Development of LLM-Driven GUI Agents

Professor:	Prof. Chunyang Chen
Supervisor:	Ludwig Felder, M. Sc.
Chair:	Chair of Software Engineering & AI
Authors:	Alp Karaagac[1], Ali Rabeh[2]
Date:	8 August 2025

[1] alp.karaagac@tum.de
[2] ali.rabeh@tum.de

Abstract. GUI Agents, powered by Large Language Models, are a new approach to automating human-computer interaction. These agents can interact with digital systems via GUIs and imitate human actions such as clicking and typing. Web agents are specialized GUI agents that focus on interacting with browsers instead of the OS or any other digital system. Almost every day, new agents compete on different benchmarks to see who can successfully solve a task and how many steps it requires. Motivated by this trend, we provide a detailed explanation and analysis of our improvement upon an open-source web agent project: Browser Use. Models are tested on the benchmark Online Mind2Web to see the models' limitations and how these issues can be solved.

Our software code is available at the following URL: Git Repo.
The Online Mind2Web dataset is available at the following URL: Online Mind2Web.

Table of Contents

1 Introduction

Large Language Models (LLMs) are becoming more popular daily,[1] transforming multiple aspects of our daily lives. For a long time, they were more popular as only conversational chatbots, where they would receive a prompt and auto-regressively generate an answer. Later, however, they were also used as agents' brains.[2] This allows the models to perform actions and interact with the environment, making them suitable for automation. For web-based tasks, things like Selenium were used for automation, which were not very user-friendly[3] for people who could not code. Nor could they adapt very well to unseen websites by themselves. This led to LLM-based web agents that could proactively make choices depending on the state and fulfill most website tasks without extra tuning. However, deploying LLMs presents unique challenges, with dynamic layouts, diverse graphical designs, and grounding issues. In our project, we base our code on an open-source project called "Browser-Use"[4] and compare our improvements to the baseline using the benchmark Online Mind2Web[5]. We first explain the architecture of the default Browser-use. Afterwards, we analyse how Online Mind2Web works. With these preliminaries explained, we then describe our improvements and compare them against the baseline. Finally, we discuss what should be researched in the future and conclude this report.

2 GUI Agents

GUI Agents interact with an environment sequentially. This can be modelled as a Partially Observable Markov Decision Process[6] ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{O}$, $\mathcal{T}$). $\mathcal{A}$ is the action space, i.e., the actions that the GUI Agents can do, $\mathcal{S}$ is the state space, and $\mathcal{O}$ is the observation space. $\mathcal{T}$ is a state transition function, mapping a state and an action to a new state. In this model, a GUI Agent is a mapping that receives $\mathcal{O}$ and outputs an action $\mathcal{A}$ to be performed.

3 Browser Use

Browser-Use[4] follows a three-tier design. This consists of planning, dispatch, and execution. There is the "Agent", which receives the task the user wants to complete. It provides multiple options, such as enabling memory, which might improve the performance and the success rate. At every step, the Agent feeds the LLM with the observed state and other extra information, such as the memory. For feeding in the state, Browser-use uses DOM snapshots and screenshots. This screenshot enumerates clickable objects, which makes it easier for the LLM to pick an action. The chosen call is routed to a Controller that lists all available actions. Each action is a typed Python coroutine annotated with a short docstring. The controller injects framework objects, such as page or browser session, executes the code, and wraps the result to be usable by the memory. The interaction happens in a Playwright BrowserContext.

Browser-Use is, of course, not perfect. Our first experiments revealed weaknesses in several different aspects. Firstly, an option provided by the Agent allowed for multiple actions per observable state. These dynamic layouts sometimes lead to the first action changing the environment. Unfortunately, Browser-User is not able to handle this. The change of the dynamic layout changes clickable elements, which are then indexed differently than they were before. Thus, max_actions_per_subtask is always limited to 1.

Besides that, Browser-use has a hard time with longer tasks. It can sometimes forget a step. Some tasks defined many filtering options that needed to be applied. Browser-use falsely concluded it was done without fulfilling all constraints. Since almost 75% of all tasks were connected with planning issues, this is our focus for improvement throughout the project.

4 Online Mind2Web

Online Mind2Web[5] is an improvement upon the offline version. They both consist of different tasks from different websites. These tasks require the correct navigation of the websites and taking the correct actions to complete a task. Unlike the offline version, online mind2web tasks are performed on live websites. This introduces new challenges, such as Captchas, pop-ups, and ads. Online Mind2Web receives the task and the history of the trajectories from the Agent. First, it extracts the key points from the task. For example, given a task such as: "Find a fridge 34-36 wide under 2000 Euros," it extracts "Filter by width 34-36" and "Filter by price" and so on. Afterwards, some screenshots are selected based on their relevance to the identified key points. Moreover, they are both sent to an LLM for benchmarking. At the end, the LLM returns whether the task was completed.

Some problems with the Online Mind2Web stem from this procedure. In trajectories with longer steps, we have seen that this mechanism becomes less reliable. Online Mind2Web can pick false screenshots associated with an identified key point. Also, some tasks are not completable. Some websites are location-blocked in Germany, such as Healthgrades.com. Moreover, some tasks are borderline impossible, and some website filtering options do not exist. What makes Online Mind2Web results less reliable is that in these cases, LLM can randomly evaluate the task as succeeded because there was no option for filtering or as failed because the filtering was not applied. Some websites dynamically change their filtering options based on the last action, for example, in Shein.com, the Agent can pick the color option blue first and then the item, but if the Agent picks the item first and there is no blue item, the Agent cannot then filter for blue color. With these in mind, we started to improve upon the model. All benchmark tests are ran using gemini-2.5-flash.

5 Automatic Prompt Optimization

For any project that uses an LLM, prompt optimization is always essential for good results.[7, 8] Prompt optimization can be defined as: Crafting and refining a text to feed an LLM so that the model's latent knowledge is steered toward the desired answer. Even browser-use changes its prompt every day, as this must be a continuous process for improving the quality of the LLM output. Our first experiments regarding prompt optimization were always random. So we wanted to automate this process.[9]

Some rules for this process are as follows:

First, we want to include the benchmark results to determine how to optimize the prompt. This means we must send the prompt and the previous benchmark results to an LLM for better prompts. We only take failed tasks to shorten the previous benchmarks, categorize them using some set rules, and summarize the categories using an LLM. Secondly, our extracted rules must be anonymous. We should not extract rules specific to websites and task. This can be done by anonymizing the benchmark results. We replace website names and specific filtering options with a mask before summarizing.

To summarize, after a run, take the benchmark results, categorize and anonymize them, summarize, and send the old prompt with the summary to an LLM for an optimized prompt.

The experiments ran with the optimized prompt after one run still showed no significant increase in performance. There was a 3% increase in the success rate of the medium tasks but a 1% decrease in complex tasks. Some tasks were completed with the new prompt; however, the same problem of prompt optimization not producing any significant increase was still present. We hypothesized that this was due to other problems, such as the actual limitation of the model instead of the prompt. One open-ended question is what would happen after multiple rounds of prompt optimization. Unfortunately, we could not test this due to API key issues.

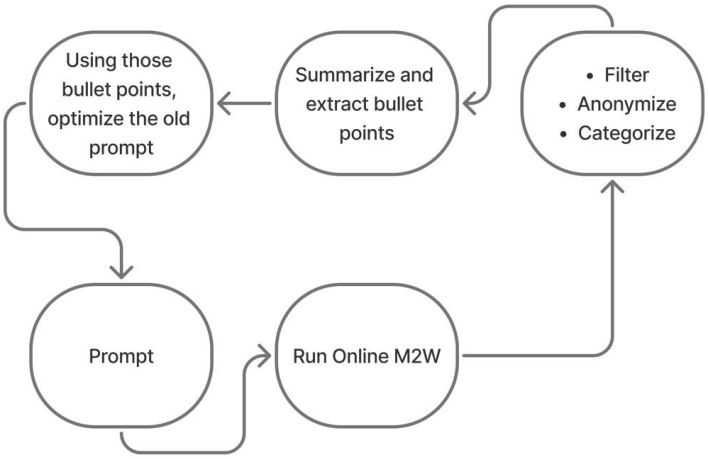

Figure 1: Auto-Prompt Optimization

6 Planner Executer Validator

Prompt optimization did not increase the success we wanted to see, even if some of the extracted rules focused on improving the planning. For example, an added rule was, ”Recheck if you have set all the filtering options correctly before deciding that the task is done”. However, no huge improvement was made. A different approach that is taken by another Web Agent called Agent-E[10] uses another LLM as a planner that reduces complex tasks into multiple more manageable tasks. So we implemented a similar approach.

Our approach takes the task first and sends it to an LLM to be decomposed into multiple more manageable tasks. This approach is very similar to the keypoint identification mechanism of the Online Mind2Web benchmark. It produces small to-dos that must be completed to complete a task. A handler then takes these tasks, manages the browser, and sends Browser-use singular tasks to be done sequentially.

One of the first problems this presented was the case of a subtask failing. We first tried to solve this issue using a memory similar to Browser-use’s. We saved the action history[11] of singular tasks and sent a summary of what has been done to the Browser-use agent.

The experiments show a significant increase in average steps and success rates for medium and complex tasks. There was an increase from a 67.41% success rate on medium tasks to 74.28%. There was a slight decrease in the success rate of hard tasks from 49.32% to 48.82%. After careful examination of all tasks, we noticed different things. The problem of some subtasks not being completed was still an issue. The longer average steps made the key screenshot identification more brittle, leading to some actually successful tasks being considered unsuccessful. An example: For the task, ”Find blue Stitch&Lilo Toys on Amazon.com,” the Agent that received the task: ”Filter for blue” tried to find a filter for blue for 12 screenshots. Online Mind2Web, however, could not identify these screenshots as trying to filter for blue and evaluated the task as failed. However, when the original Agent was benchmarked, Online Mind2Web could evaluate and understand that no filter could be used and thus predicted the label as successful. Some tasks, such as the one mentioned when explaining Online Mind2Web, were also evaluated as failures due to the hierarchical Agent’s different order of subtasks.

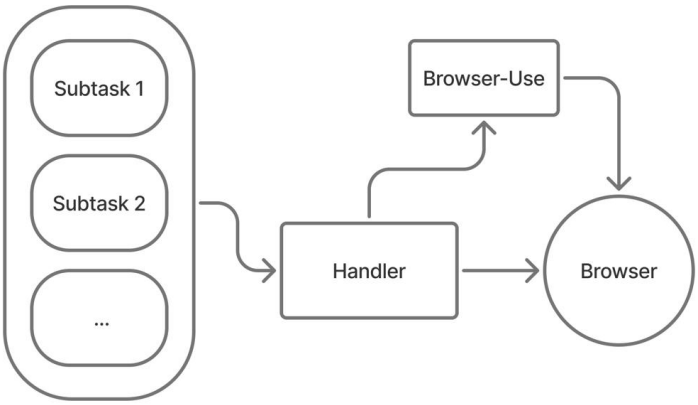

Figure 2: Planner Executer

We introduce a new component: a validator[12], to further improve the performance. For every completion step, an extra question is asked of the LLM to check if the observed state represents the desired state. This validator further ensured that every subtask could be completed as best as possible. A remark on the validator requires more steps to fully validate and ensure that every subtask is completed to the best of its abilities. Due to this, only a select few tasks could be run using the validator. The experiments using the validator did show some improvements in some tasks. It was helpful for the tasks that failed because of a missing subtask.

Figure 3: Validator

However, there are also multiple reasons why a subtask can fail. For example, the desired location we want to arrive might hide behind hidden menus or require scrolling. In these cases, the validator was not a great help, and the Agent kept being stuck in a loop of trying some actions but failing. In these cases, the normal browser-use Agent could initiate a new Google search or try out different things. However, since the memory in our Agent resets between subtasks, there has also been an increase in the Agent being stuck and making no real progress.

Some things that need improvement are not just planning, but also smarter website roaming. For example, an RAG system[13] could help the model look for FAQs in hidden menus or other similar conventions widely used in websites. Planning can also become more dynamic, and the same planner LLM can be used after every subtask to replan ahead, given what has happened before.

7 Results

Model	Medium Tasks	Hard Tasks
Current Top (OpenAI Operator)	58.00%	43.20%
Default Browser-Use	67.61%	**49.32%**
Optimized Prompt	72.14%	47.80%
Planner Executor	**74.28%**	48.82%

Table 1: **Success rates of different Agents on Online Mind2Web tasks.**

We beat the current record held by the OpenAI Agent and the default Browser Use on medium tasks. We didn't break the record on the hard task but we were very close, because these task needs a much higher maximal step limit. In our run we were capped at 35 steps due to API rate limits. Because the state-of-the-art OpenAI run consumed an average of 65 steps per hard task, we project that raising our ceiling to roughly 50–60 steps would enable our system to surpass those remaining scores as well.

In our experiment we focused on the medium and hard tasks of Online-Mind2Web, leaving the easy one aside for complete benchmark run. Our architecture should transfer seamlessly to easy tasks. In addition, those tasks are trivial that invoking a separate validation stage would add computational overhead without materially raising the success rate. It's a trade-off between success rate and latency. To balance robustness and efficiency we thought about a hybrid design in which a binary classifier, embedded in the planner, first labels each incoming task; and then the validator is activated only for instances predicted to be complicated.

Model	Filter Not Applied	Loop Stuck	Max. Steps	Ended Early	Rest	Total
Default Browser Use	22	1	1	5	14	43
Optimized Prompt	18	0	0	1	20	39
Planner Executor	10	3	6	7	12	38

Table 2: **Break-down of failure modes for medium Online-Mind2Web tasks.**

The failure-mode analysis for medium tasks revealed that most of their errors stem from the agent not applying the correct filters, this is especially for the default Browser-Use baseline, where 22 of 43 failures (51 %) fall into this category. By contrast, our Agent with Optimized Prompt variant eliminates both the Loop Stuck and Max. Steps problems entirely, leaving only a

single early termination and other issues.
However, our Agent with Planner Executor variant shows the fewest filter mistakes but still hits the step ceiling more often, suggesting that the planner helps with filtering problems but could lead to confusion of the agent where it can get stuck because its current state deviates significantly from the Plan given by the planner. In addition, the higher max steps tasks show the overhead added with the addition of the planner which leads to a higher number of steps needed. Overall, these figures confirm that raising the step limit is most likely to improve the remaining performance of the hard tasks from Table 1.

8 Future Research

During our practical course, we experimented with several other ideas that we found promising for future work and worth exploring further.

- **Sitemap:** In addition to GUI screenshots and the HTML DOM, we experimented with augmenting them with each website's sitemap. A sitemap is an XML file that lists a site's URLs along with metadata; it is primarily used by search engines and web crawlers to improve indexing and SEOs. It is usually available at /sitemap.xml or linked from robots.txt. This worked very well for small sites with compact sitemaps since the agent reached the goal page faster because it knew where to go. We observed a similar pattern when the agent was denied by Cloudflare and in those cases, instead of navigating the site directly, it searched for the task on Google. However, for large sites with very big sitemaps we faced challenges parsing them. In addition, this approach does not work for sites that do not provide sitemaps.

- **RAG-Similar Architecture:** We experimented with a RAG-style architecture as well where we embedded previous solved tasks into a vector database (ChromaDB) and, in the metadata, stored links to the corresponding trajectory (JSON) and screenshots, which we saved in database (MongoDB). For a new task, we embedded it and computed cosine similarity against the tasks in the vector database. For the top match with at least 80% similarity, we retrieved their trajectories and screenshots and provided them to the agent from the get-go. In the tasks we tested, this approach yielded promising results since for similar and close tasks the agent achieved the task in fewer steps.

- **Token reduction:** We worked extensively on reducing token usage, since we believe that it will lower latency of the agent and the costs needed. First, we tried different image compression algorithms but this did not lead to good results, because the agent struggled much more to recognize GUI elements. However, we achieved good results by converting the screenshots into a video. We found that after about 30 steps it is more efficient to send a video instead of individual screenshots to the LLM. This led to a 12% reduction in tokens. To do this, we stitched the screenshots into a video and indexed each screenshot to its corresponding frame, then we compressed the video using FFmpeg to H.264 format, with that we provided the LLM with the compressed video plus the frame index. This approach only works with LLMs that support video modality.

- **Model finetuning:** We also experimented with specialized models trained on GUI elements and task trajectories. One model in particular Seed1.5-VL [14] from ByteDance showed very good results especially on hard tasks. However, its weights are closed-source and only API access is available, so we could only test it on few selected medium and hard tasks. We believe this is the most promising path forward, since trajectory information and knowledge of GUI elements are pushed into the model's weights, allowing us to use smaller specialized models.

9 Outlook

First things first, different models should be tried out. Our most significant improvement was changing our base model from Gemini 2.0 flash to Gemini 2.5 flash. The improvement of the model can be seen as an improvement in both the understanding of the observable state and the planning of an action to take. This observation tells us that, with better models, web agents will also automatically improve over time.

Secondly, planning must be done smarter, and more knowledge should be introduced in this process. Using RAG pipelines, learning website conventions can increase the accuracy of the planner LLM, making the correct plans, instead of just identifying key points.

In this decision-making process, models have also used tree search given multiple possible actions. This can decrease the likelihood of an action disturbing the process made before it. One possible way is to also set checkpoints that the model can return to and try again. Both methods allow for better website roaming and let the model take more optimal steps.

One other research topic is planning with human feedback. Human feedback and directions can be incorporated during the website roaming to produce better results. The agents would be stuck less, and the feedback can point them in the right direction.

These are some possible improvements to Browser-Use that can improve its performance in more challenging environments. However, better research into Online Mind2Web tasks is also necessary. Some tasks, such as ”Go to chess.com and solve the first puzzle,” do not necessarily represent the tasks that can be expected from a web agent. The benchmarks should realistically represent the real use cases expected from web agents, such as ”Content extraction” and ”Automation of daily tasks”. Even default Browser-Use can achieve good performance in these tasks.

10 Conclusion

Agentic AI is one of the hottest topics in AI research right now. Allowing AI to interact with its environment can easily automate most daily tasks, help people with disabilities, optimize workflows, etc. Web Agents symbolize a great leap forward toward truly assistive software. Our experiments show that, even with a modest amount of task-specific engineering, today’s LLMs can already handle 75 % of medium-difficulty, multi-step web workflows and just under 50 % of ”hard” ones. Those numbers will rise as foundation models continue to improve. However, architecture matters too: hierarchical planning, explicit state validation, and tighter feedback loops between perception and action all contributed measurable gains. In the future, pairing better models that incorporate external knowledge with continuous fine-tuning and better benchmarks will push these models to be usable in real-life situations in various contexts.

References

[1] McKinsey & Company. Responsible AI: Realizing and managing value in artificial intelligence. Technical report, McKinsey Global Institute, 2025.

[2] Shunyu Yao, Sherry Zhao, Dian Yu, Karthik Narasimhan, Yuan Cao, and Weipeng Chen. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[3] GeeksforGeeks. Limitations of selenium webdriver. `https://www.geeksforgeeks.org/limitations-of-selenium-webdriver/` URL visited on 8th August 2025.

[4] browser-use contributors. browser-use: Web-ui framework for ai agents. `https://github.com/browser-use/browser-use` commit `2f1c9b4`, accessed 8th August 2025.

[5] Tianyi Xue, Yifan Liu, Zhihan Guo, Xinyi Huang, and Diyi Yang. Online-mind2web benchmark. `https://github.com/OSU-NLP-Group/Online-Mind2Web` URL visited on 8th August 2025, 2025.

[6] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[7] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A survey of prompt-based learning. *Journal of Artificial Intelligence Research*, 78:1112–1170, 2023.

[8] Minseo Kwon, Kyusong Kim, and Sungjin Park. Stableprompt: Automatic prompt tuning using reinforcement learning for large language models. *arXiv preprint arXiv:2410.07652*, 2024.

[9] Teven Zhao, Albert Webson, Kevin Ho, Eric Wallace, Asli Celikyilmaz, Keisuke Boyer, and Noah A. Smith. Automatic prompt engineer: Improving zero-shot performance of large language models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 26046–26066. PMLR, 2023.

[10] Tamer Abuelsaad, Soumya Koley, and Eric Jonas. Agent-e: From autonomous web navigation to foundational design principles in agentic systems. *arXiv preprint arXiv:2407.13032*, 2024.

[11] Yujia Shen, Chen Li, Jiashu Song, Boxiao Wen, and Stefano Ermon. Memory in language agents: A survey. *arXiv preprint arXiv:2311.00066*, 2023.

[12] Aman Madaan, Zhangir Tu, Amir Yazdanbakhsh, Sungjin Park, Nitish Shirish Jain, William Fedus, and Colin Raffel. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

[13] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and Sebastian Riedel. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 9459–9474. Curran Associates, 2020.

[14] Dong Guo et al. Seed1.5-vl technical report. *arXiv preprint arXiv:2505.07062*, 2025.