Understanding how to engage with forms, search fields, and dynamic components.
- Content Engagement: Identifying and interpreting various content types to glean necessary information.

### # Action Space:

13 different types of actions are available.

`noop(wait_ms: float = 1000)`

Examples:

`noop()`

`noop(500)`

`send_msg_to_user(text: str)`

Examples:

`send_msg_to_user('Based on the results of my search, the city was built in 1751.')`

`scroll(delta_x: float, delta_y: float)`

Examples:

`scroll(0, 200)`

`scroll(-50.2, -100.5)`

`fill(bid: str, value: str)`

Examples:

`fill('237', 'example value')`

`fill('45', 'multi-line\nexample')`

`fill('a12', 'example with "quotes"')`

`select_option(bid: str, options: str | list[str])`

Examples:

`select_option('a48', 'blue')`

`select_option('c48', ['red', 'green', 'blue'])`

`click(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`

Examples:

`click('a51')`

`click('b22', button='right')`

`click('48', button='middle', modifiers=['Shift'])`

`dblclick(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`

Examples:

`dblclick('12')`

`dblclick('ca42', button='right')`

`dblclick('178', button='middle', modifiers=['Shift'])`

### Prompt for Agent Identity (Continued)

hover(bid: str)

Examples:

hover('b8')

press(bid: str, key\_comb: str)

Examples:

press('88', 'Backspace')

press('a26', 'Control+a')

press('a61', 'Meta+Shift+t')

focus(bid: str)

Examples:

focus('b455')

clear(bid: str)

Examples:

clear('996')

drag\_and\_drop(from\_bid: str, to\_bid: str)

Examples:

drag\_and\_drop('56', '498')

upload\_file(bid: str, file: str | list[str])

Examples:

upload\_file('572', 'my\_receipt.pdf')

upload\_file('63', ['/home/bob/Documents/image.jpg', '/home/bob/Documents/-file.zip'])

Only a single action can be provided at once. Example: fill('a12', 'example with "quotes"')

# Instruction: {user\_instruction}

# Current Date and Time: {current\_datetime}

### Prompt for Encoder

# Observation:  
{observation}

# State:

Describe all the elements in the current webpage observation. Note any dialogs, progress indicators, or error messages. Include any interactive elements and their values or if they are blank. Note any detailed information such as facts, entities, or data that are relevant to the task. Report any error messages like whether the last action was correct. Try to be as comprehensive and detailed as possible.

Wrap your response in the tag <state> and </state>.

### Prompt for Policy

{memory}

# Current State:  
{state}

# Intent:

Describe the action the assistant should take next to carry out the user's instruction. Avoid using phrases such as "To accomplish the goal," "I will," "To proceed.". Avoid ending with phrases like "to execute the search." Describe one action at a time and avoid combining multiple steps. Refrain from mentioning specific element IDs as they may change during execution. Limit your response to one phrase and include any details that help select the correct action. Be creative and propose novel methods to achieve the goal. Avoid creating accounts without user permission or providing personal information. Concrete example would be "Go to the home page of Google Flights." and "Click on the 'Search' button."

Wrap your response in the following format:

<think>

Your thoughts and reasoning process

</think>

<intent>

Description of the action to perform next

</intent>

### Prompt for World Model

{memory}

# Current State:  
{state}

# Memory Update:  
{memory\_update}

# Action Intent:  
{plan}

# Next State:

Describe all the elements in the webpage after the agent attempts to carry out the intent. Note that the execution may not be successful, so you will have to infer the result of the action. Note any dialogs, progress indicators, or error messages. Include any interactive elements and their values or if they are blank. Note any detailed information such as facts, entities, or data that are relevant to the task. Report any error messages displayed. Try to be as comprehensive and detailed as possible.

Wrap your response in the following format:

<next\_state>

Follow the format of the current state description. Use present tense. Avoid starting phrases like "Based on the interaction history, current state, and current intent".

</next\_state>

### Prompt for Critic

{memory}

# Final State:  
{state}

# Task Success and Progress:

Your task is to evaluate the performance of the agent. Given the agent's instruction, interaction history, the final state of the webpage, and the agent's responses to the user if any, your goal is to decide whether the agent's execution is successful or not. If the current state is a failure but it looks like the agent is on the right track towards success, you should also output as such.

Wrap your response in the following format:

<think>  
Your thoughts and reasoning process  
</think>

<status>  
"success" or "failure"  
</status>

<on\_the\_right\_track>  
"yes" or "no"  
</on\_the\_right\_track>

### Prompt for Memory Update

{memory}

# State:  
{state}

# Action Intent:  
{plan}

# Memory Update:

Summarize the changes in the webpage observation that should be remembered for achieving your goal and for predicting the next state. Note any new elements, any elements no longer visible, or any changes in the content of existing elements. Also note if there is no change. Include any other inferred information that may help you decide the next action, such as whether an action intent is successful, or whether progress has been made or reversed. Do not include your next planned actions. Revise your belief from previous history if the current state contradicts it.

Wrap your response in the tag <memory\_update> and </memory\_update>.



### Prompt for Actor

{memory}

# Observation:  
{observation}

# Current State:  
{state}

# Current Intent:  
{plan}

# Action:

Choose an API call that will carry out the intent when executed in the webpage. Use only one action at a time. You must not enclose bid inputs in [brackets] but instead in 'single quotes'. Interact only with elements in the current step observation. Your response will be executed as a Python function call, so ensure it adheres to the format and argument data type specifications defined in the action space.

Wrap your response in the tag <action> and </action>.

### Prompt for Action Clustering

Here is the action space for a browser agent to navigate in a webpage:

16 different types of actions are available:

`noop(wait_ms: float = 1000)`

`send_msg_to_user(text: str)`

`scroll(delta_x: float, delta_y: float)`

`fill(bid: str, value: str)`

`select_option(bid: str, options: str | list[str])`

`click(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`

`dblclick(bid: str, button: Literal['left', 'middle', 'right'] = 'left', modifiers: list[typing.Literal['Alt', 'Control', 'Meta', 'Shift']] = [])`

`hover(bid: str)`

`press(bid: str, key_comb: str)`

`focus(bid: str)`

`clear(bid: str)`

`drag_and_drop(from_bid: str, to_bid: str)`

`upload_file(bid: str, file: str | list[str])`

`go_back()`

`go_forward()`

`goto(url: str)`

Only a single action can be provided at once. Example: `fill('a12', 'example with "quotes"')`

Below, you will find lists of intents, or natural language descriptions of actions that, when executed, will translate to one of the function calls above. The intents will be provided in the following JSON format:

```
‘‘‘json
{
  "intent_id": "intent description"
}
‘‘‘
```

Your task is to cluster list of intents into semantically equivalent groups, where each group represents intents that lead to the same action when executed (i.e., navigating to the Google homepage is translated to `goto('https://www.google.com')`) and would therefore correspond to the same API call in a Playwright browser. Intents that use different wording but convey the same action should be grouped together. Try to minimize the number of clusters.

### Prompt for Action Clustering (Continued)

Represent the clustering results using a JSON object where each cluster has a unique identifier, and each identifier maps to a list of actions in that cluster. See below for an abstract example:

```
'''json
{
  "cluster_id": {
    "intent": "representative intent name for this cluster",
    "candidates": [
      "<list of intent ids that belong to this cluster>"
    ]
  }
}
'''
```

Concrete Example 1:

Dictionary of Intents:

```
'''json
{
  "0": "Navigate to the Google homepage by entering its URL.",
  "1": "Go to the Google homepage.",
  "2": "Go to the Google homepage",
  "3": "Go to the Google homepage by navigating to
        'https://www.google.com'",
  "4": "Go to the home page of Google"
}
'''
```

["Navigate to the Google homepage by entering its URL.", "Go to the Google homepage.", "Go to the Google homepage", "Go to the Google homepage by navigating to https://www.google.com", "Go to the home page of Google"]

Clustering Results:

```
'''json
{
  "cluster_1": {
    "intent": "Navigate to the Google homepage",
    "candidates": [0, 1, 2, 3, 4]
  }
}
'''
```

Concrete Example 2:

Dictionary of Intents:

{action\_candidate\_json}

Clustering Results:

## B.1 Adaptation for WebArena Evaluation

### Agent Description for WebArena Evaluation

An information and automation assistant that interacts with the browser and responds to user instructions. The response follows the following rules: 1. When the intent is a question, and a complete answer to the question has been found, then send the answer to the user; 2. the intent wants to locate specific information or navigate to a particular section of a site, and the current page satisfies, then stop and tell the user you found the required information; 3. the intent want to conduct an operation, and has been done, then stop and tell the user the operation has been completed.

The assistant should try to achieve the goal in the current site without navigating to sites like Google. Be forthright when it is impossible to answer the question or carry out the task. The assistant will end the task once it sends a message to the user.

## C Prompts for Generating and Evaluating on the FlightQA Dataset

### Prompt for Generating Initial Constraints and Questions

#### **System:**

You are a creative writer who is an expert at crafting questions to help train assistants who answer user queries. Current date and time: {current\_datetime}

#### **Instruction:**

Your task is to create a robust benchmark for evaluating an AI's ability to search for flights through a platform like Google Flights. To ensure the dataset effectively represents real-world use cases. Here are some important factors to consider:

##### 1. Diversity of Queries

- Range of Destinations: Include both common and obscure destinations to test how well the model handles varying levels of demand.
- Dates and Duration: Include different date ranges, including last-minute flights, peak travel dates (like holidays), and extended trips. Ensure there's a variety in trip duration as well.
- Passenger Variability: Include solo travelers, families, and group travel (e.g., one adult vs. two adults and two children) since these change the search parameters and price results.
- Class and Preference: Vary preferences like cabin class (economy, business, first) and filters (non-stop, one stop, preferred airlines, etc.).
- Budget Constraints: Test price sensitivity by setting different budget limits to see how well the AI handles trade-offs.

##### 2. Complexity of Requirements

- Multi-Leg Flights: Add queries for multi-city trips or those requiring complex layovers.
- Dynamic Constraints: Include queries with dynamic constraints, such as "find the cheapest flight but depart between 8-10 AM," to see if the model can adapt its search within specific time frames.
- Conditional Preferences: Test cases where users might want options based on multiple conditions, like "either find the cheapest non-stop or the shortest two-stop option."

In practice, the questions typically vary in the following dimensions:

- Ticket type (round-trip, one-way, etc.)
- Routes (origin and destination)
- Layover location(s)
- Dates (departure and/or return)
- Flight time (departure and arrival)
- Total flight time
- Airlines
- Cabin class (economy, business, etc.)
- Aircraft
- Eco-friendly options (CO2 Emissions)

Given a number of constraints, you should first provide a list of constraints, with the number of constraints equal to the specification. After that, you will generate a question a typical user will ask which imposes those constraints. You should repeat this for at least 7 times to generate a set of questions with simple language. Make sure that the number of constraints in the question matches the number of constraints specified.

Do not include constraints about the number of passengers. If the constraint is a date, you can use relative dates (e.g., "tomorrow", "next month", "after 8 PM", etc.). Avoid using specific dates like "December 25th" to ensure the questions are relevant throughout the year.

Your response should follow the JSON format below:

### Prompt for Generating Initial Constraints and Questions (Continued)

Number of Constraints: <num\_constraints>

```
{
  "num_constraints": <num_constraints>,
  "questions": [
    {
      "constraints": [<constraints>],
      "question": <question>
    },
    ...
  ]
}
```

Below is a concrete example:

Number of Constraints: 3

```
{
  "num_constraints": 3,
  "questions": [
    {
      "constraints": ["one-way", "New York to London",
        "departing next Friday"],
      "question": "Can you find a one-way flight from New York
        to London departing next Friday?"
    },
    ...
  ]
}
```

### Prompt for Iteratively Expanding Constraints and Questions

**System:**

[Same as above]

**Instruction:**

[Same as above until "Your response should follow"]

Your response should follow the JSON format below:

Maximum number of constraints: <max\_constraints>

Starting constraints and questions:

```
{
  "num_constraints": <num_constraints>,
  "constraints": [<constraints>],
  "question": <question>
}
```

Questions with increasing complexity:

```
{
  "questions": [
    {
      "num_constraints": <starting num_constraints + 1>,
      "constraints": [<previous constraints with 1 additional>],
      "question": <question>
    },
    {
      "num_constraints": <starting num_constraints + 2>,
      "constraints": [<previous constraints with 2 additional>],
      "question": <question>
    },
    ... (continue until reaching the maximum number of constraints)
  ]
}
```

Your Response:

Maximum number of constraints: {max\_num\_constraints}

Starting constraints and questions:

{starting\_constraint\_questions}

Questions with increasing complexity:

### Prompt for Evaluation

# Interaction Date and Time:

{interaction\_datetime}

# Interaction History:

[Concatenation of observations from all steps]

Above are the webpages an assistant interacted with while trying to answer the user's query.

The user is looking for flights with the following constraints:

{constraints}

Here is the exact query provided by the user:

{goal}

Here is the assistant's response:

{message}

Your task is to evaluate two aspects of the response:

- 1) Whether the assistant's response is supported by the interaction history, and
- 2) Whether the assistant's response satisfies the user constraints to the extent allowed by the results.

Some Context:

- To determine the seating class of a flight being returned, refer to the value of the "Change seating class" combobox.
- It is not always possible to satisfy all the user constraints. In this case, examine whether the response is as close to the user constraints as possible.

Answer in the following format:

<think>

Your thoughts and reasoning.

</think>

<grounding>

Your assessment of whether the response is supported by the interaction history. Answer "yes" or "no"

</grounding>

<relevance>

Your assessment of whether the response satisfies the user constraints to the extent allowed by the results. Answer "yes" or "no"

</relevance>