Team project Report

Browser Interaction - Creating google forms using Browser-use

Giacomo D'Andria(2053226)
Jiyeon Lee(1936372)
Victoria Lahr(2189513)

May 24, 2025

1 Introduction

This project focuses on the interaction between large language models (LLMs) and web browsers, aiming to automate the creation of Google Forms such as quizzes, surveys and registration forms in the Google Chrome Browser.

We investigated browser-use, the leading open-source web agent project by now. Browser-use is based on the browser-automation protocol playwright, but acts as an abstraction layer, enabling LLMs to use the browser autonomously. The library gives the LLM access to manual playwright commands like `click()`, `fill()` or `read()` and manages navigation, memory and state within the browser. There are several benefits that come along with using browser-use. Unlike traditional browser automation tools like standalone Playwright or Selenium, which require manual scripting and concrete error-handling whenever encountering something unexpected, browser use allows an agent to interact with the web-interface of our own browser using natural language prompts. This makes it particularly powerful in fast changing or unstructured web environments, providing flexibility and human-like reasoning when encountering a problem.

Also, by operating within the user's own browser, login states, sessions and cookies can be accessed and used by the agent. This simplifies authentication and allows the agent to act as if it were the user - navigating and interacting even with unknown web interfaces and accessing real, contextual information directly.

As it is advertised to be highly customizable and ideal for automating repetitive tasks, we chose to test its capabilities with a concrete, measurable use-case: the automated creation and editing of Google Forms via natural language instructions. For this, we agreed on three different scenarios. The creation of simple quizzes, for example about the English language, generating a survey about a given text or topic and setting up an event registration form. The respective content of the google form should also be generated by the LLM and then be transferred into Google Forms by browser-use interacting in our own browsers. All three scenarios have slightly different requirements and should thus thoroughly test the real-life usability of the framework

2 Use-case & different apporaches

2.1 English Assesment forms

This section focuses on the approach that was used in the creation of a Google Form to create an English language form that assesses a student's knowledge of the topic set as input to the system (later limited to a predefined topic to measure the success-rate).

Four different answering modalities were taken into consideration for this task, namely the four types of question chosen are (1) multiple choice, (2) short answers, (3) paragraphs, meaning long form input text from the user, and (4) checkboxes, which lets the user choose *all* answers that apply to a specific question.

Each of these types of questions presupposes the agent to know how to handle the creation of such question-type, but due to the poor initial performance in creating the different types of questions, a description of what the goal for each question-type is was

then provided in the prompt, in the form of more precise instructions that the agent was then able to follow more closely. In the case of success measurement, questions and answering possibilities were pretabulated in the prompt itself.

2.1.1 Technical setup and challenges

The model that was used to carry out this task was GPT4o-mini from OpenAI. The temperature was set to 0 to avoid it deviating much from the tasks that it was supposed to carry out, though this didn't completely avoid the natural tendency of the agent to take different paths from time to time, even when all conditions were seemingly unchanged.

Among the various options that the Browser-use library offers to allow interaction with the browser, that of using prompt swapping was chosen. In prompt swapping, the whole system is only using a single agent to carry out all tasks, but the prompt is subdivided into smaller sub-prompts in order to focus its attention more closely. Initially, just using a single (longer) prompt was also tested, though the performance was very poor, as the agent couldn't really prioritize what parts of the task to focus on first, thus this method was not employed. Moreover, in order for the prompt-swapping technique to work, the context of the browser needs to be kept alive throughout the different tasks, allowing the agent with the swapped prompt to carry out the same task without incurring into errors. To achieve this, the `_force_keep_context_alive` flag was used. In the newest version of the library (version 0.1.41 at the time of writing), such flag is just named `keep_alive`, though throughout the development of this task the library stopped working properly and the agents couldn't carry out actions anymore, even though they could correctly identify the elements in the page as well as what action *should* have been carried out. For this reason, this task was completed using version 0.1.40 of the library, which ended up working well and required the former flag to keep the context of the browser alive.

All subsequent steps of this task were completed without using vision capabilities, which, though they should supposedly improve performance, in this case seemed to make no difference, other than increasing the cost.

After having navigated to `forms.google.com`, having opened a blank form, and having entered a title and description, a randomizer selects one of the four question types for the agent to add to the form. Each description is very specific in what the agent is supposed to do, in order to avoid having the agent choose autonomously what direction to take. The prompts were made as detailed as possible throughout the observation of multiple runs, which led to the identification of what the main pitfalls were, and thus the corrections needed. Due to the low necessity of having the output be structured in the same way every time, few-shot prompting was avoided, focusing on adding as many details as were deemed necessary to keep the agent focused on small, simple tasks without having to make guesses about what action to take. Forcing a specific structure in the outputs was in fact observed not to add any meaningful improvement to the performance.

2.1.2 Results for this task

The first task the agent was assigned to do was filling out the form title and description based on the topic that was chosen by the user through CLI. This part worked well most of the times, though sometimes the agent got confused and inserted some generic title and description for the form (e.g., *My google form* and *This is a description for my google form*), which then had to autonomously correct once it figured out that the topic was not correct, and in that case it changed it accordingly (e.g., *Present simple form* and *This form tests the knowledge of the Present Simple tense in English*).

Overall, the agent was sometimes able to carry out tasks correctly, while other times it would get some steps wrong, thus hindering on the success of the task by, e.g., completely ignoring what it was supposed to do and just writing the question or the options in places where it shouldn't have, or by trying to publish the form even though it was explicitly mentioned that it shouldn't have been published.

All things considered, even though the approach of swapping prompts *can* work well, it is not guaranteed that the correct result will be obtained. A full summary of the results is reported in Table 1.

2.2 Survey Form

In the second approach, we implemented a two-agent setup designed to assist in the creation of surveys for use in bachelor's and master's theses.

2.2.1 Technique

1. **Browser-Use & Browser-Config :**
   For this use case, version 0.1.41 of browser-use was employed. It is important to note that, as this is an open-source project, new versions are released frequently and may introduce relevant changes. To integrate the agent with a real Chrome instance, the path to a local `chrome.exe` and a specific user data directory were explicitly provided. To ensure stable and predictable browser behavior, and to avoid interruptions during automation, several flags were included in the `launch_args` configuration:
   - `--profile-directory=Default`: Ensures that the agent uses the default Chrome profile, allowing access to pre-existing sessions and cookies.
   - `--no-first-run and --no-default-browser-check`: Suppress pop-up dialogs and onboarding screens that previously disrupted the agent's flow during initial test runs.

   This setup was tested using both GPT-4o and GPT-4o-mini. As no significant differences in output quality were observed, all subsequent evaluations were carried out using GPT-4o-mini, primarily due to its lower token consumption.

2. **Browser-Use agent:**
   The Agent class is the most important class when using browser use and follows a very simple setup.

```python
from browser_use import Agent
from langchain_openai import ChatOpenAI

agent = Agent(
    task="Search for latest news about AI",
    llm=ChatOpenAI(model="gpt-4o"),
)
```

Figure 1: Sample initialization of an `Agent` object

The parameters task and LLM are mandatory for initializing an agent. These can be supplemented by several optional parameters that influence both the agent's behavior and browser functionality. For instance, the `use_vision` parameter enables or disables visual processing capabilities, while `save_conversation_path` allows the full conversation history to be stored, which can be particularly helpful for debugging and analysis purposes.

2.2.2 General idea

In this configuration, the first agent received a text on a specific topic along with a detailed prompt specifying the structure and content of the desired survey. This included the required number of questions, the use of varied question types, and the inclusion of general demographic questions about the participant. As this agent's task did not involve web interaction, its browser setting was explicitly disabled by setting it to None.

The second agent received the survey generated by the first agent in the form of a string. Its task was to open the Google Forms website and create a new form. It was instructed to select an appropriate title based on the content of the survey and replace the default title accordingly. For each question, the agent was required to insert the correct question text, select the appropriate question type (e.g., multiple choice, short answer, checkboxes, etc.), and, where applicable, provide the corresponding answer options. Each question was to be marked as **Required** before proceeding to the next one. This process was to be repeated for every question included in the survey generated by the first agent. After all questions had been added, the agent was instructed to close the browser without submitting the form. We added these instructions as one big prompt, including the two variables {generated_task}, containing the survey generated by the first agent and {form_url}, containing the link to the google forms website.

2.2.3 Results & Evaluation for this task

Both agents were executed multiple times, with continuous adjustments made to their prompts and parameters in response to observed issues. The high cost of running the LLM model was striking. Generating the form with Agent One required approximately 3,500 to 4,000 tokens. In contrast, inserting the survey into Google Forms consumed around 228,000 tokens per execution. After several iterations, the prompting strategy

for both agents was shifted from zero-shot to few-shot prompting. While this change significantly improved the quality of the surveys generated by the first agent, no corresponding performance improvement was observed for the second agent, which was responsible for interacting with the browser. In fact, various technical issues persisted throughout its operation. The most frequent problems occurred during the insertion of survey questions. Specifically, when more than two questions were involved, the agent often misassigned question types or incorrectly handled answer options. Common issues included overwriting or omitting answer choices, failing to assign a question title, or skipping questions entirely.

Also, when instructed to change the title of the form to a more fitting name, the LLM inserted a new title but did not delete "Untitiled form" (see Figure 2). Notably, in some cases, the agent recognized its own mistakes and attempted to correct them by going back a step. However, in other cases, it proceeded under the false assumption that the task had been completed correctly.

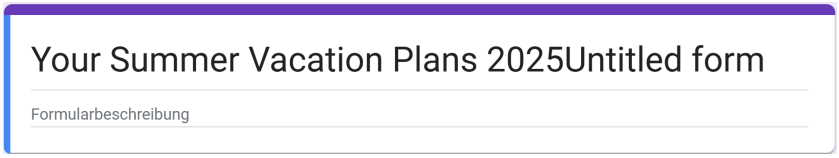


Figure 2: Example of a form with the "Untitiled form" title not deleted

A critical issue arose midway through the project with the release of Chrome version 136, which introduced a security restriction preventing automation tools from controlling Chrome via the Chrome DevTools Protocol (CDP) when using the default user profile (GitHub Issue #1520). This change caused a significant disruption and was only resolved by configuring a separate debug version of Chrome and manually logging into the required user profile before resuming the agent's task.

Additionally, language settings posed a challenge. When the Google account's language preference was set to a language other than English (e.g., German), the agent often failed to locate UI elements such as question type selectors or buttons unless the translated terms were explicitly included in the prompt. For example, the German term "Multiple-Choice-Frage" did not match the English "Multiple Choice" as expected, leading to failed interactions despite the semantic similarity.

To enable a focused evaluation of the browser interaction component and to systematically assess the issues encountered in this use case, Agent 1 was replaced with a standardized JSON file. This file contained five predefined survey questions, each representing a different question type and including distinct answer options. This controlled input allowed for consistent testing of Agent 2's behavior. The results of this evaluation are presented in Section 3.

2.3 Event Organizations Form

In the third use case, we deployed a single agent setup and employed GPT-4o-mini and version 0.1.41 of the browser-use to create and automate Google Forms for event-related tasks such as registration, or feedback.

2.3.1 Technical setup

We implemented an automation process to open and verify access to a specified Google Form using browser automation component, which is Controller. The Controller serves as the central hub for registering and executing automation actions, enabling streamlined workflow management through predefined action functions. The core functionality is demonstrated in the "open_google_forms" function, which automatically navigates to a target Google Form URL. The process begins by retrieving the currently active browser page through browser.get_current_page(). If the current page URL differs from the target Google Form URL, the script performs navigation using the goto() method and waits for complete page loading via wait_for_load_state(). To ensure proper access verification, the system employs an is_google_forms() validation function that confirms whether the current page URL matches the expected Google Form pattern. If validation fails, indicating either access permission issues or navigation problems, the function returns an ActionResult containing an error message. Conversely, when validation succeeds, the system returns a success ActionResult confirming that the Google Form has been successfully opened and is accessible.

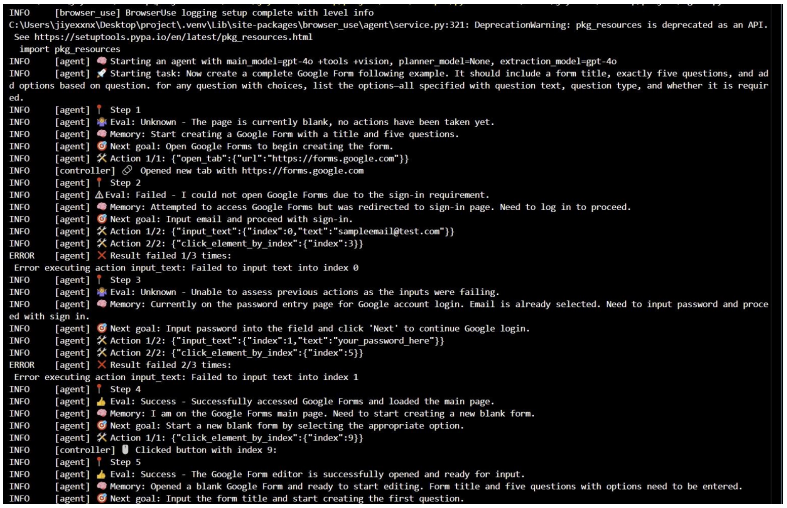


Figure 3: Sample thought process of an agent

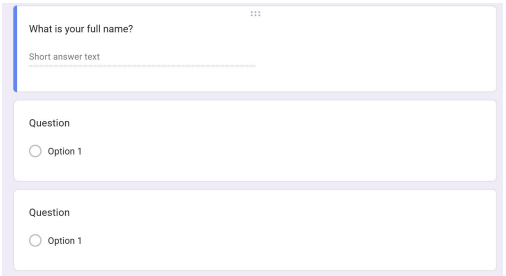
The Google Form automation is implemented using two asynchronous functions, which provide browser-based control over the interface and demonstrate the core capabilities needed to create form questions. The first function is the question Type Selection Function, specifically the "set_question_type_if_needed" function, handles setting question

types other than 'multiple choice'. This function is crucial because 'multiple choice' is the default question type in Google Forms, so it is necessary to explicitly change it when a different input format is desired. It achieves this by clicking the question type and selecting the appropriate type based on the question. The second function is the main question creation function, "add_question_to_form", orchestrates the entire question creation process. This involves a series of steps: first, click the "Add Question" button; next, fill in the question title; then, it sets the question type such as short answer, paragraph, date, time, multiple choice, checkboxes, and dropdowns. After that, it configures any necessary options or choices, and finally, it sets properties such as help text or marks the question as required.

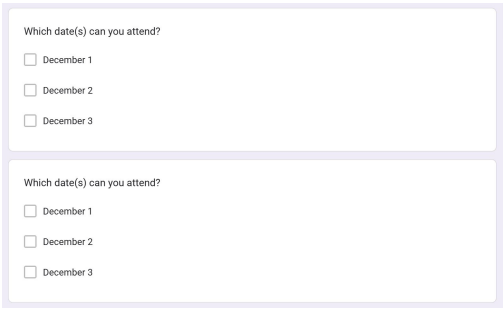
2.3.2 Result for this task

The agent generally demonstrated a variable level of success in task execution. There were instances where it followed instructions and achieved the desired outcomes, showing its potential for reliable automation. However, its performance was not consistently performed while other times it would get some steps wrong. Two major issues were encountered during the experiments. First, the flow of form interaction was interrupted by agent inducing users to log in repeatedly, which hindered automation. This problem is not related to GitHub Issue #1520.

Second, the problem occurred during the insertion of the question and the options. The agent frequently generated empty questions or incorrectly assigned new questions to the previous ones. In some cases, it assigned the question, but not options, or the agent sometimes duplicated the same questions and options. In addition, in the options part, the agent suddenly stopped creating options. The results of this evaluation are presented in Section 3.



Case 1 : Generating empty questions



Case 2 : Duplicated the same questions and options.

Case 3 : Incorrectly assigned new questions to the previous ones

Case 4 : Only assigned question with no options

3 Evaluation

The final results for the various tasks are not as promising as they were initially thought to be. Although the models and techniques *can* perform the intended actions, the extent to which they can carry them out effectively it limited. The overarching theme of all tasks that we set out to achieve is that the agents cannot be fully entrusted in achieving the correct results with complete accuracy.

Moreover, due to the very rapid developments of the field and the tools used in this project, many problems that were encountered could have been avoided by having more mature tools to work with. For example, the browser-use library is being developed on a daily basis and because of its many moving pieces, some parts of it sometimes break compatibility. We have encountered many instances of this fact throughout the development process. For example, upon trying to re-run the same code developed a few weeks prior, we noticed that using our own browsers was leading to the agents being unable to carry out the tasks. This turned out to be caused by an update of the chrome app, which then led us to using the blank chromium instance which required manual login on every run in order to access the form, and this was cumbersome and slowed the development process. Additionally, the documentation for the library is at times outdated, due to newly added components or modifications. For this reason, it was often necessary to look through the code on the GitHub page rather than use the available docs. At times, this also caused considerable slowdowns in the development.

The state of the art in form-related actions is not quite focused on the actual creation of the form itself, rather with filling it out, especially when embedded on some website. That task is notably easier to be carried out with respect to the one we set out to achieve in this project, and our agents would probably be sufficiently good at that. When it comes to the actual interaction with a webapp such as Google Forms though, the agents fall short and are not quite as advanced at interacting with complex user interfaces due to the many moving elements of the pages, as well as the very similar names that those elements can have.

Given the limitations of the library and the technology, there was no instance in which a form was fully created without any issue. In order to better evaluate each portion of the tasks, we set out to measure the accuracy we obtained over 5 different runs with 5 questions each. In the case of surveys and event organization forms, 5 additional runs with few-shot prompting were also tried out, though the performance only seemed to worsen and lead to a decrease in accuracy. All other attempts instead lead to similar results.

The metrics are summarized in Table 1, and some of the commonalities are the following. All techniques led to a pretty consistent tendency for the models to add too many questions, often leaving them blank. The titles of the questions were most consistently written correct, whereas, depending on the technique, the question-type got selected correctly more often using the prompt-swapping and controller techniques. The options to the answers that require them were mostly not inserted correctly, often inserting only some of the options that were supposed to be added, while missing others. Question that didn't require any options performed best (e.g., paragraph questions), as the title

itself often got inserted correctly and there was no need for the actual choices. Table 1 shows all other metrics into more detail.

Table 1: Evaluation metrics for different prompting techniques and form types

Metric	Run 1	Run 2	Run 3	Run 4	Run 5
Inserted new Form Title					
Surveys (zero-shot)	Yes	Yes	Yes	Yes	Yes
Surveys (few-shot)	Yes	No	Yes	Yes	Yes
Event Organization (zero-shot)	Yes	Yes	Yes	No	Yes
Event Organization (few-shot)	Yes	Yes	Yes	Yes	Yes
English assessment (zero-shot)[1]	Yes	Yes	Yes	Yes	Yes
Deleted 'Untitled Form' Title					
Surveys (zero-shot)	No	No	No	No	No
Surveys (few-shot)	Yes	No	No	No	No
Event Organization (zero-shot)	No	Yes	No	No	No
Event Organization (few-shot)	Yes	No	Yes	No	Yes
English assessment (zero-shot)	Yes	Yes	No	No	Yes
Amount of questions (out of 5)					
Surveys (zero-shot)	9	X	10	8	16
Surveys (few-shot)	10	4	1	7	X
Event Organization (zero-shot)	3	6	7	5	8
Event Organization (few-shot)	14	5	5	5	2
English assessment (zero-shot)	10	7	6	9	12
Correct question title (out of 5)					
Surveys (zero-shot)	3	X	3	4	3
Surveys (few-shot)	2	4	1	3	X
Event Organization (zero-shot)	3	4	4	5	5
Event Organization (few-shot)	4	4	3	4	1
English assessment (zero-shot)	3	2	4	4	2
Answer-options (% of total options)					
Surveys (zero-shot)	20%	6%	6%	20%	20%
Surveys (few-shot)	0%	33%	0%	13%	0%
Event Organization (zero-shot)	20%	20%	30%	50%	30%
Event Organization (few-shot)	20%	50%	40%	70%	20%
English assessment (zero-shot)	33%	16%	50%	16%	16%
Question marked as required (out of 5)					
Surveys (zero-shot)	2	0	0	2	0
Surveys (few-shot)	1	4	0	1	X
Event Organization (zero-shot)	0	0	0	1	0
Event Organization (few-shot)	0	0	0	1	1
English assessment (zero-shot)	N/A	N/A	N/A	N/A	N/A
Correct question type (out of 5)					
Surveys (zero-shot)	1	1	1	2	1
Surveys (few-shot)	1	2	0	0	X
Event Organization (zero-shot)	2	2	2	4	4
Event Organization (few-shot)	2	4	3	4	2
English assessment (zero-shot)	1	3	2	2	3

4 Conclusions

This project successfully investigated the feasibility of enabling an LLM to create Google Forms through automated browser interactions. However, while the core functionality was realized, the implementation revealed some challenges inherent in current browser automation technologies.

The main challenge encountered was the instability of the browser use library, which exhibited a lot of bugs that complicated the development process. Throughout our project, we found directly how quickly the browser use library was evolving, with new versions being published every few days over just two weeks. This rapid development cycle indicates a changing underlying technology, where ongoing updates may create both advancements and instabilities. This type of rapid iteration may be beneficial in the long term but poses challenges for stable implementations in the short term.

Given the dynamic state of browser automation technologies and the promising trend of LLM capabilities, repeating this project in one year would likely yield substantially different results. The expected evolution of both browser automation platforms and LLM reasoning functions suggest that future iterations could overcome many of the current limitations, potentially enabling more reliable and intuitive form creation processes.

In the end, our project proved that our goal worked, but it also showed us how much the technology still needs to mature before it's truly ready for everyday use.

References

GitHub Issue #1520 : https://github.com/browser-use/browser-use/issues/1520
Browser-Use : https://browser-use.com/

Contribution

Name	Task
Giacomo D'Andria	Usecase 1 (English Assessment form), Evaluation
Victoria Lahr	Usecase 2 (Survey Form), Evaluation
Jiyeon Lee	Usecase 3 (Event Organizations), Evaluation
Rajesh Mahara	No Contribution

Declaration of Used AI Tools

Tool	Purpose	Where?	Useful?
ChatGPT	Rephrasing	Throughout	+
ChatGPT	Code Explanation	general understanding	–
ChatGPT	Related work hallucination	Most of bibliography	++

Our Github repository : https://github.com/jiyexxnx/Browser-interaction

Mannheim, 24.05.2025