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

Tamer Abuelsaad
Emergence AI
tamer@emergence.ai

Deepak Akkil
Emergence AI
deepak@emergence.ai

Prasenjit Dey
Emergence AI
prasenjit@emergence.ai

Ashish Jagmohan
Emergence AI
ashish@emergence.ai

Aditya Vempaty
Emergence AI
aditya@emergence.ai

Ravi Kokku
Emergence AI
ravi@emergence.ai

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, and the advantages of a hierarchical architecture.

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 [38, 10, 16]. Automation of complex and repetitive tasks presents an invaluable opportunity to increase individual and organizational efficiency.

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 [10]. 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 [16]. Thirdly, while humans can naturally execute complex web tasks (e.g., finding the cheapest flight between two destinations), agents require detailed multi-step planning

[1]https://github.com/EmergenceAI/Agent-E

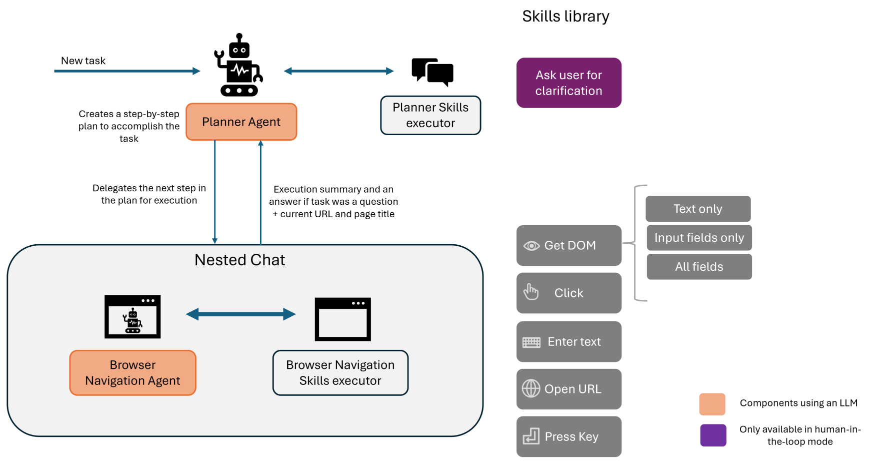

Skills library
New task
Planner Agent
Creates a step-by-step plan to accomplish the task
Planner Skills executor
Ask user for clarification
Delegates the next step in the plan for execution
Execution summary and an answer if task was a question + current URL and page title
Nested Chat
Browser Navigation Agent
Browser Navigation Skills executor
Get DOM
Text only
Input fields only
All fields
Click
Enter text
Open URL
Press Key
Components using an LLM
Only available in human-in-the-loop mode

Figure 1: A high level architecture of Agent-E

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 [34]. 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 [16, 10, 39].

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. The browser navigation agent focuses solely on the current step of the plan.

We evaluate Agent-E on the WebVoyager benchmark [10], where we achieve a new state-of-the-art result of 73.2%, 20% higher than previous state-of-the-art for text-only [16] and 16% higher than previous state-of-the-art multi-modal web agent [10].

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 introduce two novel components in Agent-E, a flexible DOM distillation approach where the browser navigation agent selects the most suitable DOM representation given the task, and the concept of *'change observation'*, a Reflexion-like paradigm [24], where the agent monitors state changes after each action and receives verbal feedback to enhance awareness and 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.

- We synthesisze our learning from the development of Agent-E into design principles for agentic systems, offering generalizable guidance beyond Agent-E and web automation for practitioners.

2 Agent-E: System Description

Figure 1 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*, *enter text on element*, *open url*, *press key (e.g. Enter*). 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 [36] and Playwright[2] for browser control. Its architecture leverages the interplay between skills and agents as shown in Figure 1.

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 step of the plan (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 (See Appendix A: Figure 3 for 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 (See Appendix A: Figure 4 for an example communication between planner and the browser navigation agent and Figure 5 for the browser navigation agent using the primitive skills available to it to perform a related sub task).

2.1 Skills Design for Browser Navigation Agent

There are two key novel components in skills design used in Agent-E.

- Sensing Skills: Agent-E supports three different DOM synthesis techniques (*text only, input fields, all fields*) that allows the browser navigation agent to choose the approach best suited for the task (see Appendix A: Figure 3). 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 optimizes the information available to the agent and prevents the problems associated with noisy DOM. Another key aspect is that our DOM de-noising techniques for *all_fields* and *input_fields* attempt to preserve the parent child relationship of elements wherever possible and relevant. This is unlike some previous implementations which use a flat DOM encoding (e.g. Lutz et al. [16]). Further, to make identifying and interacting with HTML elements easier, Agent-E injects a custom identifier attribute (*mmid*) in each element as part of sensing, similar to Zhou et al. [39] and He et al. [10].
- 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 the Reflexion paradigm [24] 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 detailed skill responses nudge the agent towards the correct next step.

[2] `https://playwright.dev/`

The change observation capability is implemented by observing changes in selected attributes of elements (e.g *aria-expanded*) using the 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 flight booking websites. (see Appendix A: Figure 5 for an example)

2.2 Self-Improvement

3 Evaluation

3.1 The WebVoyager Benchmark

WebVoyager [10] is a recent web agent benchmark that consists of web navigation tasks across 15 real websites such as Amazon, Google Flights, Github, Booking.com etc. Each website has about 40-46 tasks resulting in a benchmark dataset of 643 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 focus solely on a single task domain [37], use custom created websites that are significantly simpler than real-world versions in terms of DOM complexity [39], or use cached versions of real-websites only supporting a fixed route for the agent and not allowing free form exploration [8].

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). For the evaluation, we used GPT-4-Turbo as the LLM for both planner and browser navigation agent.

3.3 Results

In this section, we present quantitative results measuring Agent-E's performance on the WebVoyager benchmark, and comparing to prior state-of-art approaches. Table 1 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 2024 (text)	57.8	43.1	36.4	50.4	63.4	2.3	28.6	24.6
He 2024 (multi)	51.1	52.9	62.8	52.0	59.3	32.6	47.0	57.9
Lutz 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**
Publication	**Task success rates on websites**							
	Dictionary	BBC	Flights	Maps	Search	Hug.Face	Wolfram	Overall
He 2024 (text)	66.7	45.2	7.1	62.6	75.2	31.0	60.2	44.3
He 2024 (multi)	71.3	60.3	**51.6**	64.3	77.5	55.8	60.9	57.1
Lutz 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**

Table 1: Evaluation of Agent-E on WebVoyager and comparison with other webagents He 2024 [10] and Wilbur [16]

Comparison with prior state-of-the-art agents

Overall, Agent-E successfully completed 73% of the tasks and outperformed the prior state-of-the-art text-only web agent WILBUR [16] by 21% and the state-of-the-art multi-modal web agent [10]

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

by 16%. The effectiveness of Agent-E varied across websites from 27.3% (Booking) to 95.7% (WolframAlpha). Comparing performance on specific websites, Agent-E outperformed both Wilbur [16] and the WebVoyager multi-modal agent [10] on 11 out of 15 websites. Some websites remain hard for all web agents; specifically, on Booking.com and Google Flights all web agents, including Agent-E, did not do well.

It is important to note that WILBUR [16] uses task and website-specific prompting, while He at al. [10] use vision for observing the page. In contrast, Agent-E is a general, 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 website/task-specific strategies and vision.

Agent-E failure modes

We also asked: Is Agent-E aware when it fails to complete a task? This is important for reliability, because an agent that is aware of failures can take corrective actions like invoking human help.

We found that Agent-E was self-aware of failures for more than 52% of the failed tasks, i.e. it was obvious from Agent-E response that it could not complete the task (e.g. *I'm unable to provide a description of the first picture due to limitations in accessing or analyzing visual content.*). 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.).

On the other hand, oblivious failures are scenarios where Agent-E gave a response that was wrong. These are typically scenarios where the agent overlooks certain task requirements and provides an answer that only partially meets the requirements. These also stem from DOM observation issues (e.g., not being aware that the date got reset due to incorrect format in Google Flights) or issues in understanding website capabilities (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. [10] who classify them as agent *hallucinations*.

For more detailed analysis on Agent-E's self-aware vs oblivious failure modes across websites, see Appendix B.

Task Completion Times

On average, Agent-E took significantly longer for completion when the task was a failure, versus on successful tasks (an average of 220 seconds on failed tasks vs 150 seconds on successful tasks, in our experiments). The longer duration for failed tasks is expected, since given a difficult task, Agent-E may try multiple approaches to complete the task before giving up on it. There were also significant differences in task completion times across websites (e.g., 68 seconds to successfully complete a task in WolframAlpha vs. 286 seconds in Amazon), reflecting the differences in task and website complexity. See Tables 2 and 3 in Appendix B for detailed analysis.

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. See Tables 2 and 3 in Appendix B for detailed analysis on LLM calls.

3.4 Qualitative Analysis

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

Hierarchical planning 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. Appendix A: Figure 7 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>*).

The need for multiple DOM observation methods

Typical HTML DOMs can be extremely large (e.g., the YouTube homepage with all DOM elements and 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 hand. Some tasks may only need a complete textual representation (e.g., *summarise the current page*), and 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 agents have used a single DOM representation, e.g. Zhou et al. [39] used accessibility tree, He et al. [10] used screenshots and Lutz et al. [16] 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, this allows Agent-E to fallback to different representations, when one representation unexpectedly does not work well. There were numerous examples in our benchmark where these multiple DOM representations were useful. Appendix A: Figure 6 illustrates an example where Agent-E adaptively uses *all_fields* DOM representation for interaction and *text_only* for summarization.

Change observation helps grounding

Change observation is a technique wherein each action execution is accompanied by an observation of changes in state, and this is returned via linguistic feedback to the LLM. A typical scenario where this is useful is when the browser navigation agent tries to click on a navigation item (e.g., *click on the soccer link on ESPN.com*), and instead of navigating to the relevant section, the page instead opens a popup menu that requires further selection. Another common example is when the browser agent attempts to set the source airport in a flight booking website, and a list of possible airports opens up as a drop down. In both these cases, the interaction is not yet complete (since completion requires clicking a popup link or selecting a drop down entry, respectively), but the browser navigation agent may assume it is. With change observation, in both these cases, the *click* skill will return feedback to the LLM that *as a consequence of the click, a menu has appeared where you may need to make further selection*. See Appendix A: Figure 5 for an example.

Conceptually, the purpose of change observation is to provide linguistic feedback to the LLM whether the action led 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 [24]. However, there are nuanced differences between the two. The Reflexion technique provides feedback on a prior failure, by using an LLM to analyse the scalar 'success/failure' signal based on an action and current trajectory. In contrast, change observation is not a binary signal and instead observes the change in the environment as a consequence of an action (e.g. new elements added to DOM, pop-up expanded etc). Change observation is implemented using mutation-observer API to observe the consequence of an action and provide linguistic feedback of actions to help the system to be better aware of the new state of the environment, and nudge the system towards the correct next action.

4 Discussion

In this section, we 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 sets 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 consider the primitive skills we enabled in Agent-E to be enough for the vast majority of general web automation tasks. Nonetheless, if one were to specialise Agent-E to work on certain websites where right-click to select a functionality is a prominent interaction pattern, it may be advisable to introduce that as a new skill.
2. **Consider hierarchical architectures for complex tasks**: Hierarchical architectures can be useful in agents with multiple LLM-based components. They allow execution of more complex tasks through a clear separation of roles and responsibilities. A 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. **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 all use cases.
   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 [36]. 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, reflect on and aggregate past tasks

and human demonstrations to convert them to more classical automation workflows. These automated workflows could then be re-triggered upon a new task if it matches a workflow that has been encountered in the past, with the exploratory agentic approach used only as a fallback. This would enable faster and cheaper task completion.

6. **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 [32, 7] 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 [6, 9, 15, 18]. With tool-usage for sensing and acting, LLMs have also been used to drive planning in software environments and embodied agents e.g., [3, 29, 31, 13, 5, 35, 4]. Finally, there has been related work investigating the limits of LLMs when it comes to planning and validation. For examples of negative results, see [28, 19, 27, 12, 14] 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 [38, 10, 16] and on device [2, 33]. Also related is recent work on building agentic workflows to replace robotic process automation [34]. 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 [11, 21, 22] 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 [26, 20, 17] and many more; see Russell et al. [23] for more details. Also related is the use of temporal abstractions in planning and in reinforcement learning, for example the use of options in [25, 1]. In recent years, multiple papers have proposed the use of hierarchical planning for solving tasks in complex environments or with embodied agents; examples include [30, 13] 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, including the use of hierarchical architecture, flexible DOM distillation and denoising methods, and the 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 six general design principles for developing agentic systems that can be applied beyond the realm of web automation.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

A Supplementary Diagrams

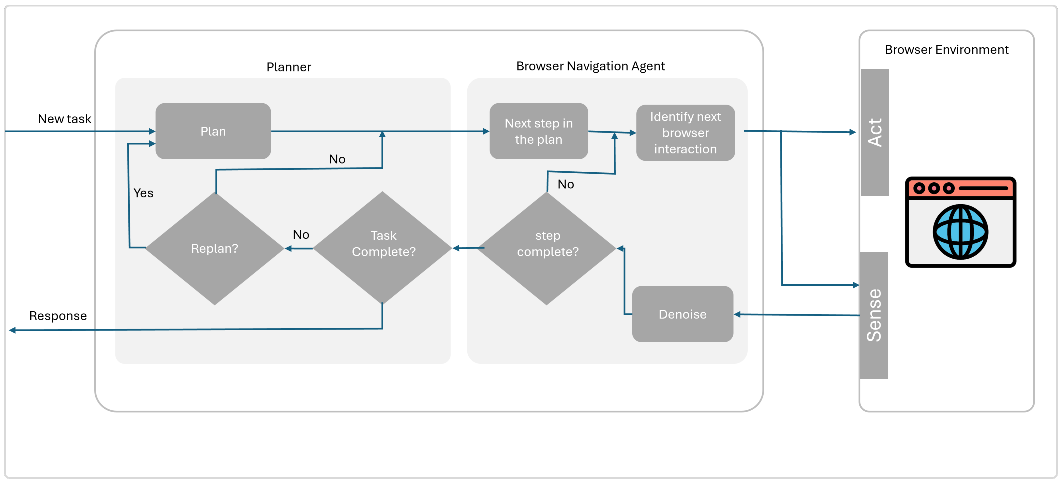
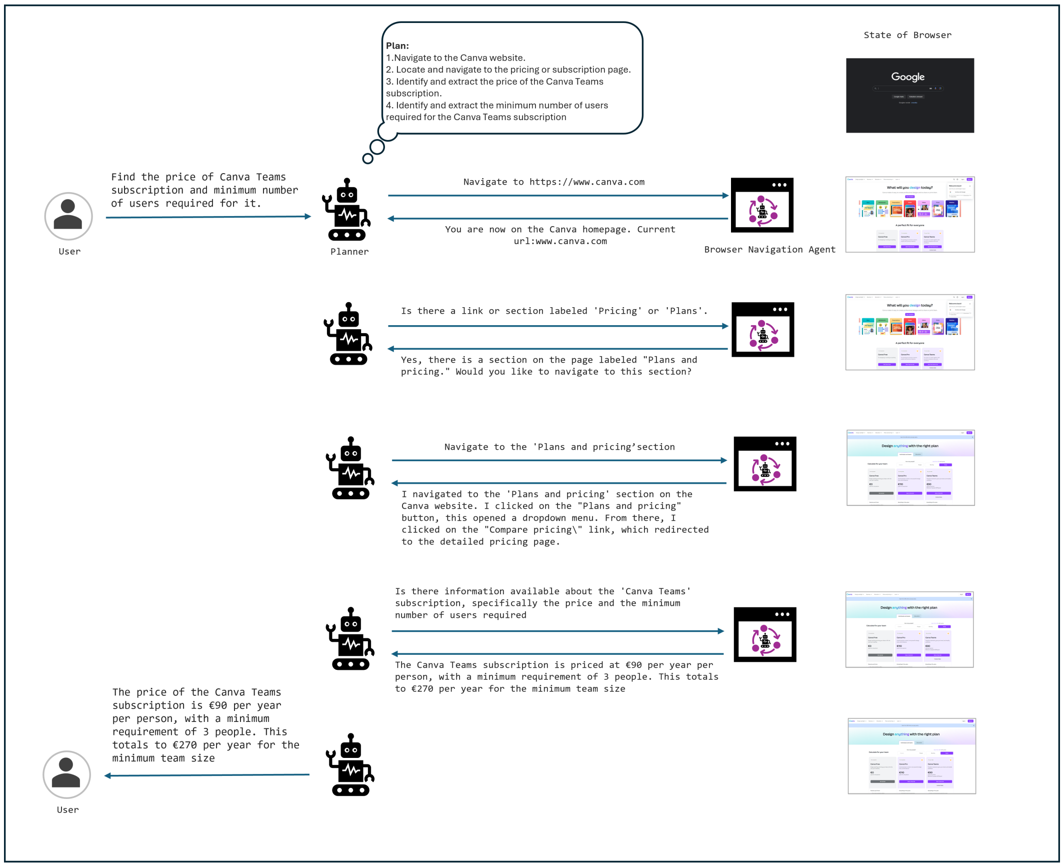


Figure 2: Conceptual flow diagram of Agent-E. Individual blocks represent functions; In Agent-E, a single LLM call is used to perform multiple functions.

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



Plan:
1.Navigate to the Canva website.
2. Locate and navigate to the pricing or subscription page.
3. Identify and extract the price of the Canva Teams subscription.
4. Identify and extract the minimum number of users required for the Canva Teams subscription

Find the price of Canva Teams subscription and minimum number of users required for it.

Navigate to https://www.canva.com

You are now on the Canva homepage. Current url:www.canva.com

Is there a link or section labeled 'Pricing' or 'Plans'.

Yes, there is a section on the page labeled "Plans and pricing." Would you like to navigate to this section?

Navigate to the 'Plans and pricing'section

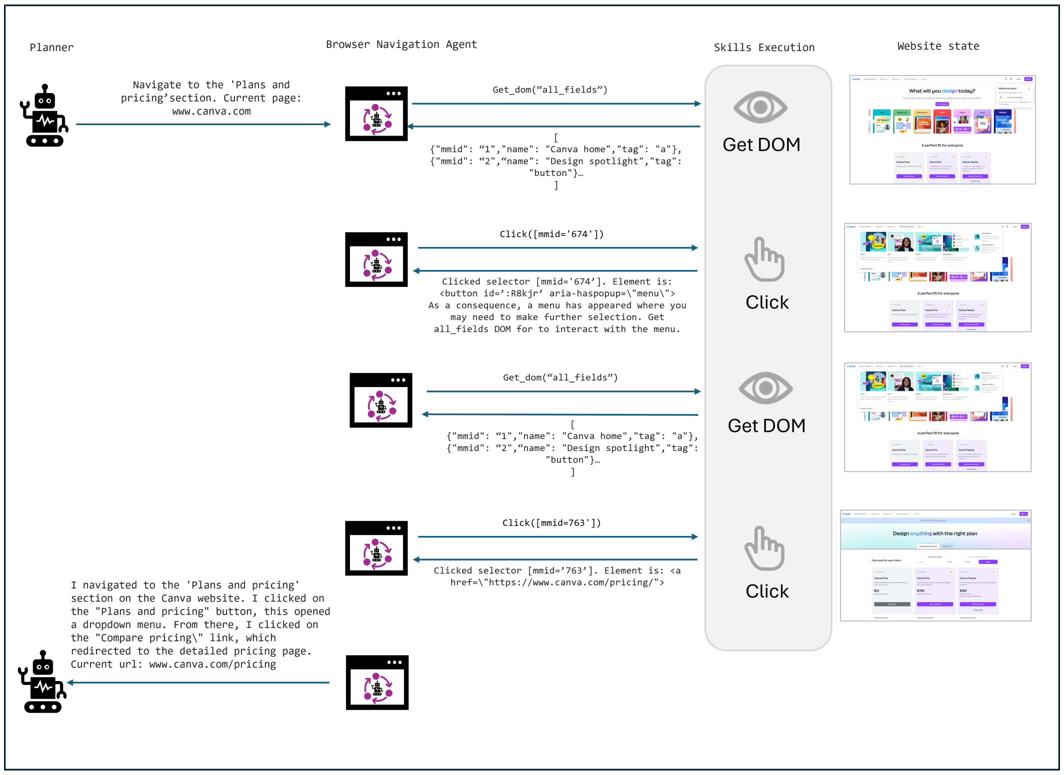
I navigated to the 'Plans and pricing' section on the Canva website. I clicked on the "Plans and pricing" button, this opened a dropdown menu. From there, I clicked on the "Compare pricing\" link, which redirected to the detailed pricing page.

Is there information available about the 'Canva Teams' subscription, specifically the price and the minimum number of users required

The Canva Teams subscription is priced at €90 per year per person, with a minimum requirement of 3 people. This totals to €270 per year for the minimum team size

The price of the Canva Teams subscription is €90 per year per person, with a minimum requirement of 3 people. This totals to €270 per year for the minimum team size

Figure 4: 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*



Navigate to the 'Plans and pricing’section. Current page: www.canva.com

Get_dom(“all_fields”)

```
[
{"mmid": “1","name": "Canva home","tag": "a"},
{"mmid": “2",“name": "Design spotlight","tag": "button"}…
]
```

Click([mmid='674'])

Clicked selector [mmid='674’]. Element is: <button id=’:R8kjr’ aria-haspopup=\"menu\"> As a consequence, a menu has appeared where you may need to make further selection. Get all_fields DOM for to interact with the menu.

Get_dom(“all_fields”)

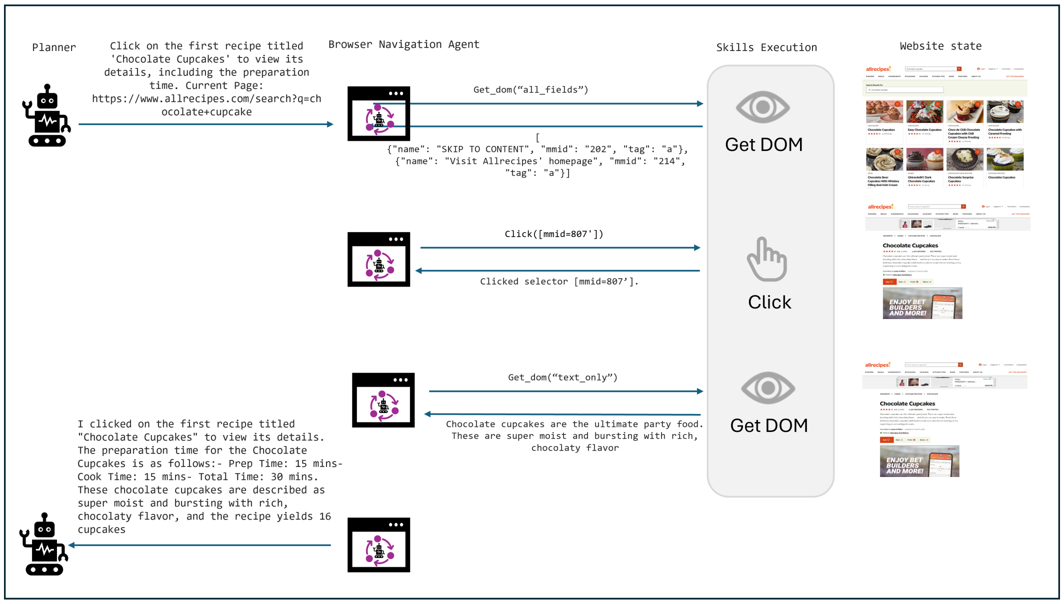
```
[
{"mmid": “1","name": "Canva home","tag": "a"},
{"mmid": “2",“name": "Design spotlight","tag": "button"}…
]
```

Click([mmid=763'])

Clicked selector [mmid=’763’]. Element is: <a href=\"https://www.canva.com/pricing/">

I navigated to the 'Plans and pricing' section on the Canva website. I clicked on the "Plans and pricing" button, this opened a dropdown menu. From there, I clicked on the "Compare pricing\" link, which redirected to the detailed pricing page. Current url: www.canva.com/pricing

Figure 5: 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"*



Click on the first recipe titled 'Chocolate Cupcakes' to view its details, including the preparation time. Current Page: https://www.allrecipes.com/search?q=chocolate+cupcake

Get_dom(“all_fields”)

```
[
{"name": "SKIP TO CONTENT", "mmid": "202", "tag": "a"},
{"name": "Visit Allrecipes' homepage", "mmid": "214", "tag": "a"}]
```

Click([mmid=807'])

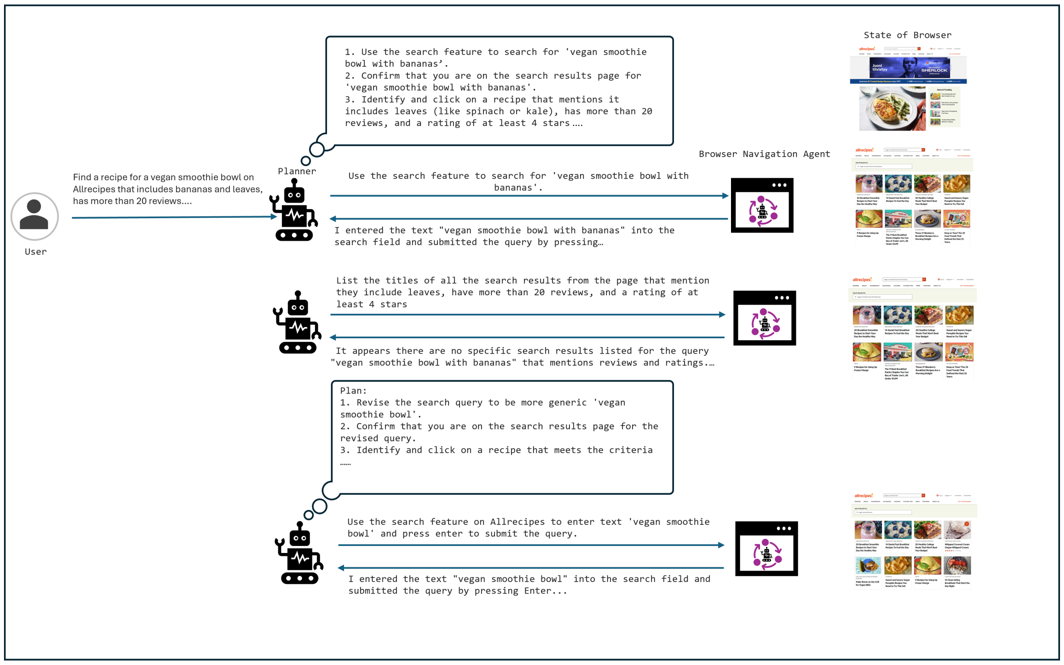
Clicked selector [mmid=807’].

Get_dom(“text_only”)

Chocolate cupcakes are the ultimate party food. These are super moist and bursting with rich, chocolaty flavor

I clicked on the first recipe titled "Chocolate Cupcakes" to view its details. The preparation time for the Chocolate Cupcakes is as follows:- Prep Time: 15 mins- Cook Time: 15 mins- Total Time: 30 mins. These chocolate cupcakes are described as super moist and bursting with rich, chocolaty flavor, and the recipe yields 16 cupcakes

Figure 6: 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.



1. Use the search feature to search for 'vegan smoothie bowl with bananas'.
2. Confirm that you are on the search results page for 'vegan smoothie bowl with bananas'.
3. Identify and click on a recipe that mentions it includes leaves (like spinach or kale), has more than 20 reviews, and a rating of at least 4 stars….

Find a recipe for a vegan smoothie bowl on Allrecipes that includes bananas and leaves, has more than 20 reviews….

Use the search feature to search for 'vegan smoothie bowl with bananas'.

I entered the text "vegan smoothie bowl with bananas" into the search field and submitted the query by pressing…

List the titles of all the search results from the page that mention they include leaves, have more than 20 reviews, and a rating of at least 4 stars

It appears there are no specific search results listed for the query "vegan smoothie bowl with bananas" that mentions reviews and ratings.…

Plan:
1. Revise the search query to be more generic 'vegan smoothie bowl'.
2. Confirm that you are on the search results page for the revised query.
3. Identify and click on a recipe that meets the criteria
……

Use the search feature on Allrecipes to enter text 'vegan smoothie bowl' and press enter to submit the query.

I entered the text "vegan smoothie bowl" into the search field and submitted the query by pressing Enter...

Figure 7: An example instance of Agent-E detecting and recovering from errors. The conversation is truncated with '...' to enhance readability in the image.

B Additional Results

In this section, we present supplementary quantitative evaluation of Agent-E on WebVoyager. We report three additional measures that are relevant for comprehensive evaluation of web agent and understanding their practical implementation readiness.

- 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 [34].
- 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.

	Allrecipe	Amazon	Apple	Arxiv	Github	Booking	ESPN	Coursera
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.

	Dictionary	BBC	Flights	Maps	Search	Hug.Face	Wolfram	Overall
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.)