Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan,

Aditya Vempaty, Ravi Kokku

{tea, deepak, prasenjit, ashish, aditya, ravi}@emergence.ai

Emergence AI

July, 2024

Abstract

AI Agents are changing the way work gets done, both in consumer and enterprise domains. However, the design patterns and architectures to build highly capable agents or multi-agent systems are still developing, and the understanding of the implication of various design choices and algorithms is still evolving. In this paper, we present our work on building a novel web agent, Agent-E [1]. Agent-E introduces numerous architectural improvements over prior state-of-the-art web agents such as hierarchical architecture, flexible DOM distillation and denoising method, and the concept of *change observation* to guide the agent towards more accurate performance. We first present the results of an evaluation of Agent-E on WebVoyager benchmark dataset and show that Agent-E beats other SOTA text and multi-modal web agents on this benchmark in most categories by 10-30%. We then synthesize our learnings from the development of Agent-E into general design principles for developing agentic systems. These include the use of domain-specific primitive skills, the importance of distillation and de-noising of environmental observations, the advantages of a hierarchical architecture, and the role of agentic self-improvement to enhance agent efficiency and efficacy as the agent gathers experience.

1 Introduction

Recent advancements in large language models have led to significant interest in the development of autonomous agents that can execute complex tasks on the web [Zheng et al., 2024, He et al., 2024, Lutz et al., 2024] and on device [Bai et al., 2024, Wen et al., 2024]. Automation of complex and repetitive tasks presents an invaluable opportunity to increase individual and organizational efficiency. In addition, robust automation systems will unlock new use cases and experiences by enabling collaboration between users and AI agents working together to accomplish sophisticated tasks.

Figure 1 shows a simplified anatomy of a web agent. A typical web-agent, at its most basic, has two key capabilities: the capability to sense the web page and the capability to act on the web page.

- Sensing: Sensing the state of the web-page, for a web agent, typically involves encoding the Document Object Model (DOM) of the page [Nakano et al., 2022, Lutz et al., 2024], using the accessibility tree of the page [He et al., 2024] and/or using screenshots of the page [He et al., 2024].

- Acting: The action space can be comprised of simple actions such as navigating to URLs, clicking on elements, and entering text in a field, or composite actions comprising of several simple actions. An example of composite action is used by Lutz et al. [Lutz et al., 2024], which they call 'Input'. 'Input' selects a text box, deletes any existing content, inputs text and presses submit button.

Based on the user's task and the current state of the page, the agent can create a plan i.e., it determines the subsequent next action or series of actions required to complete the task.

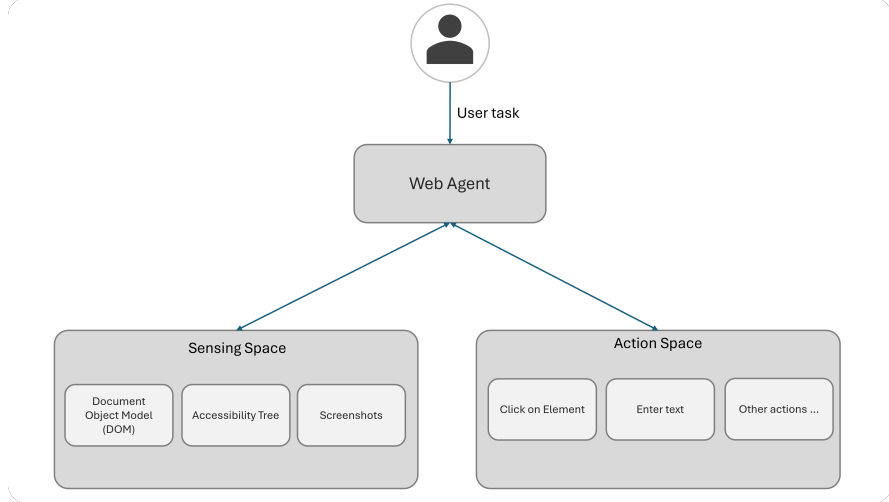[1] Our code is available at https://github.com/EmergenceAI/Agent-E

Figure 1: Simplified anatomy of web agents

Developing a robust web agent that can autonomously perform tasks on the browser presents multiple challenges. Firstly, HyperText Markup Language (HTML) Document Object Models (DOMs) can be noisy and expansive, so that they often exceed an LLM's context window. This renders them unusable without intelligent simplification or de-noising steps. Further, even with simplification, information may exceed the context window limit of modern LLMs after a few interactions. This is an important challenge that restricts their use for complex tasks [He et al., 2024]. Secondly, websites are primarily designed for human visual consumption, wherein information is organized to prevent information overload for a user; thus, websites utilize established human-computer interaction patterns that may not be optimal for agent interaction. Complex widgets, such as date selectors, are easy for humans to operate but pose difficulties for agents [Lutz et al., 2024]. Thirdly, while humans can naturally execute complex web tasks (e.g., finding the cheapest flight between two destinations), agents require detailed multi-step planning to execute such tasks. Further, the plan may need multiple revisions if the initial approach proves unfruitful.

Despite these challenges, recent work has demonstrated promising outcomes with web agents executing diverse tasks on the internet. Yet, state-of-the-art web agents leave much to be desired in terms of practical usefulness [Wornow et al., 2024]. Most previous studies simply report success rates. Task success rates show that web agents are more error-prone compared to a human performing the same task and are not yet ready for effective mainstream use [Lutz et al., 2024, He et al., 2024, Zhou et al., 2023].

Task success rates, while important, may not tell the whole story when it comes to evaluating web-agents. For example Kapoor et al. [Kapoor et al., 2024] note that since that the underlying LLMs are stochastic systems, simple retry mechanisms often improve success rates. Thus, simply measuring task success rates is misleading. There are other important factors that influence the utility and user experience of using agentic systems, such as:

- Task completion times: The duration required for the system to complete a task, which impacts overall efficiency and productivity. Task completion times are especially relevant for online tasks or tasks with a human in the loop. The task completion times can be influenced by extrinsic factors (network speed, computing environment influencing loading times of web pages) or intrinsic factors (token generation speed of the LLMs, the path taken by the agent to perform the task, errors made during execution etc). Despite the variability, task completion times is an important measure since it can be critical in many real world applications.

- Cost: The total expenditure involved in executing the task. The cost is a variable measure and depends on the underlying model and the per token input/output cost of different model providers. This is also a 'point in time' measure, since the pricing models set by LLM providers change frequently. While not fully encompassing, a good proxy for cost is number of LLM calls involved.

- Error-awareness: The capability of the system to recognize and report its own errors or failures, ensuring reliability and ability to provide a seamless user experience by highlighting and recommending fallback options. Ability of system to detect that it has not completed the task appropriately will also open up avenues to collect human demonstration (e.g. *I could not complete this task, can you demonstrate how you would do it?*) and a path towards continuous improvement.

In this paper, we introduce Agent-E, a state-of-the-art web agent capable of performing complex web-based tasks. Central to Agent-E are two LLM-powered components: the planner agent and the browser navigation agent. The planner agent is responsible for task planning and task management. It breaks down the user task into a sequence of sub tasks and delegates them one at a time to the browser navigation agent. The browser navigation agent is tasked with executing the individual sub tasks by sensing the page using different DOM distillation capabilities available to it, finding the next actions to execute and reporting its task success or failure back to the planner. This loop is executed iteratively until the task is completed. This tiered architecture ensures that the planner agent is insulated from the overwhelming and noisy details of the website and DOM, and the browser navigation agent is freed from the complexities of the overall task planning and orchestration.

Agent-E can run in two modes: autonomous mode and human-in-the-loop mode. In the autonomous mode, Agent-E performs the task end to end as best as it can, and in the human-in-the-loop mode, it falls back to the user when it encounters steps that it cannot accomplish (e.g. logging to a web page, solve a captcha triggered by the website) or to ask for clarifications when the task itself is ambiguous. We believe human-in-the-loop workflows are critical for adoption of agentic systems. However, since there are no web automation benchmarks for human-in-the-loop agents [Kapoor et al., 2024], in this paper we will focus on Agent-E as an autonomous web agent.

We evaluate Agent-E in autonomous mode on the WebVoyager benchmark [He et al., 2024], where we achieve a new state-of-the-art result of 73.2%, 20% higher than previous state-of-the-art for text-only [Lutz et al., 2024] and 16% higher than previous state-of-the-art multi-modal web agent [He et al., 2024]. For specific sites like Wolfram Alpha, it achieves upto 30% improvement.

1.1 Contributions

- We introduce a novel hierarchical architecture for web agents that enables the execution of more complex tasks through a clear separation of roles and responsibilities between a planner agent and a browser navigation agent. We illustrate the utility of the hierarchical architecture for capabilities such as task verification, backtracking, and error recovery, with examples where these capabilities are invoked.

- We propose a flexible DOM distillation approach whereby, given a task, the browser navigation agent can choose the most suitable DOM representation from three different implementations. We demonstrate its effectiveness with examples.

- We highlight the benefits of *'Change observation'*, a paradigm conceptually similar to Reflexion [Shinn et al., 2024]. After the execution of each action, the outcome (change in state) is monitored and used to provide verbal feedback to the browser navigation agent. This guides the agent towards better awareness of the current state and more accurate performance.

- We report detailed end-to-end evaluations of Agent-E on the WebVoyager benchmark and show that it achieves a new state-of-the-art results with a 73.2% success rate. This marks a 20% improvement over the previous text-only web agent and a 16% increase over the former state-of-the-art multi-modal web agent.

- Beyond task success rates, we are the first to report additional metrics such as error awareness, task completion times, and the number of LLM calls, offering a baseline for comprehensive evaluation of future web agents.

- Agent-E is a result of numerous experimentation and exploration into the space of AI Agents. We synthesize our learning from the development of Agent-E into design principles for development of agentic systems that we believe generalize beyond Agent-E and web automation, and provide useful insights to practitioners.

2 Agent-E: System Description

Figure 2 shows the high level architecture of Agent-E. It comprises of two LLM-powered agents: Planner Agent and Browser Navigation Agent, and two executor components: Planner Skills executor and Browser Navigation Skills executor. Each LLM-powered agent has skills associated with it, which are python functions that are described to the LLM for function calling. We do not make any distinction between skills associated with sensing (e.g. *Get DOM*) and skills associated with acting on the page (e.g. *click on element*). The executor components execute the function suggested by the LLM and relays the response back to the LLM.

Agent-E is built using Autogen, the open-source programming framework for building multi-agent collaborative systems [Wu et al., 2023b] and Playwright[2] for browser control. Its architecture leverages the interplay between skills and agents as shown in Figure 2.

Figure 3 shows a conceptual flow diagram of Agent-E. Given a new user task, the planner decomposes the task into a sequence of steps that need to be executed. The next step is then delegated to the browser navigation agent for execution. Browser navigation agent is implemented as a nested chat that is freshly instantiated by the Planner for each run (i.e., it does not contain previous steps in its chat history). Browser navigation agent has a set of predefined 'primitive' or foundational skills for observing denoised browser state and controlling the browser instance using Playwright. Figure 4 shows the skills available to the browser navigation agent. The browser navigation agent uses the skills available to it to perform the sub task and return a summary of actions it took to perform the task and/or answer the planner if the task was a question. Figure 5 shows an example communication between planner and the browser navigation agent and Figure 6 shows the browser navigation agent using the primitive skills available to it to perform a related sub task.
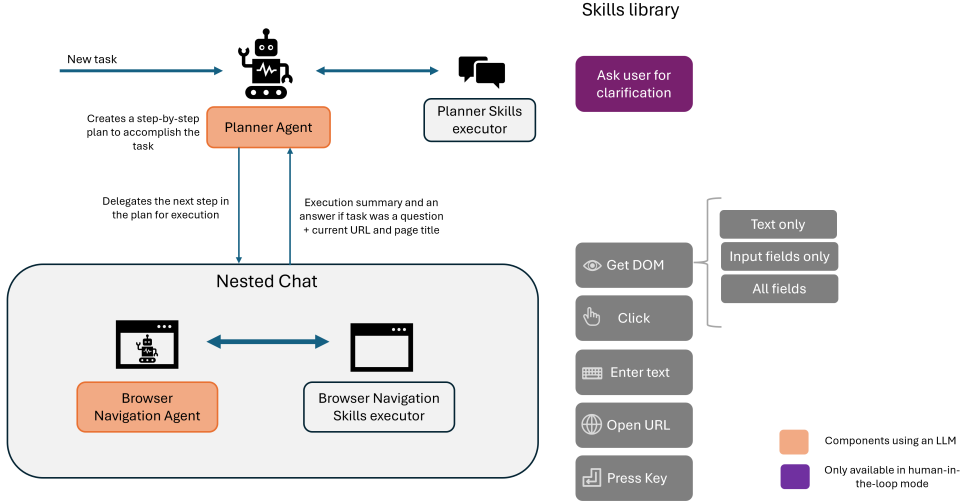

Figure 2: A high level architecture of Agent-E

2.1 Skills Design for Browser Navigation Agent

There are two key differences between Agent-E and previous web agent implementations such as Wilbur [Lutz et al., 2024] and WebVoyager [He et al., 2024] in terms of skills .

- Sensing Skills: Agent-E supports multiple DOM synthesis techniques that allows the browser navigation agent to choose the approach best suited for the task (see Figure 4). If the task is to summarise information on a page, it can simply use Get DOM with *text_only* content type. If the task is to identify and execute a search on a page, it can use the content type *input_fields*. If the task is to list all the interactive elements on a page, it can use *all_fields*. This
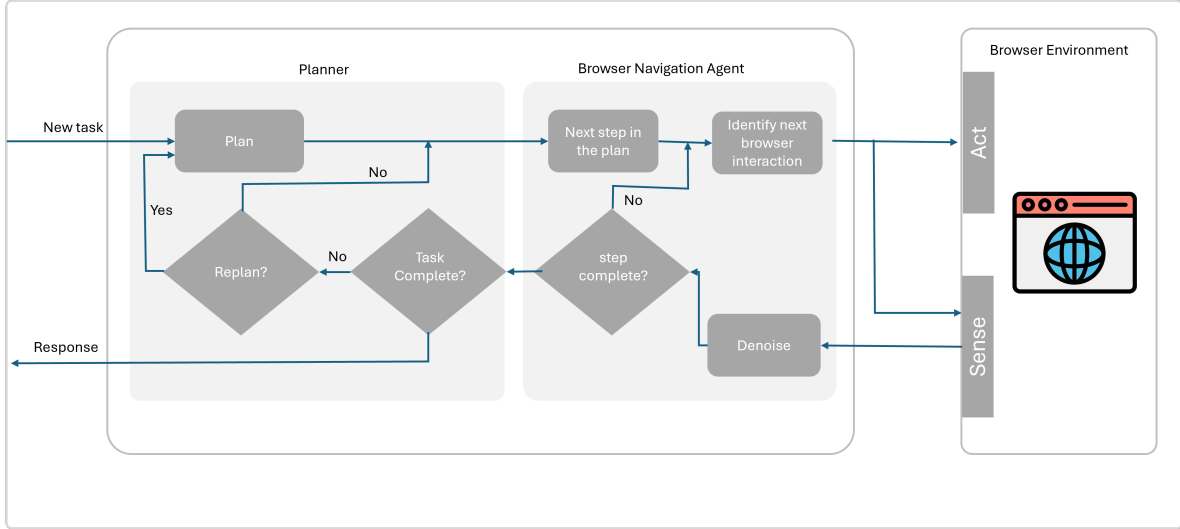
---

[2]https://playwright.dev/

Figure 3: Conceptual flow diagram of Agent-E. Individual blocks represent functions; In Agent-E, a single LLM call is used to perform multiple functions.

Skills	Input parameter	*Change Observation* during skill execution	Return
Get DOM	content type: text_only	None	Returns the innertext of body element of HTML DOM with some post processing. Ideal for text summarization and information extraction.
	Content type: input fields	None	Returns a json representation of specific HTML elements such as buttons, input fields and links in a page. Ideal for interacting with search fields or buttons.
	Content type: all fields	None	Returns a json representing the full page. Most complete representation of all elements, also most lengthy and noisy.
Click	Selector: identifier of the element to be clicked	Observe for DOM change events immediately following the click.	Returns a textual response indicating if click was performed and a summary of changes observed (if any).
Enter text	Selector: identifier of the element to enter text.	Observe for DOM change events during or following the text input.	Returns a textual response indicating if text input was performed and a summary of changes observed (if any).
Open URL	URL: The url to navigate	Web navigation	Page url and title of the new page
Press Key	Keys to press. (e.g. Submit, PageDown)	Observe for DOM change events immediately following the key press.	Returns a textual response indicating if keypress was performed and a summary of changes observed (if any).

Figure 4: Skills registered to the Browser Navigation Agent for sensing and acting on the web page.

optimizes the information available to the agent and prevents the problems associated with noisy DOM. Another key difference is our DOM de-noising techniques for *all_fields* and *input_fields* attempts to preserve the parent child relationship of elements wherever possible and relevant. This is unlike some of the other previous implementations which uses a flat DOM encoding (e.g. [Lutz et al., 2024]). Further, to make identifying and interacting with HTML elements easier, Agent-E injects a custom identifier attribute (*mmid*) to each element as part of sensing, similar to [Zhou et al., 2023] and [He et al., 2024].

- Action Skills: All the action skills are designed to not only execute an action but also report on any change in state as an outcome of the action, a concept we call *'Change observation'*. This is conceptually similar to Reflexion paradigm [Shinn et al., 2024] which uses verbal reinforcement to help agents learn from prior failings. However, a key difference is that change observation is not directly associated or limited to a prior failure. The observation returned can be any type of outcome of the action. For example, a click action may return a response *Clicked the element with mmid 25. As a consequence, a popup has appeared with following elements.* Such verbal responses nudges the agent towards the correct next step.

Change Observation capability is implemented by observing changes in selected attributes of
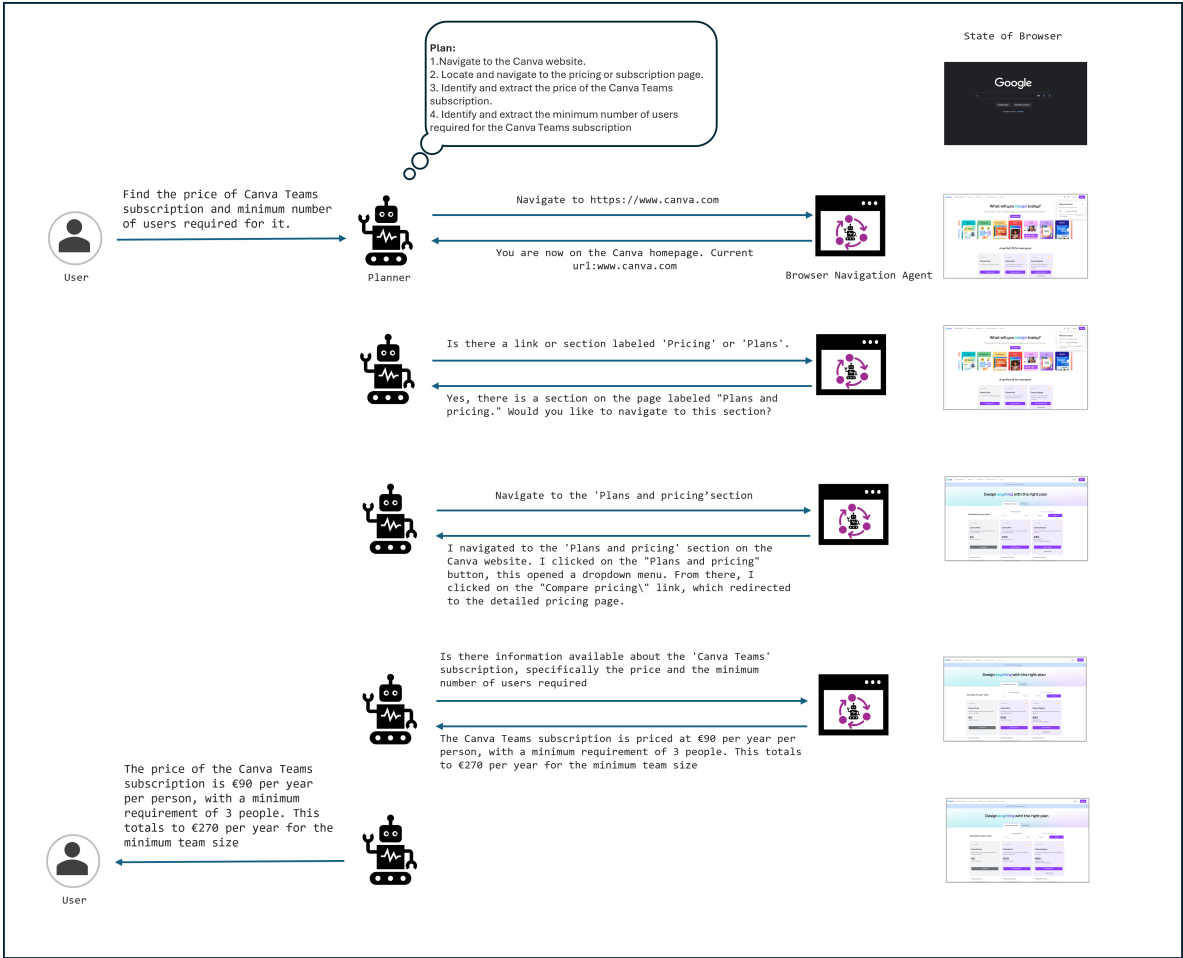
Figure 5: An example of Agent-E execution highlighting communication between the planner and browser navigation agent for the user task *Find the price of Canva Teams subscription and minimum number of users required for it*

elements (e.g *aria-expanded*) and Mutation Observer Web API [3] that allow observing changes in DOM immediately following an action execution. This is especially useful for extremely dynamic pages such as a flight booking website and nudges the LLM to take appropriate next step given the task. (see Figure 6 for an example)

3 Evaluation

3.1 Introduction to WebVoyager

WebVoyager [He et al., 2024] is a recent web agent benchmark that consists of web navigation tasks across 15 real websites as shown Figure 7. Each website has about 40-46 tasks resulting in a benchmark dataset of 643 tasks (see Table 1 for example tasks). These tasks could be completed through DOM manipulation (textual) as well as augmenting with image understanding (multi-modal). We chose WebVoyager since it covers a diverse range of tasks across different live, dynamic and representative websites. Other alternative benchmarks either focused solely on single task domain [Yao et al., 2022], used custom created websites that are significantly simpler than real-world versions in terms of DOM complexity [Zhou et al., 2023], or used cached versions of real-websites only supporting a fixed route for the agent and not allowing free form exploration [Deng et al., 2024]. One of the issues with WebVoyager though is that it uses static dates for some of the tasks (e.g. *Search a hotel with free WiFi and*
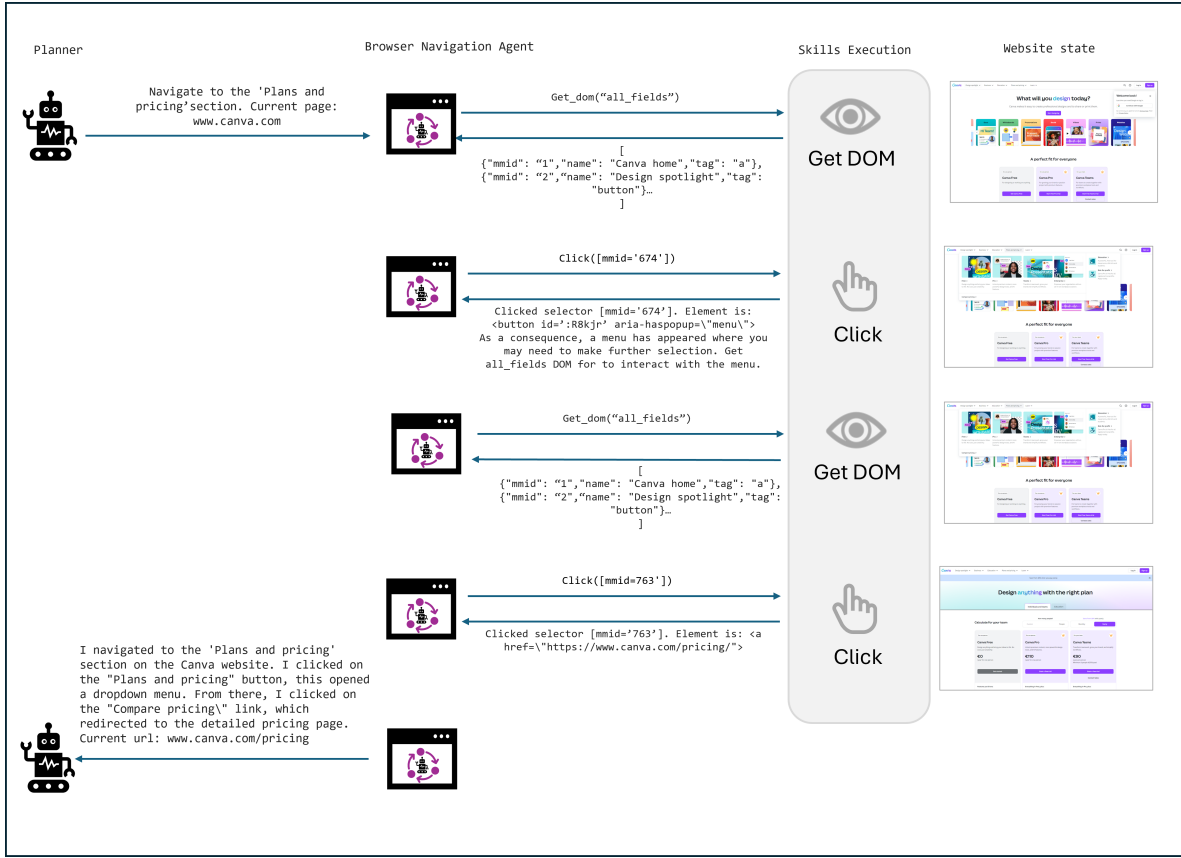
---

[3]https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

Figure 6: An example of Agent-E nested chat execution loop for the sub task *"Navigate to the plans and pricing section"* which is part of the larger task introduced earlier *"Find the price of Canva Teams subscription and minimum number of users required for it"*

air conditioning in Bali from Jan 1 to Jan 4, 2024.). The static dates makes some of these tasks unachievable since the dates have passed. We manually went through the tasks to identify ones with fixed dates in the tasks that make the task unachievable, and added 8 months to the dates (i.e., *Search a hotel with free WiFi and air conditioning in Bali from Aug 1 to Aug 4, 2024.*).

Figure 7: WebVoyager Websites

Website	Task
Allrecipes	Find a recipe for Baked Salmon that takes less than 30 minutes to prepare and has at least 4-star rating based on user reviews.
Booking	Find the cheapest available hotel room for a three-night stay from 1st Jan in Jakarta. The room is for 2 adults, just answer the cheapest hotel room and the price.
Github	Search for an open-source project related to 'climate change data visualization' on Github and the report the project with the most stars.
Google Flights	Search a one-way flight from Dublin to Athens Greece for 1 Adult that leaves on December 30 and analyze the price graph for the next 2 months.

Table 1: Example tasks from WebVoyager benchmark.

3.2 Evaluation Setup

The entire benchmark was divided among 5 human evaluators who ran 125-130 tasks each. For each task, the evaluators were instructed to classify the task as *pass* or *fail* along with a textual reason in case of failures. They were instructed to mark the task as pass if it was completed successfully in full (in case the task has multiple parts). The evaluators ran the task from India during IST office hours (this may have implications when you interpret or compare measures such as task completion times). For the evaluation, Agent-E was used in autonomous mode (i.e., *ask user for input* skill was disabled) and used GPT-4-Turbo as the LLM for both planner and browser navigation agent.

3.3 Measures

We report four important measures that are relevant for comprehensive evaluation of web agent and understanding their practical implementation readiness.

- Task success rates: The percentage of tasks that Agent-E performed successfully across websites.

- Self-aware vs Oblivious failure rates: Detecting when the task was not completed successfully is of utmost importance, since it can be used for enabling a fallback workflow, to notify the user of failure or use as an avenue to gather human demonstration for the same task. Self-aware failures are failures where agent is aware of its own failure in completing the task and responds with a final message explicitly stating so, e.g. *I'm unable to provide a description of the first picture due to limitations in accessing or analyzing visual content.* or *'Due to repeated rate limit errors on GitHub while attempting to refine the search...'*. The failures could be due to technical reasons or agent deeming the task unachievable since it could not complete the task after repeated attempts. On the other hand, oblivious failures are cases were the agent wrongly answers the question or performs the wrong action (e.g. adds the wrong product to cart or provides a wrong information). For mainstream utility, oblivious failures should be as minimal as possible. For the current evaluation, failures were categorized to self-aware and oblivious failures by manual annotation. However, it would be trivial to employ an LLM critique to automatically do the same task, similar to [Wornow et al., 2024].

- Task completion times: The average time required to complete the task, across websites for failed and successful tasks.

- Total number of LLM calls: The average number of LLM calls (both planner and browser navigation agent) that was required to perform the task. This includes both successful and failure cases.

3.4 Result: Quantitative

In this section, we will present quantitative results of how well Agent-E performs in WebVoyager benchmark. Tables 2 and 3 shows the summary of evaluation of Agent-E on WebVoyager.

Publication	Task success rates on websites							
	Allrecipe	Amazon	Apple	Arxiv	Github	Booking	ESPN	Coursera
He et al. 2024 (text)	57.8	43.1	36.4	50.4	63.4	2.3	28.6	24.6
He et al. 2024 (multi)	51.1	52.9	62.8	52.0	59.3	32.6	47.0	57.9
Lutz et al. 2024 (text)	60	43.9	60.5	51.2	22.0	**38.6**	59.1	51.1
Agent-E (text)	**71.1**	**70.7**	**74.4**	**62.8**	**82.9**	27.3	**77.3**	**85.7**
Failure modes	Agent-E Error Analysis on Websites							
Overall failures %	28.9	29.3	25.6	37.2	17.1	72.7	22.7	14.3
Self-aware failures %	17.8	14.6	9.3	18.6	12.2	4.5	13.6	4.8
Oblivious failures %	11.1	14.6	16.3	18.6	4.9	68.2	9.1	9.5
TCT	Agent-E Avg. Task Completion Times (seconds)							
TCT (Success)	116	286	122	137	104	183	187	119
TCT (Failed)	196	246	200	176	384	317	387	177
LLM Calls	Agent-E Avg. Number of LLM calls							
Total	22	23.1	21.5	25.5	21.5	36.4	24.0	25.5
Planner	6.5	6.4	5.9	6.9	5.4	6.6	6.3	6.3
Browser Nav Agent	15.5	16.7	15.6	18.6	16.1	29.8	17.7	19.2

Table 2: Evaluation of Agent-E on WebVoyager.

Comparison with prior state-of-the-art agents

Agent-E overall successfully completed 73% of the tasks and outperforms prior state-of-the-art text-only web agent WILBUR [Lutz et al., 2024] by 21% and state-of-the-art multi-modal web agent [He et al., 2024] by 16%. The effectiveness of Agent-E varied across websites from 27.3% (Booking) to 95.7% (WolframAlpha). Comparing by websites, Agent-E outperformed both Wilbur [Lutz et al., 2024] and WebVoyager multi-modal agent [He et al., 2024] in 11 out of 15 websites. Booking.com and Google Flights continue to be websites, where all web agents, including Agent-E does not do well.

It is important to note that [Lutz et al., 2024] uses task and website-specific prompting and [He et al., 2024] uses vision for observing the page. In contrast, Agent-E is a generic text-only web agent which does not employ any task or website-specific instructions. This suggests that there is likely room for further improvement in Agent-E using some of these strategies.

Agent-E failure modes

Overall, Agent-E was self-aware of the failures for more than 52% of the failed tasks. Typically, Self-aware failures occur when the reason for failure are technical in nature (e.g., navigation issues, inability to extract certain information from DOM elements such as Iframes, canvas or images, inability to operate a button, anti-scraping policies employed by websites, inability to find the answer despite multiple attempts etc.). The Oblivious failures are scenarios where Agent-E gives a response that was wrong. These are typically scenarios where agent overlooks certain task requirements and provides an answer that only partially meets the requirements. It could also stem from DOM observation issues (e.g., not being aware that the date got reset due to incorrect format in Google Flights) or website capability understanding issues (e.g., not using advanced search capability when needed, or assuming search functionalities are perfect and every search result will completely satisfy the search requirements). Similar error modes were also observed by [He et al., 2024] who classify them as agent

Publication	Task success rates on websites							
	Dictionary	BBC	Flights	Maps	Search	Hug.Face	Wolfram	Overall
He et al. 2024 (text)	66.7	45.2	7.1	62.6	75.2	31.0	60.2	44.3
He et al. 2024 (multi)	71.3	60.3	**51.6**	64.3	77.5	55.8	60.9	57.1
Lutz et al. 2024 (text)	**86.0**	**81.0**	0.0	39.0	67.4	53.5	65.2	52.6
Agent-E (text)	81.4	73.8	35.7	**87.8**	**90.7**	**81.0**	**95.7**	**73.1**
Failure modes	Agent-E Error Analysis on Websites							
Overall failures %	18.6	26.2	64.3	12.2	9.3	19.0	4.3	26.9
Self-aware failures %	16.2	9.6	57.1	12.0	4.6	14.3	2.1	14.1
Oblivious failures %	2.4	16.6	7.1	0	4.6	4.7	2.1	12.8
TCT	Agent-E Avg. Task Completion Times (seconds)							
TCT (Success)	98	105	244	127	106	140	68	150
TCT (Failed)	136	110	234	177	135	167	94	220
LLM Calls	Agent-E Avg. Number of LLM calls per Task							
Total	22.0	21.3	53.8	22.9	19.4	22.8	14.5	25.0
Planner	6.6	6.0	11.4	5.8	5.6	6.2	4.4	6.4
Browser Nav Agent	15.4	15.3	42.2	17.0	13.7	16.6	10.15	18.6

Table 3: Evaluation of Agent-E on WebVoyager (Contd.)

hallucinations.

Task Completion Times

Agent-E on an average took 150 seconds to successfully complete a task and 220 seconds for completion when the task was a failure. The longer duration for failed tasks is interesting and also expected, since given a difficult task, Agent-E may try multiple approaches to complete the task before giving up on it. The difference in task completion times across websites (e.g., 68 seconds to successfully complete a task in WolframAlpha vs. 286 seconds in Amazon) also reflects the differences in task and website complexity. Amazon.com is a feature-rich and content-heavy website (for example, the Amazon and Booking.com homepage contains 3189 and 2,952 DOM elements respectively, in contrast to 1520 in Google Flights and 590 for WolframAlpha homepages). When the website is content-heavy and has numerous DOM elements, the DOM de-noising step takes more time, and the LLM token count for browser navigation agent may increase as well.

LLM Calls

On an average Agent-E took 25 LLM calls to execute a task (6.4 calls by the planner and almost 3 times as much by the browser navigation agent). The average number of LLM calls per website, as expected, is consistent with task completion times. Unfortunately, no prior studies have reported an end-to-end number of LLM calls required for task completion. While [Lutz et al., 2024] provides the number of LLM calls required by one (out of three) of the LLM-based components in their system, this information is insufficient for a conclusive comparison.

3.5 Result: Qualitative

In this section, we will present qualitative results of concrete examples showing how different design choices made in Agent-E help perform complex web tasks.

Hierarchical Planner helps error detection and recovery

The hierarchical architecture allows easily detecting and recovering from errors. The planner agent is prompted to perform verification (by asking questions or asking confirmation) as part of the plan whenever necessary. Figure 8 shows an example instance where the planner agent asks the browser navigation agent for more information (i.e., *list the search results*), and from the response (i.e., *there are no specific search results*) identifies that it may have made an error by making the search query too focused. In the example, the planner creates a new plan of action for performing the task. Another common pattern in the evaluation was the planner's ability to detect errors and easily backtrack to a previous page to continue execution. Given that the planner has the URL of the page at each step available to it, it allows the planner to effortlessly backtrack to a previous page by adding it as a step in the plan (e.g., *navigate to the search result page using the <url>*).
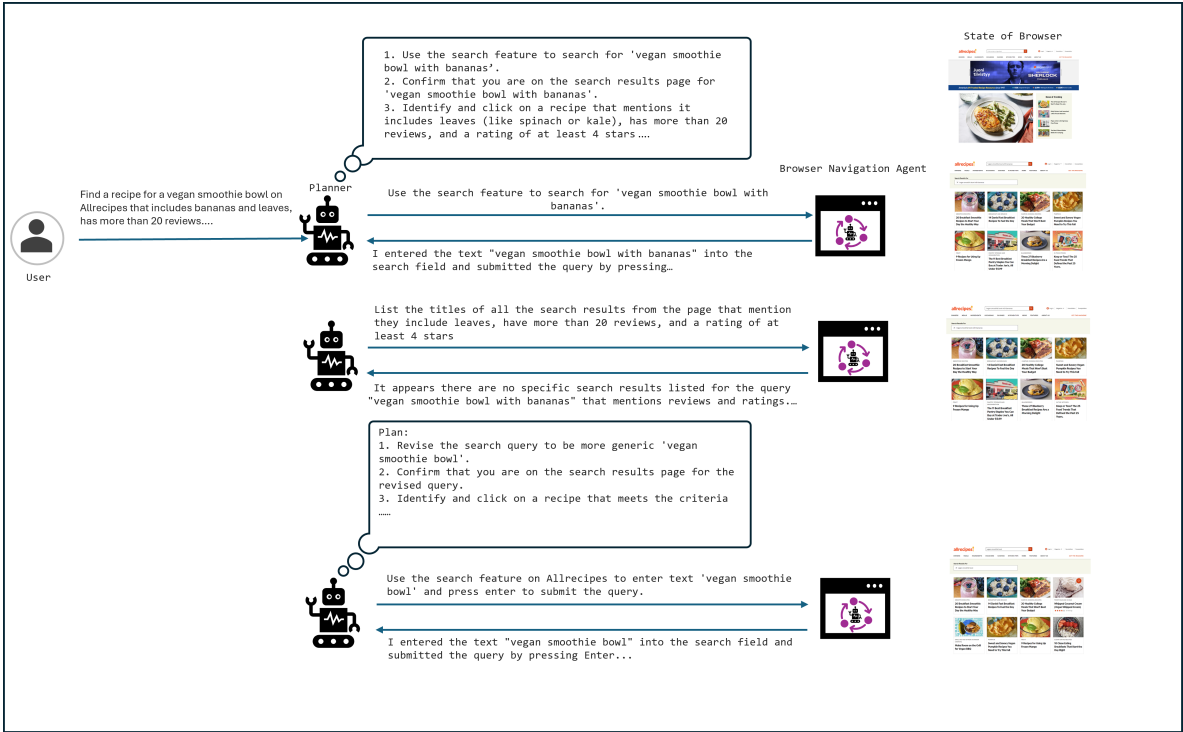

Figure 8: An example instance of Agent-E detecting and recovering from errors. The conversation is truncated with '...' to enhance readability in the image.

Support for multiple DOM observation methods

HTML DOM can be extremely large (e.g., YouTube Homepage with all the DOM elements and their attributes is about 800,000 tokens). Thus, it is important to denoise and encode the DOM such that only task relevant information is presented to the LLM. However, information relevant for a given task is very dependent on the task at the hand. Some tasks may only need a complete textual representation (e.g., *summarise the current page*), some tasks may only need input fields and buttons (e.g., *search on google*). On the other hand, more exploratory tasks, may need a complete representation of the page (e.g., *what elements are on this page*). Most previous web agent work have used a single DOM representation, e.g. [Zhou et al., 2023] used accessibility tree, multi-modal web agent in [He et al., 2024] used screenshots and [Lutz et al., 2024] used direct encoding and denoising of the HTML DOM. However, in our view, there is no single DOM observation method that suits all the

tasks. Thus, Agent-E supports three different DOM representation methods *text_only, input_fields, all_fields*. This allows Agent-E to flexibly select the DOM representation that it feels is best suited for the task. Also, it allows for backup, that if one does not work as expected, Agent-E can fall back to a different representation. There were numerous examples in our benchmark where this multiple DOM representation was useful. Figure 9 illustrates an example where Agent-E adaptively uses *all_fields* DOM representation for interaction and *text_only* for summarization.
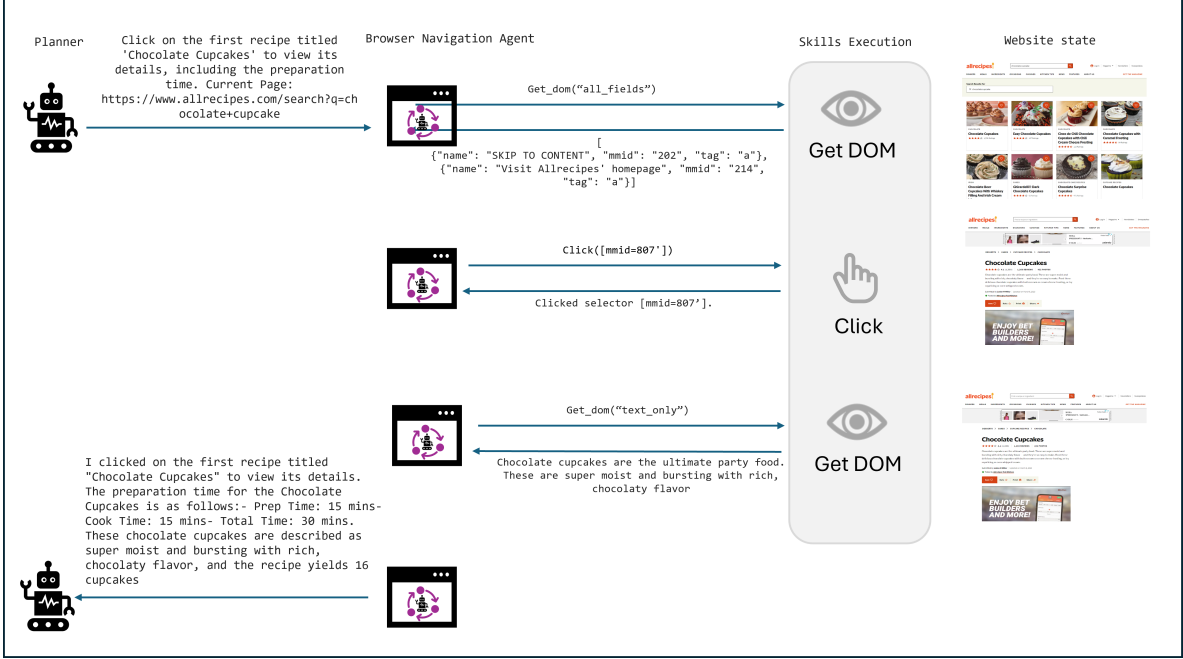

Figure 9: Providing multiple options for DOM observation allows to flexibly select one fit for task. The conversation is truncated with '...' to enhance readability in the image.

Change observation helps grounding

Change observation is a concept we introduced in which each action execution also observes for changes in the state and returns a linguistic feedback to the LLM. Some typical scenarios where this is useful is where the browser navigation agent tries to click on a navigation item (e.g., *click on the soccer link on ESPN.com*), however instead of navigating to the relevant section, it may instead open a popup menu that requires further selection. Another common example is when browser navigation agent attempts to set the source airport in a flight booking websites and a list of possible airports open up as a drop down. In both these cases, the interaction is not yet complete, but the browser navigation agent may assume it is (i.e., navigation to soccer section would require clicking on a new link from the popup menu, setting source airport would require selecting one of the option from the drop down). In both these cases, the *click* skills will return a feedback to the LLM that *as a consequence of the click, a menu has appeared where you may need to make further selection.* See Figure 6 for an example. Conceptually, the purpose of the change observation is to provide linguistic feedback to the LLM whether the action has been completed without any errors and did it lead to any tangible change in the environment, in order to inform subsequent actions. We also envision efficiency improvements if the change observation can return a list of elements, so that LLM can make subsequent selection without again using Get DOM skill to observe the state of the DOM. Change observation is adjacent to the concept of Reflexion [Shinn et al., 2024]. However, there are nuanced differences between the two. Reflexion paradigm provides feedback of a prior failure. However, change observation is not directly associated or limited to a prior failure. The observation returned can be any outcome of the action. Reflexion uses an LLM to analyse the scalar 'success/failure' signal from an action and current trajectory to produce the verbal feedback to the agent. Reflexion did not provide any tangible improvement when directly applied to web automation in the Webshop[Yao et al., 2022] benchmark.

On the other hand, change observation is implemented using classical code-based approaches to observe the consequence of an action and linguistic feedback of actions is implemented using heuristic-based approach to help the system to be better aware of the current state of the environment, and nudge the system towards the correct next action.

4 Discussion

In this section, we will synthesize our learnings from the development and evaluation of Agent-E into a series of agent design principles. We believe these principles can be generalized beyond the domain of web automation.

4.1 Agent Design Principles

1. **Well crafted set of primitive skills can enable powerful use-cases**: A well crafted ensemble of foundational skills can serve as a building block to support more complex functionalities. LLMs can effectively combine these skills to unlock a broad range of usecases. The analysis of the intended use case is crucial to arrive at the requisite primitive skills. In the case of Agent-E these primitive skills were *click, enter text, get DOM, Open URL* and *Press Keys*. These were only a subset of what a user could potentially perform on a page (e.g. we did not support *drag, double click, right click, tab management*, etc. We considered the primitive skills we enabled in Agent-E to be enough for vast majority of general web automation tasks. Nonetheless, if we were to specialise Agent-E to work on certain websites where right-click to select a functionality is a prominent interaction pattern, we would need to introduce that as a primitive skill.

2. **Consider hierarchical architecture for complex tasks**: Hierarchical architecture can be useful in agents with multiple LLM-based components. It allows execution of more complex tasks through a clear separation of roles and responsibilities. Hierarchical architecture excels in scenarios where tasks can be decomposed into sub-tasks that need to be handled at different levels of granularity. Additionally, it aids in the identification of tasks that can be executed in parallel, potentially leading to performance enhancements. It also supports the development and improvement of various components in isolation.
It is important to keep in mind that hierarchical architecture may not be suitable for all tasks. In case of Agent-E, if all we had to support were, e.g., navigate to specific URLs or perform simple web search, a hierarchical architecture may be over-complicated. In such cases, a much simpler architecture may suffice.

3. **Perform payload denoising when relevant**: Mitigating noise in the payload is critical in creating reliable, cost and time efficient agents. Noise could be in the form of unnecessary or irrelevant information, which could lead to incorrect or sub-optimal performance. It is also important to keep in mind that what is considered noise may be dependent on the task.
Simple techniques such as filtering irrelevant data, transforming the data to an easily consumable format, and focusing on key information can contribute to more accurate decision-making by the agent. This is especially important in environments with a high degree of uncertainty or very large input payload. In the case of Agent-E, we performed denoising on the HTML DOM. Further, we provided multiple DOM observation capabilities that the agent could adaptively use given the task requirements.

4. **Provide linguistic feedback of actions:** Our exploration into the space of agents capable of performing actions that has tangible consequences suggests that linguistic feedback of actions could be useful to help executor agents to be better aware of the environment and any consequence of the actions (e.g., *a dialog box appeared as a consequence of the click action*). Change observation helps refine the agent's subsequent actions by providing a clear narrative of cause and effect, and also improved awareness of the environment. This could apply in a variety of usecases such as desktop automation or automations in the physical space (e.g robot control).

5. **Support human in the loop architecture when necessary**: Supporting human in the loop architecture is critical for agentic systems. Given that agentic systems are so new, users may not trust the system completely. Further, there will arise scenarios where the agent may need to

have varying degree of user involvement for a multitude of reasons: e.g., to get more clarification on the task, when there is ambiguity how to proceed, to approve critical actions, perform some mission critical actions that the agent cannot or should not perform, audit the final output and re-invoke if needed, or take over responsibility when agent identifies that it cannot perform the task. Human in the loop architecture could also be leveraged for self-improvement where the agent identifies opportune times to gather human demonstration (e.g. *Sorry, I could not do this task. Can you show me how you would do it?*. Thus, incorporating mechanisms for human oversight can play a pivotal role in building that trust, and seamless handover.

Human in the loop could mean different levels of human involvement. From simple notifications of task progress, human involvement at key decision points, all the way to a handover of an incomplete task to a human.

6. **Analyse, reflect and aggregate past experiences routinely for self-improvement:** For Agentic systems to be adopted widely, they need to (gradually) achieve close to human-level performance. In the context of web automation, Agent-E could perform 73.1% of the tasks with an average task completion time of 150-220 second and 25 LLM calls. While very promising, this is not practical for production use-cases in terms of both effectiveness and efficiency.

A simple agentic solution to improving efficiency is to cache LLM calls. These mechanisms are supported out-of-the-box by agent development frameworks such as Autogen [Wu et al., 2023b]. However, a naive caching may not work well in dynamic contexts (e.g., a web site will continuously change in terms of content, advertisements etc.). A better approach would be to establish offline workflows that routinely analyse, reflects on and aggregates past tasks and human demonstrations to convert them to more classical automation workflows. These classical approach could be re-triggered upon a new task if it matches a workflow that it has encountered in the past, and use the exploratory agentic approach only as a fallback. This would mean the task could be completed faster and cheaper.

7. **Introduce internal and external guardrails into the Agentic System:** To ensure safe and effective operation of an agentic system, it's important to introduce various guardrails into its operation. These guardrails can be in the form of rules, guidelines or constitution that the system must adhere to. Some of the guardrails may be external to system, e.g., a task appropriateness guardrail may filter out inappropriate tasks even before agentic system is invoked. In addition, there could be guardrails intrinsic to the system such as guardrails against prompt injection attacks, and operational guardrails that limits how the agent performs different tasks. In the context of Agent-E, the operational guardrails include domain boundaries within which Agent-E should operate (e.g., *intranet of an organisation*), task specific boundaries (e.g., *Agent-E should only respond to a medical question from authoritative websites*) or user-configured boundaries (e.g., *for shopping, use only Amazon*)

8. **Choose between generic agent vs. task specific agent:** Generic agentic systems by definition can perform a wide range of tasks. However, in many practical implementations, a more focused set of capability may be desirable. For example, Agent-E is a generic web agent that can perform a wide range of tasks on the internet, but not necessarily optimized for any specific task. It would be possible to optimize Agent-E for specific type of tasks (e.g., form filling) or specific websites (e.g., Atlassian Confluence pages) to achieve significantly higher performance. Depending on the use case, an optimized agent may suit better for certain workflows than a generic version.

5 Related Work

LLM-based Planning and Reasoning Over the last few years, large language models (LLMs) have demonstrated extraordinary abilities in text generation, code generation, and the generation of natural language multi-step reasoning traces. Spurred by these, there has been much work in the use of LLMs to solve multi-step reasoning and planning problems. The many variants of 'chain-of-thought' techniques [Wei et al., 2023, Chu et al., 2023] encourage the LLM to produce a series of tokens with causal decoding that drive towards the solution of problems in math, common-sense reasoning and other similar tasks [Chowdhery et al., 2022, Fu et al., 2023, Li et al., 2023, Mitra et al., 2024].

With tool-usage for sensing and acting, LLMs have also been used to drive planning in software environments and embodied agents e.g., [Baker et al., 2022, Wang et al., 2023a, Wang et al., 2023b, Irpan et al., 2022, Bousmalis et al., 2023, Wu et al., 2023a, Bhateja et al., 2023]. Finally, there has been related work investigating the limits of LLMs when it comes to planning and validation. For examples of negative results, see [Valmeekam et al., 2023b, Momennejad et al., 2023, Valmeekam et al., 2023a, Huang et al., 2023, Kambhampati et al., 2024] among others.

Specialized Agents for Repetitive Tasks Beyond the examples above, and as described in Section 1, there has been much recent interest in building specialized agents for the web [Zheng et al., 2024, He et al., 2024, Lutz et al., 2024] and on device [Bai et al., 2024, Wen et al., 2024]. Also related is recent work on building agentic workflows to replace robotic process automation [Wornow et al., 2024]. Further afield, the work on building agents and training language models for API usage is also related, given that many software tasks and workflows involve the use of APIs; examples include [Hosseini et al., 2021, Patil et al., 2023, Qin et al., 2024] and many more.

Hierarchical Planning The notion of hierarchical AI planning has been around for five decades or more. Instead of planning directly in the space of low-level primitive actions, planning in a space of 'high-level actions' constrains the size of the plan length (and hence the size of the plan-space), which can result in more effective and efficient search. Examples from prior work include [Tate, 1977, Nau et al., 1991, Marthi et al., 2007] and many more; see [Russell and Norvig, 2009] for more details. Also related is the use of temporal abstractions in planning and in reinforcement learning, for example the use of options in [Sutton et al., 1999, Bacon et al., 2017]. In recent years, multiple papers have proposed the use of hierarchical planning for solving tasks in complex environments or with embodied agents; examples include [Wang et al., 2022, Irpan et al., 2022] and several others.

6 Conclusion

In this paper, we introduced Agent-E, a novel web agent designed to perform complex web-based tasks that makes use of numerous architectural improvements over prior state-of-the-art web agents such as hierarchical architecture, flexible DOM distillation and denoising method and concept of *change observation* to guide the agent towards more accurate performance.

Agent-E was evaluated on the WebVoyager benchmark, achieving a state-of-the-art success rate of 73.2%, a significant improvement over previous text-only and multi-modal web agents. Beyond task success rates, we also reported on additional metrics such as error awareness, task completion times, and the number of LLM calls, providing a more comprehensive evaluation of Agent-E's performance.

We presented our learnings in the form of eight general design principles for developing agentic systems that can be applied beyond the realm of web automation.

References

[Bacon et al., 2017] Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. *AAAI Conference on Artificial Intelligence*.

[Bai et al., 2024] Bai, H., Zhou, Y., Cemri, M., Pan, J., Suhr, A., Levine, S., and Kumar, A. (2024). Digirl: Training in-the-wild device-control agents with autonomous reinforcement learning. *arXiv preprint arXiv:2406.11896*.

[Baker et al., 2022] Baker, B., Akkaya, I., Zhokhov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., and Clune, J. (2022). Video pretraining (vpt): Learning to act by watching unlabeled online videos.

[Bhateja et al., 2023] Bhateja, C., Guo, D., Ghosh, D., Singh, A., Tomar, M., Vuong, Q., Chebotar, Y., Levine, S., and Kumar, A. (2023). Robotic offline rl from internet videos via value-function pre-training.

[Bousmalis et al., 2023] Bousmalis, K., Vezzani, G., Rao, D., Devin, C., Lee, A. X., Bauza, M., Davchev, T., Zhou, Y., Gupta, A., Raju, A., Laurens, A., Fantacci, C., Dalibard, V., Zambelli, M., Martins, M., Pevceviciute, R., Blokzijl, M., Denil, M., Batchelor, N., Lampe, T., Parisotto, E., Żołna, K., Reed, S., Colmenarejo, S. G., Scholz, J., Abdolmaleki, A., Groth, O., Regli, J.-B., Sushkov, O., Rothörl, T., Chen, J. E., Aytar, Y., Barker, D., Ortiz, J., Riedmiller, M., Springenberg, J. T., Hadsell, R., Nori, F., and Heess, N. (2023). Robocat: A self-improving foundation agent for robotic manipulation.

[Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., Reif, E., Du, N., Hutchinson, B., Pope, R., Bradbury, J., Austin, J., Isard, M., Gur-Ari, G., Yin, P., Duke, T., Levskaya, A., Ghemawat, S., Dev, S., Michalewski, H., Garcia, X., Misra, V., Robinson, K., Fedus, L., Zhou, D., Ippolito, D., Luan, D., Lim, H., Zoph, B., Spiridonov, A., Sepassi, R., Dohan, D., Agrawal, S., Omernick, M., Dai, A. M., Pillai, T. S., Pellat, M., Lewkowycz, A., Moreira, E., Child, R., Polozov, O., Lee, K., Zhou, Z., Wang, X., Saeta, B., Diaz, M., Firat, O., Catasta, M., Wei, J., Meier-Hellstern, K., Eck, D., Dean, J., Petrov, S., and Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

[Chu et al., 2023] Chu, Z., Chen, J., Chen, Q., Yu, W., He, T., Wang, H., Peng, W., Liu, M., Qin, B., and Liu, T. (2023). A survey of chain of thought reasoning: Advances, frontiers and future.

[Deng et al., 2024] Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. (2024). Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36.

[Fu et al., 2023] Fu, Y., Peng, H., Ou, L., Sabharwal, A., and Khot, T. (2023). Specializing smaller language models towards multi-step reasoning.

[He et al., 2024] He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., and Yu, D. (2024). Webvoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

[Hosseini et al., 2021] Hosseini, S., Awadallah, A. H., and Su, Y. (2021). Compositional generalization for natural language interfaces to web apis. *arXiv preprint arXiv:2112.05209*.

[Huang et al., 2023] Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., and Zhou, D. (2023). Large language models cannot self-correct reasoning yet.

[Irpan et al., 2022] Irpan, A., Herzog, A., Toshev, A. T., Zeng, A., Brohan, A., Ichter, B. A., David, B., Parada, C., Finn, C., Tan, C., Reyes, D., Kalashnikov, D., Jang, E. V., Xia, F., Rettinghouse, J. L., Hsu, J. C., Quiambao, J. L., Ibarz, J., Rao, K., Hausman, K., Gopalakrishnan, K., Lee, K.-H., Jeffrey, K. A., Luu, L., Yan, M., Ahn, M. S., Sievers, N., Joshi, N. J., Brown, N., Cortes, O. E. E., Xu, P., Sampedro, P. P., Sermanet, P., Ruano, R. J., Julian, R. C., Jesmonth, S. A., Levine, S., Xu, S., Xiao, T., Vanhoucke, V. O., Lu, Y., Chebotar, Y., and Kuang, Y. (2022). Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

[Kambhampati et al., 2024] Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., and Murthy, A. (2024). Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*.

[Kapoor et al., 2024] Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N., and Narayanan, A. (2024). Ai agents that matter. *arXiv preprint arXiv:2407.01502*.

[Li et al., 2023] Li, Y., Bubeck, S., Eldan, R., Giorno, A. D., Gunasekar, S., and Lee, Y. T. (2023). Textbooks are all you need ii: phi-1.5 technical report.

[Lutz et al., 2024] Lutz, M., Bohra, A., Saroyan, M., Harutyunyan, A., and Campagna, G. (2024). Wilbur: Adaptive in-context learning for robust and accurate web agents. *arXiv preprint arXiv:2404.05902*.

[Marthi et al., 2007] Marthi, B., Russell, S., and Wolfe, J. (2007). Angelic semantics for high-level actions. *International Conference on Automated Planning and Scheduling.*

[Mitra et al., 2024] Mitra, A., Khanpour, H., Rosset, C., and Awadallah, A. (2024). Orca-math: Unlocking the potential of slms in grade school math.

[Momennejad et al., 2023] Momennejad, I., Hasanbeig, H., Vieira, F., Sharma, H., Ness, R. O., Jojic, N., Palangi, H., and Larson, J. (2023). Evaluating cognitive maps and planning in large language models with cogeval.

[Nakano et al., 2022] Nakano, R., Hilton, J., Balaji, S., Wu, J., Ouyang, L., Kim, C., Hesse, C., Jain, S., Kosaraju, V., Saunders, W., Jiang, X., Cobbe, K., Eloundou, T., Krueger, G., Button, K., Knight, M., Chess, B., and Schulman, J. (2022). Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332.*

[Nau et al., 1991] Nau, D., Cao, Y., Lotem, A., and Muñoz-Avila, H. (1991). Shop: Simple hierarchical ordered planner. *International Joint Conference on Artificial Intelligence.*

[Patil et al., 2023] Patil, S. G., Zhang, T., Wang, X., and Gonzalez, J. E. (2023). Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334.*

[Qin et al., 2024] Qin, Y., Liang, S., Ye, Y., Zhu, K., Yan, L., Lu, Y., Lin, Y., Cong, X., Tang, X., Qian, B., Zhao, S., Hong, L., Tian, R., Xie, R., Zhou, J., Gerstein, M., Li, D., Liu, Z., and Sun, M. (2024). Toolllm: Facilitating large language models to master 16000+ real-world apis. *International Conference on Learning Representations.*

[Russell and Norvig, 2009] Russell, S. J. and Norvig, P. (2009). *Artificial Intelligence: a modern approach.* Pearson, 3 edition.

[Shinn et al., 2024] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2024). Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

[Sutton et al., 1999] Sutton, R., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence Journal.*

[Tate, 1977] Tate, A. (1977). Generating project networks. *International Joint Conference on Artificial Intelligence.*

[Valmeekam et al., 2023a] Valmeekam, K., Marquez, M., and Kambhampati, S. (2023a). Can large language models really improve by self-critiquing their own plans?

[Valmeekam et al., 2023b] Valmeekam, K., Sreedharan, S., Marquez, M., Olmo, A., and Kambhampati, S. (2023b). On the planning abilities of large language models (a critical investigation with a proposed benchmark).

[Wang et al., 2023a] Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023a). Voyager: An open-ended embodied agent with large language models.

[Wang et al., 2022] Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., and Liang, Y. (2022). Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *Advances in Neural Information Processing Systems*, 37.

[Wang et al., 2023b] Wang, Z., Cai, S., Liu, A., Jin, Y., Hou, J., Zhang, B., Lin, H., He, Z., Zheng, Z., Yang, Y., Ma, X., and Liang, Y. (2023b). Jarvis-1: Open-world multi-task agents with memory-augmented multimodal language models.

[Wei et al., 2023] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.

[Wen et al., 2024] Wen, H., Li, Y., Liu, G., Zhao, S., Yu, T., Li, T. J.-J., Jiang, S., Liu, Y., Zhang, Y., and Liu, Y. (2024). Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 543–557.

[Wornow et al., 2024] Wornow, M., Narayan, A., Opsahl-Ong, K., McIntyre, Q., Shah, N. H., and Re, C. (2024). Automating the enterprise with foundation models. *arXiv preprint arXiv:2405.03710*.

[Wu et al., 2023a] Wu, H., Jing, Y., Cheang, C., Chen, G., Xu, J., Li, X., Liu, M., Li, H., and Kong, T. (2023a). Unleashing large-scale video generative pre-training for visual robot manipulation.

[Wu et al., 2023b] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., and Wang, C. (2023b). Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*.

[Yao et al., 2022] Yao, S., Chen, H., Yang, J., and Narasimhan, K. (2022). Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

[Zheng et al., 2024] Zheng, B., Gou, B., Kil, J., Sun, H., and Su, Y. (2024). Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*.

[Zhou et al., 2023] Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Bisk, Y., Fried, D., Alon, U., et al. (2023). Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.