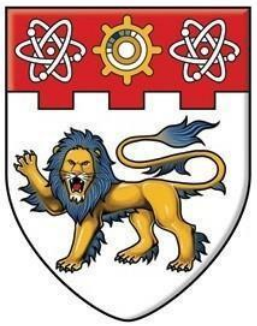
College of Computing and Data Science



NANYANG TECHNOLOGICAL UNIVERSITY

SC4052 CLOUD COMPUTING

Assignment 2
(Choice 2: Study basics of API in Github and Code Search)

Choh Lit Han Owen
(U2221094K)

Table of contents

1. Introduction

The explosive growth of open-source software and the proliferation of code repositories on platforms like GitHub have created both opportunities and challenges for developers [1]. While access to a vast amount of code can accelerate development and promote collaboration, finding the right code for a specific task can be time-consuming and inefficient. This report describes the development of a SaaS tool designed to mitigate this challenge by providing a robust and intuitive code search interface. The tool leverages the GitHub Search API to enable users to quickly locate relevant code snippets based on keywords and language. Furthermore, the report investigates the potential of integrating Large Language Models (LLMs) to provide advanced features such as automated code documentation generation and other code analysis tasks. This integration aims to not only improve code discoverability but also to enhance code understanding. The following sections detail the design, experiments and implementation of the SaaS, highlighting the potential of LLMs to aid with code search and analysis.

2. Understanding GitHub API

This project began with studying the structure and usage of the GitHub REST API, focusing on authentication, pagination, rate limits, and endpoint behavior. For example, search queries had to be carefully crafted using qualifiers like in:file and language:typescript to filter relevant results, revealing the importance of understanding GitHub's indexing limitations (e.g., .tsx files not included in language:typescript). Additionally, the project explored how underlying endpoints like "Create Tree," "Create Commit," and "Create Pull Request" work together to simulate Git operations via API. This foundational understanding was critical in building features like automatic PR creation and effective code search.

3. GitHub API Integration

Making use of the Github REST API documentation [2], the project explored how to make use of the APIs. Figure 1 to 3 is a simple prototype created to understand the parameters to use to search code in repositories owned by me [2]. Similarly, the function can be adjusted to search for commits created by me in a specific repository by swapping the url and removing extra parameters [3] as shown in figure 4 and 5. Similar code is also used to list repositories and is used to populate the dropdown list on the "GeneralInfo" UI.

Constructing the search query was challenging due to inconsistencies such as filtering by "language:typescript" only returns ".ts" files, not ".tsx". Through trial and error, I found filtering by file extension to be more reliable. To support various file types, users are provided with a custom input field (see figure 9). Pagination is implemented to retrieve all matching files [4] (see handleSearch in CodeSearch.tsx). Code content is then fetched using the GitHub Octokit SDK [5], [6] and decoded from base64 [6] (see figures 10 to 12).

Another feature of the app is the submission of pull requests to make uploading large or numerous updated files simpler. To achieve this, the steps taken and the corresponding APIs are as follows:

1. Obtaining the default branch of a repository using “Get a repository” [7]
2. Query for the latest commit and tree object in that branch using “Get a branch” [8]
3. Creating tree objects to store the new files using “Create a tree” [9]
4. Creating a commit with the new tree object using “Create a commit” [10]
5. Creating a new branch for the pending commit to live on using “Create a reference” [11], otherwise the new object created will not show up on the Github website and users can only access it using the link returned by the API when creating the commit.
6. Create a pull request to merge the branch above into the default branch using “Create a pull request” [12]

The last integration is the creation of a fine grained token so that the SaaS can access and interact with the APIs described above. Permissions of the token are shown in figure 13. Links to source code and application are in the appendix under “source code and running application”.

4. UI Design

This section outlines the application’s user interface following the finalization of core features.

Figure 14 shows the front page where users input credentials, username, and repository. A dropdown is used instead of a text box to reduce spelling errors and ensure the GitHub token has the permissions to view the intended repository.

Figure 15 shows the CodeSearch UI, which integrates the GitHub Search API. During testing, some files temporarily disappeared after new commits but reappeared after a short delay which may be due to indexing by the API. To help users browse large result sets, a “Generate description” button powered by Google Gemini summarizes each file, reducing the need to manually open them to look for what they need.

Figure 16 displays the CodeEdit UI, which handles most of the LLM features. Users can set the model “temperature” [13] and interact with selected files from CodeSearch. A caching system reduces fetch time for faster iteration of outputs. Similar buttons were color-matched to guide users to similar actions and text is included to clarify which actions apply to selected files only.

5. Prompt Engineering & LLM Functions Discussion

This project uses prompt engineering to guide LLMs in generating accurate, contextual code explanations to support developer productivity. Since LLMs are sensitive to phrasing, prompt design was refined iteratively to improve output quality [14], [15].
In this project, prompt engineering is used to generate relevant outputs for code snippets retrieved via the GitHub Search API. Prompts include file metadata (e.g., name, path, repository

name) and file content from GitHub Search API results to create meaningful explanations. Experiments explored various strategies, simple descriptions, metadata-aware prompts, and role-based framing. Given the system's reliance on zero-shot prompting [15], clarity and specificity of prompts were especially important.

5.1 Generate code description

In the application, LLM is used to analyze code retrieved via the GitHub Search API. Prompts are dynamically constructed using file content and metadata to obtain relevant descriptions. Table 1 displays the iteration of prompts, the outputs and the observations made before arriving at the final prompt that is currently used in the application. These series of prompt iterations were conducted to optimize the description generation feature. Early prompts produced detailed but overly long explanations, while later refinements successfully generated concise, two-to-three sentence summaries suitable for UI display. Adding contextual metadata (e.g., search query and file path) did not degrade output quality and was retained for added specificity.

Figure 22 shows the final change to include a default to "all code in the repository" when users do not search for any specific file which is likely to occur since the features in "CodeEdit" are better when obtaining all files for full context. This change did not make breaking changes to the generated description (refer to figure 23 and 24 for the description before and after the change).

5.2 AI generated code experiment

This experiment investigates whether an LLM (Google Gemini) can accurately distinguish between human and AI-generated code. Two sets of code, a React login page and a user service in Golang, were written manually with minor AI contributions (refer to figure 25 and 26) and then recreated using ChatGPT (accessed on April 16, 2025). Gemini was prompted to evaluate each file to assess the likelihood of AI generated code. Please refer to the appendix "Figures for AI generated code experiment" for more information.

The results showed mixed accuracy. Gemini correctly identified the AI-generated files but also misclassified the human-written versions as likely AI-generated. This highlights key challenges in distinguishing AI generated code, particularly when both follow standard libraries and best practices. Misclassifications were likely due to the LLM focusing on superficial traits (e.g., naming conventions or consistent formatting) and lacking deeper contextual understanding. The presence of real-world identifiers (AWS domain) slightly improved accuracy, but cannot serve as a reliable indicator. Correct classifications may have stemmed from pattern matching typical of AI-generated content [16], and a lack of nuanced design or customization.

The experiment highlights the limitations of LLMs in detecting AI-generated code, especially without fine-tuning or large sample sets. As a result, this feature was excluded from the final SaaS, pending more robust detection techniques involving project-level context and semantic reasoning.

5.3 Generating Code Documentation and Repository README

This section explores using LLMs to generate clear, beginner-friendly documentation and README files without directly embedding source code into them to be concise. The experiment files are included in the appendix “Generating Code Documentation” and “Generating Repository README”. Initial prompts (see generate documents 1.md) resulted in overly long outputs due to full code inclusion. Adjusted prompts excluded code and improved formatting but often lacked technical depth. A structured output format, dividing output into “How-To Guides” and “Reference Guides”, significantly enhanced clarity and usability (refer to generate documents 3 to 6.md). Iterative prompt tuning added specific instructions for markdown styling, inclusion of file paths, and user notes (e.g., avoiding .env file commits), with improved consistency when temperature was set to zero (generate documents with 0 temperature.md).

Subsequent README experiments applied similar principles. Early versions had formatting issues or unneeded sections (e.g., “Contributing”) and duplicated lines (see generate readme 1 to 4.md). Later prompts introduced system instructions and reworded guidance to improve compliance (generate readme 5–10.md), ensuring a high-quality README template tested across repositories.

Key Takeaways: Structured prompts, model temperature control, and explicit formatting rules led to more reliable outputs. These improvements have been incorporated into the final SaaS.

5.4 LLM Function Calling

To automate README generation and pull request creation, this project leveraged the function calling capability of the Gemini API. A manual trigger (figure 28) was implemented for controlled testing. While third-party tools like Model Context Protocol exist [17] , the built-in function calling was chosen for simplicity [18], and direct support in the Gemini SDK [19]. Experiment files are provided in the appendix “LLM Function Calling”, prompt iteration is included in table 4.

Initial attempts to include full repository contents in prompts resulted in errors (see “MALFORMED_FUNCTION_CALL response.json”)[20], which experimentation suggests is due to high token count. This was mitigated by limiting context to a subset of files and shifting file content to chat history. However, this sometimes yielded overly generic READMEs (see generate function 2 to 5.md). Repeated retries by adjusting temperature had limited success.

To improve reliability, the approach shifted to requesting function calls in JSON format. This made parsing and Pull Request (PR) submission more robust (see “generate with function json experiments”). Despite the model wrapping JSON in markdown (e.g., ```json), output could be post-processed to extract valid calls. Further refinements included using personas, temperature control (0 during testing, validated with 1 after testing), and adding success messages when rendering on the UI. The method was validated on larger repositories with successful submissions (e.g., PR #14 to 15, see generate function json 5 to 7). Ultimately, JSON-based function calling proved to be a reliable method, now fully integrated into the SaaS platform.

5.5 Comment Checker and Custom Prompts

I observed that modern IDEs do not verify comment accuracy or suggest improvements. To address this, a feature was added to detect outdated or missing comments and suggest replacements. Due to token limits, the system processes one file at a time, returning structured JSON (fileContent, explain) for PR automation. Due to the context sensitive nature of this feature, it made subtle errors hard to catch, though later iterations seemed to improve accuracy (see appendix “Comment Checker”).

A one-shot custom prompt feature was also added, letting users query the full repository (e.g., “Summarize this codebase”) via system instructions. Useful for onboarding or repo exploration, it does not support follow-ups due to context limits (see “Custom Prompts” in appendix).

6. Evaluation & Expected Impact

Through usage by the author, the code search and automated documentation features are revealed to improve productivity, especially for large repositories. However, limitations in context occasionally led to subtle inaccuracies since not all information about the repository is contained in code. Iterative prompt tuning detailed in the earlier sections mitigated some issues, but future versions may benefit from conversational context support or fine-tuned models.

This tool benefits developers working with unfamiliar or large codebases by speeding up code exploration, onboarding, and documentation. It would be most useful for teams contributing to multiple repositories where rapid understanding and consistent documentation are critical.

7. Conclusion

This project demonstrates the potential of integrating GitHub APIs with LLMs to enhance code search, documentation, and understanding. By combining traditional API functionality with intelligent LLM-driven features, the SaaS enables users to efficiently search repositories, generate code explanations, and automate routine development tasks such as pull requests and README creation. Through iterative prompt engineering and experimentation, the system evolved to handle real-world challenges such as rate limits, token limits, contextual accuracy, and output formatting. While limitations remain, especially in complex reasoning or subtle code analysis, the outcomes show that even with current constraints, LLMs can provide significant value in developer workflows. Future work may involve supporting searching for code by asking questions to an LLM, improving comment validation reliability, and integrating model fine-tuning for detection accuracy of AI generated code.

8. References

[1] G. Staff, ‘Octoverse: AI leads Python to top language as the number of global developers surges’, The GitHub Blog. Accessed: Apr. 07, 2025. [Online]. Available: https://github.blog/news-insights/octoverse/octoverse-2024/

[2] ‘REST API endpoints for search’, GitHub Docs. Accessed: Apr. 07, 2025. [Online]. Available: https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28#about-search

[3] ‘Searching commits’, GitHub Docs. Accessed: Apr. 07, 2025. [Online]. Available: https://docs.github.com/en/search-github/searching-on-github/searching-commits

[4] ‘Using pagination in the REST API’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/using-the-rest-api/using-pagination-in-the-rest-api?apiVersion=2022-11-28

[5] ‘Octokit’, GitHub. Accessed: Apr. 10, 2025. [Online]. Available: https://github.com/octokit

[6] ‘REST API endpoints for repository contents’, GitHub Docs. Accessed: Apr. 10, 2025. [Online]. Available: https://docs.github.com/en/rest/repos/contents?apiVersion=2022-11-28#get-repository-content

[7] ‘REST API endpoints for repositories’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#get-a-repository

[8] ‘REST API endpoints for branches’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/branches/branches?apiVersion=2022-11-28#get-a-branch

[9] ‘REST API endpoints for Git trees’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/git/trees?apiVersion=2022-11-28#create-a-tree

[10] ‘REST API endpoints for Git commits’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/git/commits?apiVersion=2022-11-28#create-a-commit

[11] ‘REST API endpoints for Git references’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/git/refs?apiVersion=2022-11-28#create-a-reference

[12] ‘REST API endpoints for pull requests’, GitHub Docs. Accessed: Apr. 16, 2025. [Online]. Available: https://docs.github.com/en/rest/pulls/pulls?apiVersion=2022-11-28#create-a-pull-request

[13] ‘About generative models | Gemini API’, Google AI for Developers. Accessed: Apr. 16, 2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/models/generative-models

[14] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr, ‘Quantifying Language Models’ Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting’, Jul. 01, 2024, *arXiv*: arXiv:2310.11324. doi: 10.48550/arXiv.2310.11324.

[15] ‘Prompt Engineering for AI Guide’, Google Cloud. Accessed: Apr. 16, 2025. [Online]. Available: https://cloud.google.com/discover/what-is-prompt-engineering

[16] ‘How Do AI Detectors Work? | GPTZero’, AI Detection Resources | GPTZero. Accessed: Apr. 16, 2025. [Online]. Available: https://gptzero.me/news/how-ai-detectors-work/

[17] ‘Introduction’, Model Context Protocol. Accessed: Apr. 17, 2025. [Online]. Available: https://modelcontextprotocol.io/introduction

[18] ‘Function Calling with the Gemini API’, Google AI for Developers. Accessed: Apr. 17, 2025. [Online]. Available: https://ai.google.dev/gemini-api/docs/function-calling

[19] *googleapis/js-genai*. (Apr. 16, 2025). TypeScript. Google APIs. Accessed: Apr. 17, 2025.

[Online]. Available: https://github.com/googleapis/js-genai
[20] 'Generate content with the Gemini API in Vertex AI | Generative AI on Vertex AI', Google Cloud. Accessed: Apr. 17, 2025. [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/inference

Appendix

Link to the source code and running application

Owen-Choh/SC4052-Cloud-Computing-Assignment-2
https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2

Application is accessible here
https://sc4052-cloud-computing-assignment-2.glitch.me/

Guide on creating github personal access token

https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens

Guide on getting Gemini api key

https://ai.google.dev/gemini-api/docs/api-key

Folder containing all experiments

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration

Link to the larger repository referenced in the report

Owen-Choh/SC4052-Cloud-Computing-Project
https://github.com/Owen-Choh/SC4052-Cloud-Computing-Project

Figures for GitHub API Integration

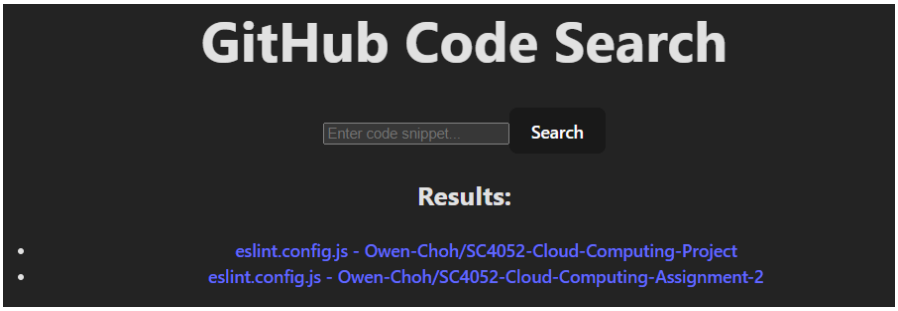


Figure 1: Prototype UI to display search results

```
const response = await githubSearchApi.get(
  `https://api.github.com/search/code?q=${query}+language:javascript+user:Owen-Choh`,
);

setResults(response.data.items);
```

Figure 2: JavaScript code to send GET request to Github API

```
export const githubSearchApi = axios.create({
  baseURL: 'https://api.github.com/search/code',
});

githubSearchApi.interceptors.request.use((config) => {
  config.headers.Authorization = `Bearer ${GITHUB_TOKEN}`;
  return config;
});
```

Figure 3: JavaScript code to create axios object with search code endpoint for reuse

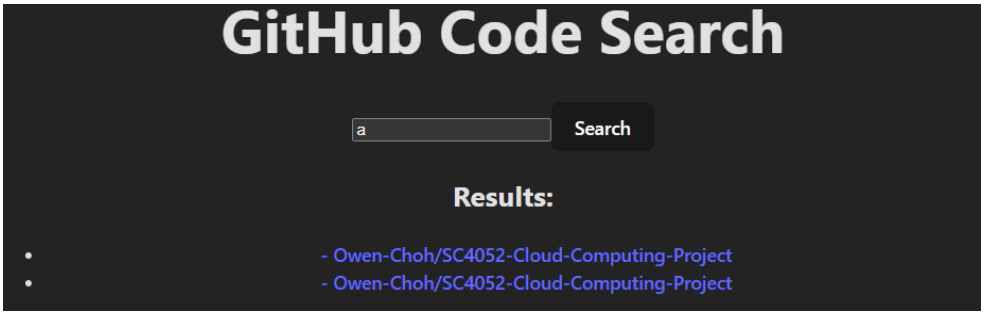


Figure 4: Prototype UI to display search commit results

```
const response = await githubSearchApi.get(
  `https://api.github.com/search/commits?q=${query}+author:Owen-Choh+repo:Owen-Choh/SC4052-Cloud-Computing-Project`,
);
```

Figure 5: JavaScript code to search for commits

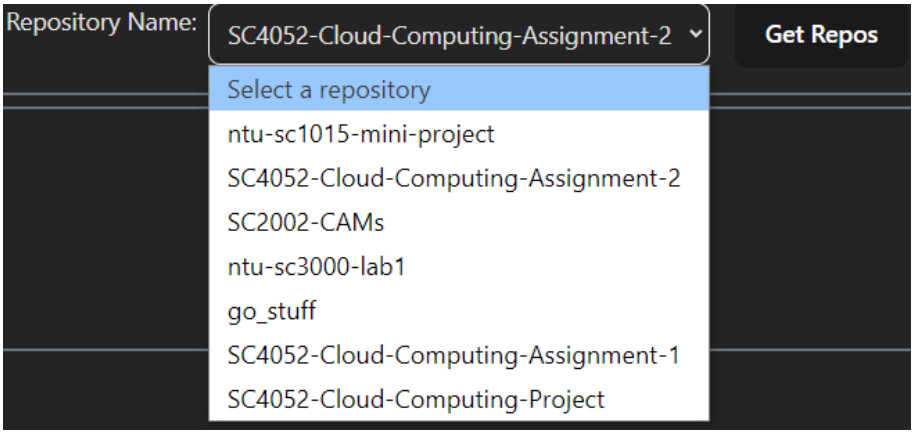


Figure 6: Dropdown UI for users to select repository

```
const response = await githubSearchRepoApi.get(
  githubSearchRepoApi.defaults.baseURL + `?q=user:${username}`
);
```

Figure 7: JavaScript code to obtain a list of repositories

```
export const githubSearchRepoApi = axios.create({
  baseURL: 'https://api.github.com/search/repositories',
  headers: {
    Authorization: `Bearer ${GITHUB_TOKEN}`,
  },
});
```

Figure 8: JavaScript code to create axios object with search repository endpoint for reuse

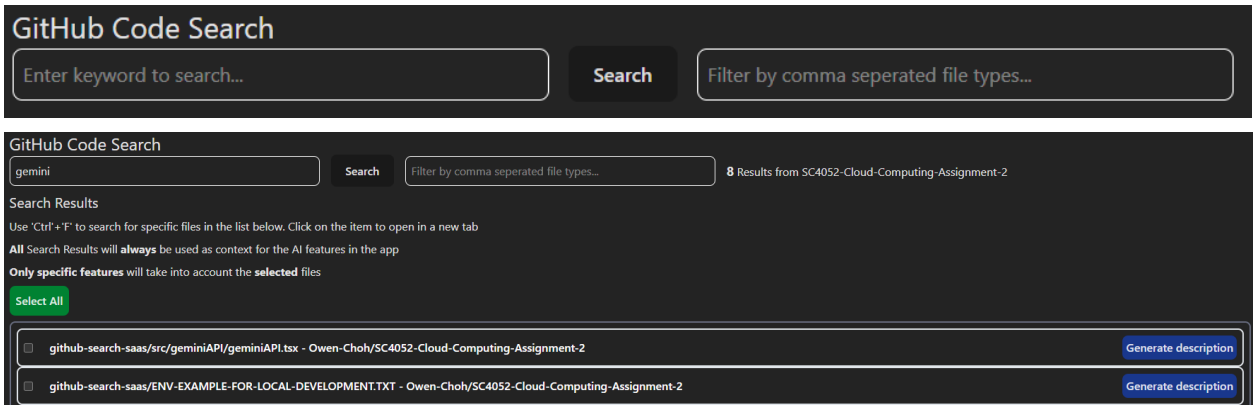


Figure 9: UI with results for users to search for code using keywords and filter by file type

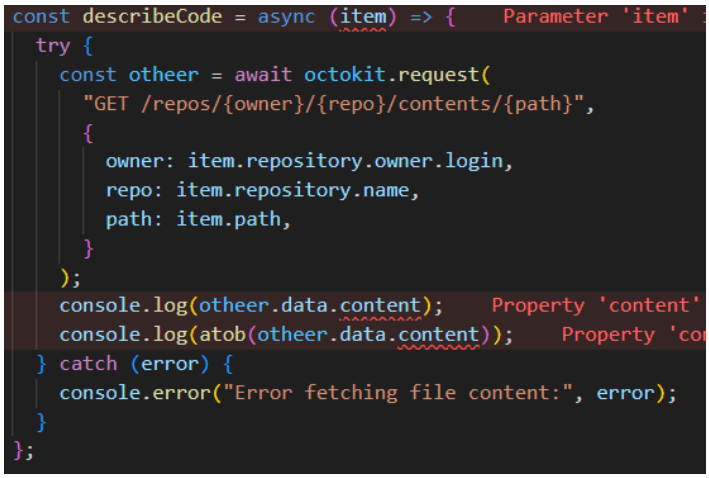


Figure 10: Fetching file contents using Octokit

Figure 11: Fetching file contents using JavaScript

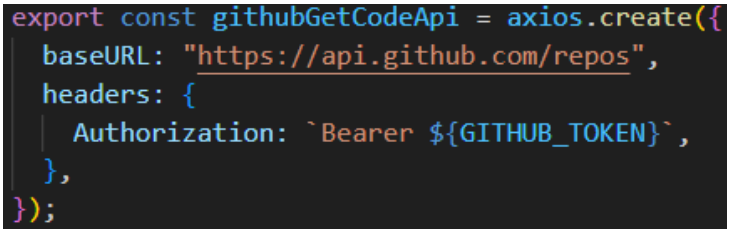


Figure 12: Axios object created to support fetching items from repositories

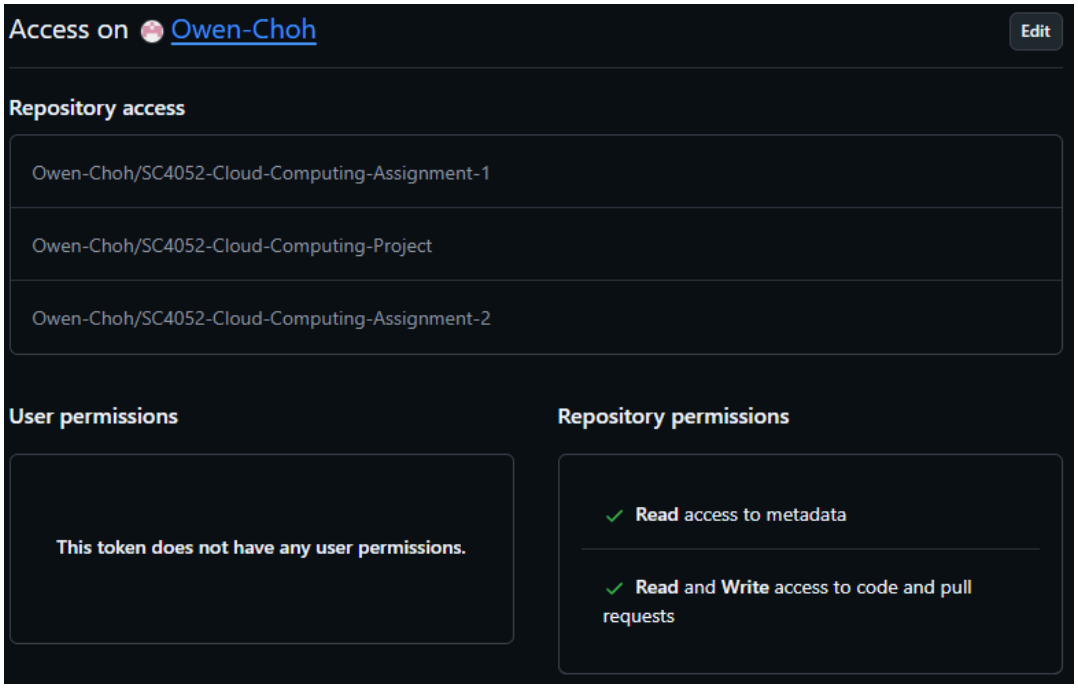


Figure 13: Fine Grained Token permissions for project displayed on Github website

Figures for UI Design

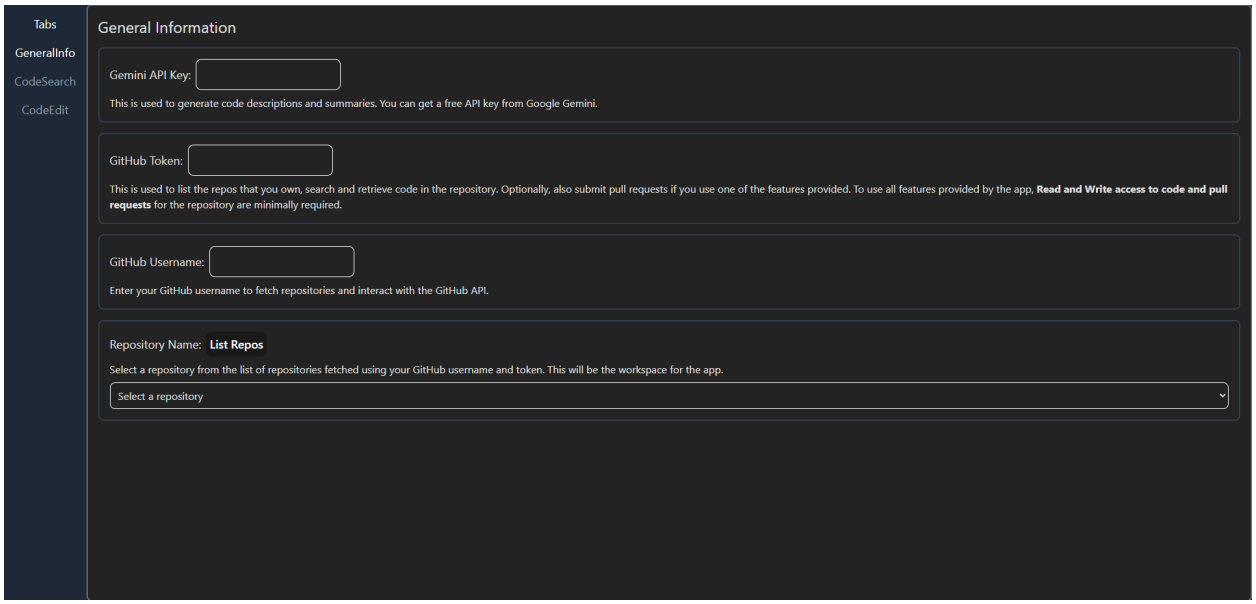


Figure 14: “GeneralInfo” UI for users to input the parameters of the SaaS

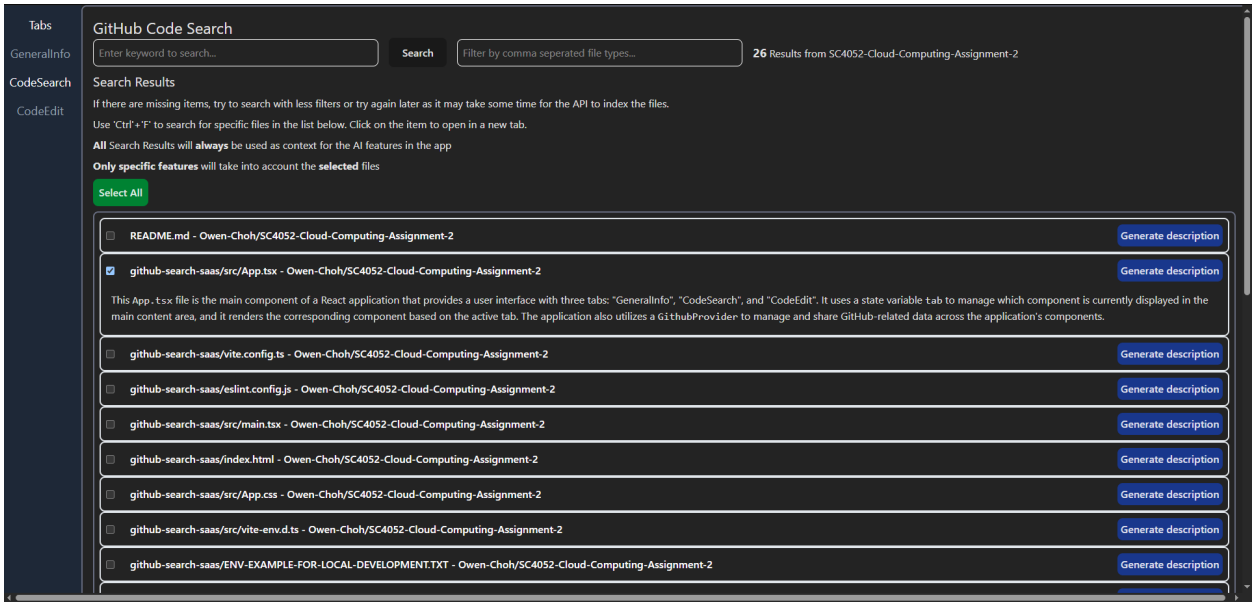
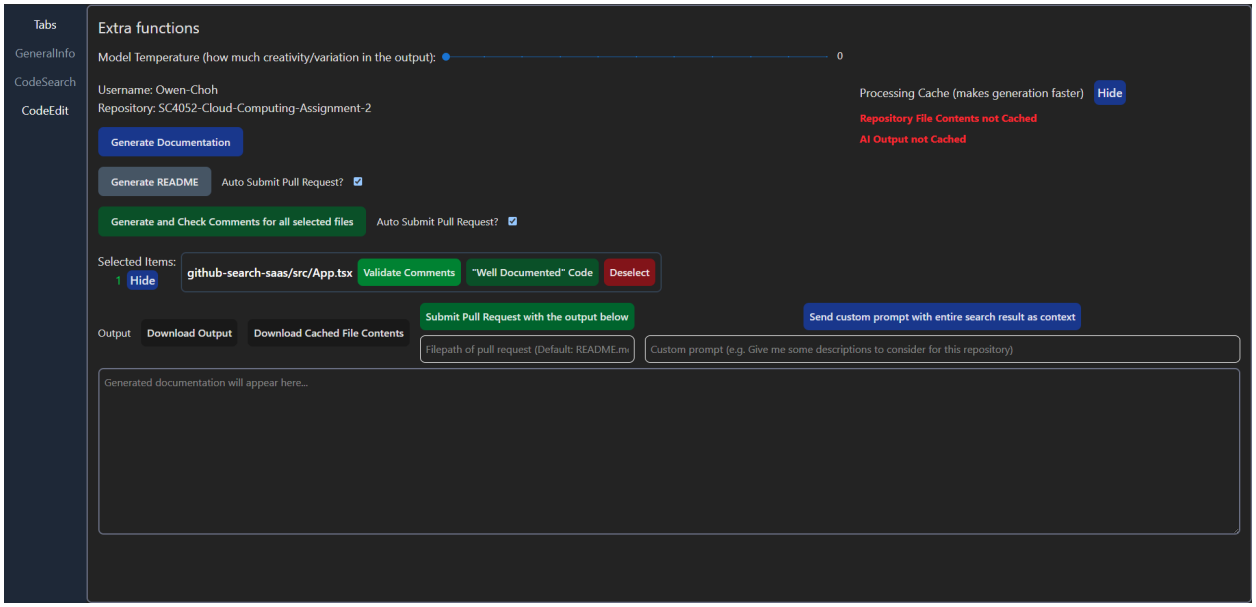


Figure 15: “CodeSearch” UI for users to search for code and generate short descriptions



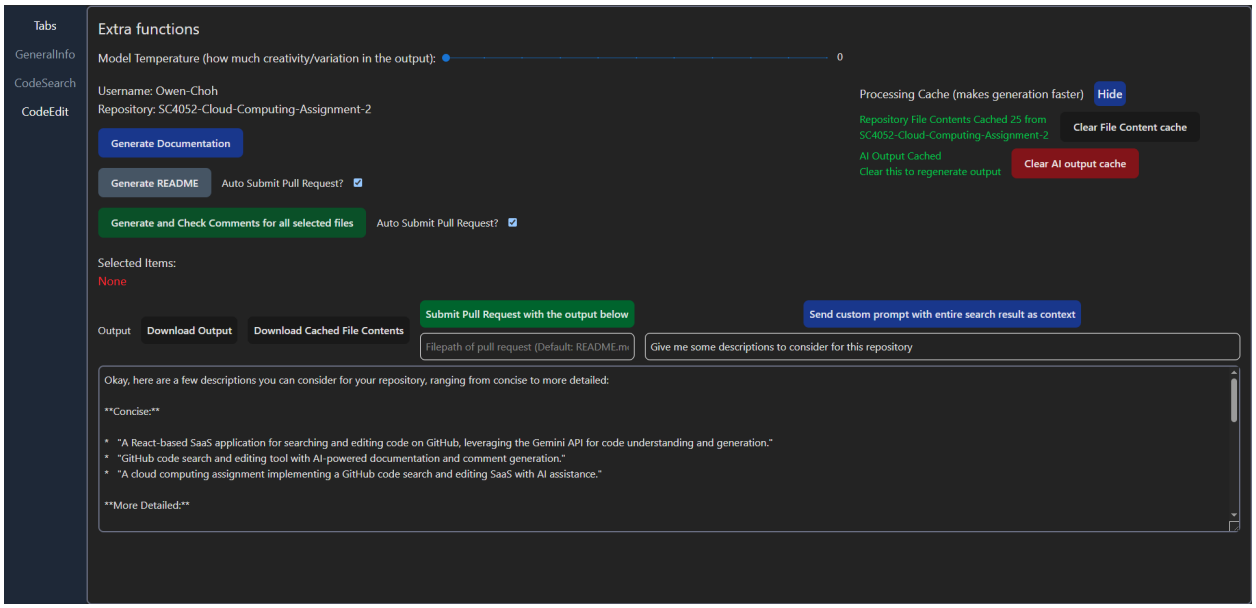


Figure 16: “CodeEdit” UI containing most of the SaaS LLM functionalities, with and without data

Figures for Generate code description

Link to files for Generate code description

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/generate%20code%20description%20experiments

Table 1: Iteration of prompts used to generate code descriptions and observation of changes

No.	Description of Iteration	Prompt	Input	Output	Observation
1	Simple prompt to start off	“What does this code do? ” + codeContent	File content is in “backup of loginpage.tsx”	description 1.md	Quite thorough in explaining the different lines in the file, the summary in the end looks ok for a summary on the UI but the whole explanation is too long.
2	Trying to get a shorter description	“Give a beginner-friendly explanation of what this code does. ” + codeContent		description 2.md	Asking for a beginner-friendly explanation makes it explain the lines more thoroughly. These explanations allow beginners to understand how to use the code but are not relevant for the UI.

3	Trying to get the AI to summarise its explanation	“Summarise what this code is doing. ” + codeContent		description 3.md	Similar to the first prompt, there are too many explanations but it is explaining the code in sections rather than line by line.
4	Trying to get a shorter description	“Summarise what this code is doing in two to three sentences. ” + codeContent		description 4.md	Able to generate a short summary of the code.
5	Add additional context to see if the output is more relevant to the application’s functionality	`User searched for ${query} and wants a description of the code from the file ${item.name} - ${item.repository.full_name} Summarise what this code is doing in two to three sentences. ${codeContent}`	• File content is in “backup of loginpage.tsx” • Query for “login” • Describe LoginPage.tsx - Owen-Choh/SC4052-Cloud-Computing-Project	description 5.md	Seems similar to description 4.md. Since adding more context did not make it worse, this is used for the description functionality.

```
const result = await chatSession.sendMessage("What does this code do? " + code);
```

Figure 17: Prompt iteration 1

```
const result = await chatSession.sendMessage("Give a beginner-friendly explanation of what this code does. " + code);
```

Figure 18: Prompt iteration 2

```
const result = await chatSession.sendMessage("Summarise what this code is doing. " + code);
```

Figure 19: Prompt iteration 3

```
const description = await genCodeDescription(
  "Summarise what this code is doing in two to three sentences. " + atob(otheer.data.content)
);
```

Figure 20: Prompt iteration 4

```
const description = await genCodeDescription(
  `User searched for ${query} and wants a description of the code from the file ${item.name} - ${item.repository.full_name}
  Summarise what this code is doing in two to three sentences. ${atob(otheer.data.content)}`    Property 'content' does not
);
```

Figure 21: Prompt iteration 5 used on LoginPage.tsx in SC4052-Cloud-Computing-Project

```
var desc = query ? query : "all code in the repository";
const description = await generateContent(
  geminiApiKey,
  `User searched for ${desc} and wants a description of the code from the file ${
    item.name
  } - ${item.repository.full_name}
  Summarise what this code is doing in two to three sentences. ${atob(
    getCodeResponse.data.content
  )}`
);
```

Figure 22: Default query given to the prompt when there is no keywords used for searching

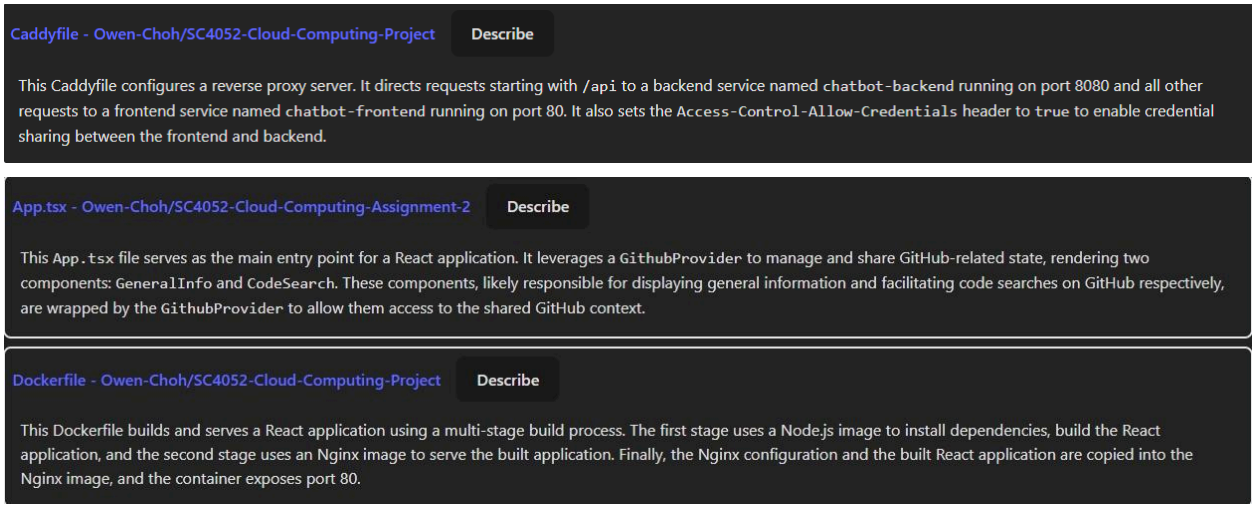


Figure 23: Description before adding a “default query” to the prompt

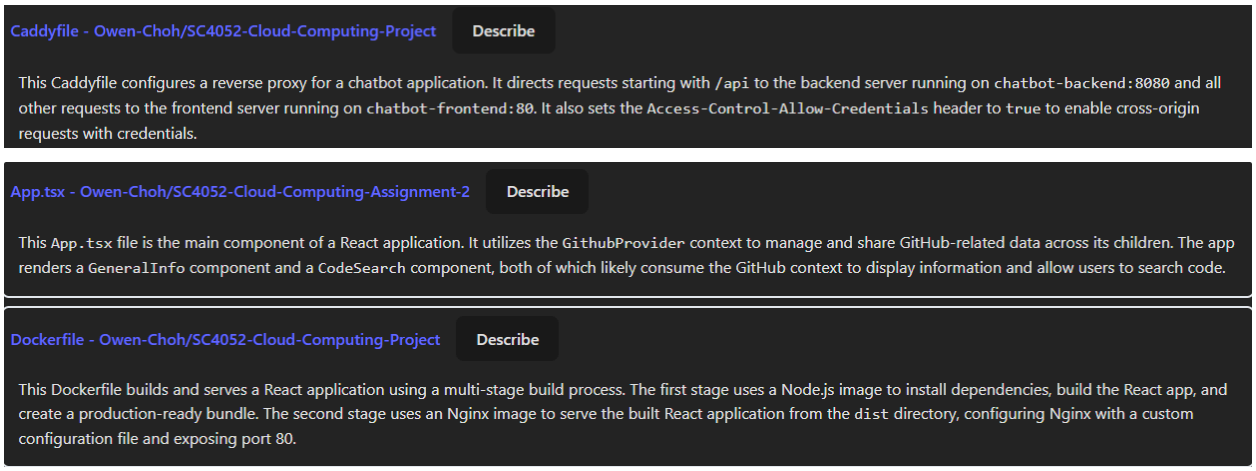


Figure 24: Description after adding a “default query” to the prompt

Figures for AI generated code experiment

Link to files for AI generated code experiment

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/ai%20generated%20code%20experiments

- experiment 1.md: Evaluation of human-written React login page
- experiment 2.md: Evaluation of AI-generated React login page
- experiment 3.md: Evaluation of human-written Go user service
- experiment 4.md: Evaluation of AI-generated Go user service
- Code backups and generation prompts are stored in supplementary .txt files

```
<button
  type="submit"
  className="bg-blue-800 text-white p-2 rounded
  mt-2"
>
  Register
</button>
```

Figure 25: Example of AI generated code in styling the React login page code

```
if err := r.ParseMultipartForm(1000); err != nil {
  log.Println("Error parsing login form:", err)
  utils.WriteError(w, http.StatusBadRequest, err)
  return
}

payload := types.LoginUserPayload{
  Username: r.FormValue("username"),
  Password: r.FormValue("password"),
}
```

Figure 26: AI assistance in implementing MultipartForm parsing in Golang user service

```
const result = await chatSession.sendMessage("Given this code, is it likely AI-generated? Justify your reasoning. " + code);
```

Figure 27: Prompt given to Gemini to evaluate the code

Figures for Generating Code Documentation

Link to files for Generating Code Documentation

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/generate%20code%20documentation%20experiments

Table 2: Iteration of prompts used to generate code documentation and observation of changes

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Initial prompt	Code in a for loop to collate all file contents: repoFileContents += item.path + "\n" + fileContent + "\n\n"; Final code to construct prompt sent to the LLM: finalPrompt = `Generate documentation for the repository ${repository} with the following code:\n\n${repoFileContents}`;	generate documents 1.md	Additional ```\n prepended to the output. Attach the code directly which made the documentation very long.
Attempt to reduce verbosity and avoid code inclusion	finalPrompt = `Generate documentation for the repository ${repository} with the following code. **You do not need to include the code directly in the documentation, you may chose to include the file path if required**.\n\n${repoFileContents}`;	generate documents 2.md	Too little technical information in the documentation which reduces its usefulness
Attempt to format and structure the output	finalPrompt = `Generate documentation for the repository ${repository} with the following code. **For conciseness,** you do not need to include the code directly in the documentation, you may chose to include the file path if required. **Write the documentation in a way that is easy to understand for a beginner. The documentation should be in markdown format. The documentation should be split into two sections: how-to guides and reference guides.**\n\n${repoFileContents}`;	generate documents 3.md	introduced proper structure but wrapped the entire content in a ```markdown code block, which was not ideal
Attempt to remove markdown tags wrapping the output	finalPrompt = `Generate documentation for the repository ${repository} with the following code. For conciseness, you do not need to include the code directly in the documentation, you may chose to include the file path if required. Write the documentation in a way that is easy to understand for a beginner. The documentation should use markdown **styling, do not wrap your entire output in markdown tags.** The documentation should be split into two sections: how-to guides and reference guides.\n\n${repoFileContents}`;	generate documents 4.md	Fixed the wrapping issue but had a less detailed reference guide. Added warning to not commit api key to version control

Include more details as the previous iteration resulted in a less detailed reference guide	finalPrompt = `Generate documentation for the repository ${repository} with the following code. For conciseness, you do not need to include the code directly in the documentation, you may chose to include the file path if required. Write the documentation in a way that is easy to understand for a beginner. The documentation should use markdown styling, do not wrap your entire output in markdown tags. The documentation should be split into two sections: how-to guides and reference guides. **Try to be detailed for the reference guide.**\n\n${repoFileContents}`;	generate documents 5.md	Included details of what each files contains and what useful functions are exported from the file. The notes from earlier are missing
Add relevant notes for users, such as avoiding the inclusion of API keys in version control	finalPrompt = `Generate documentation for the repository ${repository} with the following code. For conciseness, you do not need to include the code directly in the documentation, you may chose to include the file path if required. Write the documentation in a way that is easy to understand for a beginner. The documentation should use markdown styling, do not wrap your entire output in markdown tags. The documentation should be split into two sections: how-to guides and reference guides. Try to be detailed for the reference guide. **Also include notes for anything the reader should look out for**\n\n${repoFileContents}`;	1. generate documents 6.md 2. generate documents 6 - regenerate 1.md 3. generate documents 6 - regenerate 2.md 4. generate documents 6 - regenerate 3.md	Satisfactory output but seemed to get the file structure wrong. Regenerating the output with no changes seemed to correct the error in the file structure.
Regenerating the output resulted in the file structure being fixed. Tried again with model temperature = 0	Same prompt as previous iteration Temperature = 0	1. generate documents with 0 temperature.md 2. generate documents with 0 temperature - 1.md	Regenerating with temperature = 0 produces the satisfactory results and removed variation in the output

Validate prompt effectiveness on a larger repository	Same prompt as previous iteration Temperature = 0 Tested on Owen-Choh/SC4052-Cloud-Computing-Project instead	generate documents with larger repo.md	The output structure is still maintained but seem to ignore the instruction to not include code

Figures for Generating Repository README

Link to files for Generating Code Documentation

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/generate%20readme%20experiments

Table 3: Iteration of prompts used to generate README and observation of changes

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Initial prompt adapted from finalized prompt used to generate code documentation. Started by testing on the larger repository with temperature = 0	finalPrompt = `Generate a **README** for the repository ${repository} with the following code. For conciseness, you do not need to include the code directly in the **README**, you may chose to include the file path if required. Write the **README** in a way that is easy to understand for a beginner. The **README** should use markdown styling, do not wrap your entire output in markdown tags. Also include notes for anything the reader should look out for\n\n${repoFileContents}`;	generate readme 1.md	Included unnecessary wrapping in markdown tags
Shift the markdown style requirements to the front since it might be because the ai lost track of the instruction	finalPrompt = `Generate a README for the repository ${repository} with the following code. **The README should use markdown styling, do not wrap your entire output in markdown tags.** For conciseness, you do not need to include the code directly in the README, you may chose to include the file path if required. Write the README in a way that is easy to understand for a beginner. Also include notes for anything the reader should look out for\n\n${repoFileContents}`;	generate readme 2.md	Many duplicated lines at the bottom. This may be due to the low temperature causing it to keep selecting the same output tokens by selecting the highest probability.

			Includes too much information for a readme
Attempt to format and structure the output	finalPrompt = `Generate a README for the repository ${repository} with the following code. The README should use markdown styling, do not wrap your entire output in markdown tags. For conciseness, you do not need to include the code directly in the README, you may chose to include the file path if required. Write the README in a way that is easy to understand for a beginner. **Include a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.** Also include notes for anything the reader should look out for\n\n${repoFileContents}`;	generate readme 3.md	Able to follow instructions but there is a lot of duplicated lines at the bottom same as previous. Still including too much information for a readme
Attempt to remove the duplicated lines by using system instructions	**finalPrompt** = `These are the contents of the files in the repository\n\n${repoFileContents}`; **systemInstruction** = `Generate a README for the code repository ${repository}. The README should use markdown styling, do not wrap your entire output in markdown tags. For conciseness, you do not need to include the code directly in the README, you may chose to include the file path if required. Write the README in a way that is easy to understand for a beginner. Include a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it. Also include notes for anything the reader should look out for`;	generate readme 4.md	Able to follow instructions but there is a lot of duplicated lines at the bottom same as previous
Attempt to regenerate with slight modifications	systemInstruction = `Generate a README for the code repository ${repository}. The README should use markdown styling, do not wrap your entire output in markdown tags. For conciseness, you do not need to include the code directly in the README, you may chose to include the file path if required. Write the README in a way that is easy to understand for a beginner. Include a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use	generate readme 5.md	Duplicated lines at the bottom are removed. Started wrapping in markdown tags again.

	it. Also include notes for **key things** that the reader should look out for`;		
Attempt to remove the markdown tags, format the input with new line characters to see if it works better	systemInstruction = `Generate a README for the code repository ${repository}.**\n**The README should use markdown styling, do not wrap your entire output in markdown tags.**\n**You do not need to include the code directly in the README, you may chose to include the file path if required.**\n**Write the README in a way that is easy to understand for a beginner.**\n**Include a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.**\n**Also include notes for key things that the reader should look out for when using the repository.`;	generate readme 6.md	Still wraps in markdown tags. Added a “description” section
Attempt to remove the markdown with capital letters	systemInstruction = `Generate a README for the code repository ${repository}.\nThe README should use markdown styling, **DO NOT** wrap your entire output in markdown tags.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include notes for key things that the reader should look out for when using the repository.`;	generate readme 7.md	Still wrapped in markdown tags.
Attempt a different prompt to specify in markdown	systemInstruction = `Generate a README for the code repository ${repository}.\n**Format the README using standard Markdown syntax for text styling. Avoid using code blocks unless displaying code.**\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include notes for key things that the reader should look out for when using the repository.`;	generate readme 8.md	markdown tags are removed. Added extra sections such as “Secrets” and “Contributing”. “Contributing” section is not desired

specify “brief” notes instead	systemInstruction = `Generate a README for the code repository ${repository}.\nFormat the README using standard Markdown syntax for text styling. Avoid using code blocks unless displaying code.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include **brief** notes for key things that the reader should look out for when using the repository.`;	generate readme 9.md	Gave an even longer readme. Added an explanation of the output at the end, the LLM may have believed that someone is interacting with it
Gave an example of notes to steer the output and specify to return readme contents only	systemInstruction = `Generate a README for the code repository ${repository}**, only return the contents of the README.**\nFormat the README using standard Markdown syntax for text styling. Avoid using code blocks unless displaying code.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository **such as not commiting their env file.**`;	generate readme 10.md	Outside of a few details specific to this repository such as needing to change the configured domain in the Caddyfile, the readme is satisfactory
Validate prompt on different repository	Tested on Owen-Choh/SC4052-Cloud-Computing-Assignment-2	generate readme 10 - regenerate on different repo.md	Produced satisfactory readme

Figures for LLM Function Calling

Link to files for LLM Function Calling

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/function%20calling%20experiments

Link to files for LLM Function Calling with JSON

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/generate%20with%20function%20json%20experiments

Table 4: Iteration of prompts used to implement function calling and observation of changes

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Initial prompt adapted from prompt used to generateREADME. Started by testing on the project's repository with temperature = 0	**systemInstruction** = `Submit a pull request for a suggested README file for this code repository ${repository}.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file.`  **finalPrompt** = `These are the contents of the files in the repository\n\n${repoFileContents}` Included function declaration (see figure 29)	The writer did not save a copy of the output.	Rarely produced a successful response. README was too generic.
Simplify the prompt to narrow down the error	systemInstruction = "Submit a pull request for a suggested README file for this code repository" Appended all file paths and contents individually in the chat session history	generate function 1.md generate function 1 pull request.md	The pull request contains incomplete readme with placeholders.
Remove "suggested" as it may have caused the placeholders	systemInstruction = "Submit a pull request for **a README** file for this code repository" Appended all file paths and contents individually in the chat session history	generate function 2.md generate function 2 pull request.md	The pull request still contains incomplete readme with placeholders.

Attempt to get a complete README. Used temperature = 1 as 0 kept returning error.	systemInstruction = "Submit a pull request for a **complete** README file for this code repository" Appended all file paths and contents individually in the chat session history	generate function 3.md generate function 3 pull request.md	The pull request still contains incomplete readme with placeholders.
Attempt to get it to reference the files provided	systemInstruction = "Submit a pull request for a complete README file **based on the files given**" Appended all file paths and contents individually in the chat session history	generate function 4.md generate function 4 pull request.md	The pull request contains a generic readme
Reattempting system instructions with temperature = 1	**systemInstruction** = `Submit a pull request for a suggested README file for this code repository ${repository}.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file.`;  **finalPrompt** = `These are the contents of the files in the repository\n\n${repoFileContents}`	generate function 5.md generate function 5 pull request.md	Rarely produced a successful response. The pull request contains a generic readme
Also attempted prompts similar to this iteration and altering the function declaration without much success.	systemInstruction = “Submit a pull request for a suggested README file for my code repository Owen-Choh/SC4052-Cloud-Computing-Assignment-2. Just make the function call directly by having the function call be in your candidate response reply.\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief	generate function not working.md generate function not working 2.md generate function not working 2.md generate function not	Responses appeared to be python code that runs the submit pull request function. Made use of code suggested by Gemini flash on aistudio (see “parse output directly solution from google aistudio.md”), however it was not consistent as the responses frequently failed leading to the

	notes that the reader should look out for when using the repository such as not commiting their env file." **finalPrompt** = `These are the contents of the files in the repository\n\n${repoFileContents}`	working 4 copy.md	app being rate limited from repeated generation attempts. Moreover, the output format is not consistent.

Table 5: Iteration of prompts to generate JSON for function calls and observation of changes

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Initial prompt tested in aistudio to verify feasibility of approach	systemInstruction = "Submit a pull request for a suggested README file for my code repository Owen-Choh/SC4052-Cloud-Computing-Assignment-2. **Give me the function call in a JSON format with the name of the function and the arguments to make the function call.**\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file." **finalPrompt** = `These are the contents of the files in the repository\n\n${repoFileContents}`	generate function json.md	Successfully produced a JSON response with correct formatting and attributes
Implement in the app	Same prompt as previous iteration Temperature = 1	generate function json 1.md	Did work as intended as it included explanations
Attempt to get it to only return json Set temperature	systemInstruction = `Submit a pull request for a suggested README file for my code repository ${repository}. Give me the function call in a JSON format with the name of the function and the arguments to make the function call. **Your response will be parsed directly as json without being seen by the**	generate function json 2.md	Explanations are removed but it wraps the response in json markdown tags

= 0 for testing for consistency	**user.**\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file.`;		
Attempt to fix the json tag issue.	systemInstruction = `Submit a pull request for a suggested README file for my code repository ${repository}. Give me the function call in a JSON format with the name of the function and the arguments to make the function call. Your response will be parsed directly as json without being seen by the user. Do not wrap your output in \`\`\`json tags\nYou do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file.`;	generate function json 3.md	Still wraps the response in json markdown tags
Attempt to give it a persona to remove the json tag	systemInstruction = `You are an API agent. Your response will be consumed directly by code and parsed as a JSON object. Do not format your JSON output in markdown fence blocks. Do not include any explanations. Do not use code fences like \`\`\`json. Just return a raw JSON object.\nSubmit a pull request for a suggested README file for my code repository ${repository}. You do not need to include the code directly in the README, you may chose to include the file path if required.\nWrite the README in a way that is easy to understand for a beginner.\nInclude a short description of what the repository contains, an overview of the code, architecture (if applicable) and how to set up and use it.\nAlso include brief notes that the reader should look out for when using the repository such as not commiting their env file.`;	generate function json 4.md	Still wraps the response in json markdown tags. However, the README looks satisfactory.

Regenerate with code added (see figure 30) to extract the json object	Same prompt. Attempted with both temperature = 0 and 1	generate function json 5 - #14.md generate function json 6 - #15.md	successfully submitted pull requests (#14 and #15)
Validation with a larger repository	Same prompt but on Owen-Choh/SC4052-Cloud-Computing-Project Works well with temperature of 0 and 1	generate function json 7 - #3 0 temperature .md generate function json 7 - #4 1 temperature .md	successfully submitted pull requests (#3 and #4)

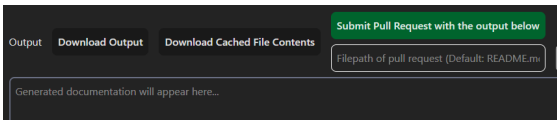


Figure 28: Green button to manually submit the output as a pull request on the CodeEdit UI

```
export const submitPullRequestFunctionDeclaration = {
  name: "submit_pull_request",
  description: "Submit a pull request to the repository.",
  parameters: {
    type: Type.OBJECT,
    properties: {
      filePath: {
        type: Type.STRING,
        description: 'path of the file with file name, e.g. "README.md"',
      },
      commitMessage: {
        type: Type.STRING,
        description: "commit message of the change",
      },
      branchName: {
        type: Type.STRING,
        description: "branch name created for the pull request",
      },
      pullRequestTitle: {
        type: Type.STRING,
        description: "The title of the pull request.",
      },
      pullRequestBody: {
        type: Type.STRING,
        description: "The body of the pull request.",
      },
      fileContent: {
        type: Type.STRING,
        description: "The content of the file.",
      },
    },
    required: [
      "filePath",
      "commitMessage",
      "branchName",
      "pullRequestTitle",
      "pullRequestBody",
      "fileContent",
    ],
  },
};
```

Figure 29: The final version of the function declaration passed to the LLM to make pull requests (also saved in function declaration.json).

```
// try again after cleaning since the ai keeps wrapping in these tags
const cleanJson = genWithToolsResponse.text
  .replace(/```json|```$/g, "")
  .trim();
const parsedResponse: PullRequestArgs = JSON.parse(cleanJson);
parsedResponse.fileContent = stripCodeFences(
  parsedResponse.fileContent);
console.log("parsed response after cleaning: ", parsedResponse);
({ result: pullRequestResult, errmsg } = await submitPullRequest(
  username,
  repository,
  token,
  [
    {
      filePath: parsedResponse.filePath,
      fileContent: parsedResponse.fileContent,
    },
  ],
  parsedResponse.commitMessage,
  parsedResponse.branchName,
  parsedResponse.pullRequestTitle,
  parsedResponse.pullRequestBody
));
```

Figure 30: Code added to extract the JSON object from the LLM response

Figures for Comment Checker

Link to files for Comment Checker

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/check%20comments%20experiments

Table 6: Iteration of prompts to check comments and observation of changes (backup of file contents are provided and are relevant for subsequent outputs unless specified)

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Initially tested on smaller repository Owen-Choh/SC4052-Cloud-Computing-Assignment-2 (21 files), temperature = 0	systemInstruction = `Help me check the comments written for the code repository ${repository} and make sure they are accurate` finalPrompt = `These are the contents of the files in the repository\n\n${repoFileContents}`	comments 1.md Backup of file contents are in Repository_File_Contents 1.txt	Gave feedback for all the files with application code.
Attempt to get the final updated file as output to function within the SaaS	systemInstruction = `Help me check the comments written for the code repository ${repository} and make sure they are accurate. **Give me the full updated file if the comment needs changes.**`;	comments 2.md	Managed to get the full updated file but missing the other comments that is mentioned in comments 1.md
Attempt to get all the files missing from the previous prompt	systemInstruction = `Help me check the comments written for the code repository ${repository} and make sure they are accurate. **Go through all files** and give me the full updated file if any comments in the file needs changes.`;	comments 3.md	Still managed to update the file but now justifies that other files do not have comments that need to be updated.

Increase temperature to try to get more suggestions for more files	Same prompt as previous iteration	comments 3 - temperature 1.md comments 3 - temperature 1 (api response).txt	Stopped giving the output as the model output tokens maxed (see candidates.finishReason in the api response).
Limit to one file at a time to manage output tokens	systemInstruction = `Help me check the comments written for the **code** ${selectedFilePath} and make sure they are accurate. **Give** me the full updated file if comments in the file needs changes.`; Attempted with: github-search-saas/src/components/CodeEdit.tsx	comments 4.md	Added some comments and also mentioned that "The comments in the file are accurate and descriptive. No changes are needed."
Attempt to get the LLM to return only no change is needed	systemInstruction = `Help me check the comments written for the code ${selectedFilePath} and make sure they are accurate. Give me the full updated file only if comments in the file needs changes. **Otherwise just let me know that the comments are accurate.**`; Attempted with: github-search-saas/src/App.tsx	comments 5.md	Able to only output The comments in `github-search-saas/src/App.tsx` are accurate. Without giving me the original file with no changes

Table 7: Iteration of prompts to generate well documented code and observation of changes
(backup of file contents are provided and are relevant for subsequent outputs unless specified)

Description of changes	The final prompt constructed in JavaScript before sending to the LLM	Output	Observation
Continued from previous prompt iteration in table 6	systemInstruction = `Help me make sure that the code ${selectedFilePath} is **well documented**. Give me the full updated file only if comments in the file needs changes.`; Attempted one file at a time with github-search-saas/src/App.tsx and github-search-saas/src/components/CodeEdit.tsx	comments 6 - App.md comments 6 - CodeEdit.md Backup of file contents are in Repository_File_Contents 9.txt	Added comments to both files to document the code

Attempt to get the LLM to return only none if change is needed so that my code can easily detect it	systemInstruction = `Help me make sure that the code ${file.path} is well documented. Give me the full updated file only if comments in the file need changes. **Return "none" if no changes are needed. Your output will be parsed by code and will not be seen by users.**`;	comments 7 - #20.md see pull request 20	Only file content is returned in the AI response. It is parsed by code to submit a pull request and rendered as JSON in the output file. But AI response contain markdown code fences
Attempt to stop LLM from wrapping response in markdown tags	systemInstruction = `Help me make sure that the code ${file.path} is well documented. Give me the full updated file only if comments in the file need changes. Return "none" if no changes are needed. Your output will be parsed by code and will not be seen by users. **Do not wrap your output in markdown tags.**`;	comments 8 - #23.md see pull request 23	LLM still only returns the file content but does not exclude the markdown tags. It is parsed by code to submit a pull request and rendered as JSON in the output file. The application still made pull requests with the file content after adding code to remove the markdown code fence.
Get the LLM to include some explanations for the change by adding a new attribute in the JSON	systemInstruction = `Help me make sure that the code ${file.path} is well documented. Give me the full updated file only if comments in the file need changes. Provide brief explanation for your changes to be appended to the body of a pull request. **Format your response in JSON with two attributes 'fileContent' and 'explain'. Set fileContent="none" if no changes are needed.** Your output will be parsed by code and will not be seen by users. Do not wrap your output in markdown tags.`;	comments 9.md Backup of file contents are in Repository_File_Contents 9.txt	Able to follow instructions to format the response. But does not follow instruction when asked to just put “none”

Another attempt after placing fileContent in quotes	systemInstruction = `Help me make sure that the code ${file.path} is well documented. Give me the full updated file only if comments in the file need changes. Provide brief explanation for your changes to be appended to the body of a pull request. Format your response in JSON with two attributes 'fileContent' and 'explain'. Set **"fileContent"**="none" if no changes are needed. Your output will be parsed by code and will not be seen by users. Do not wrap your output in markdown tags.`;	comments 10.md	does not follow instructions to just put “none”
Attempt to reorder the instructions	systemInstruction = `Help me make sure that the code ${file.path} is well documented. **Format your response in JSON with two attributes 'fileContent' and 'explain'. Set 'fileContent'='none' if no changes are needed.** Give me the full updated file only if comments in the file need changes. **'explain' will be appended to the body of a pull request to explain the changes.** Your output will be parsed by code and will not be seen by users. Do not wrap your output in markdown tags.`; Attempted with github-search-saas/src/components/CodeSearch.tsx and github-search-saas/src/components/GeneralInfo.tsx	comments 11.md comments 12.md	Added comments (comments 11.md) But does not only put “none” if no changes (comments 12.md)
Amend prompt to make it start with no changes needed since it is quite consistently starting with a similar phrase	systemInstruction = `Help me make sure that the code ${file.path} is well documented. Format your response in JSON with two attributes 'fileContent' and 'explain'. Set 'fileContent'='none' **AND 'explain' must start with 'No changes needed.'** if no changes are needed. Give me the full updated file only if comments in the file need changes. 'explain' will be appended to the body of a pull request to explain the changes. Your output will be parsed by code	comments 13 - response.md	Able to follow instructions

	and will not be seen by users. Do not wrap your output in markdown tags.`;		
Amended prompt as the LLM frequently puts no changes needed when I feel like there can be some comments added.	systemInstruction = `Help me make sure that the code ${file.path} is well documented **by adding comments and checking the accuracy of existing comments.** Format your response in JSON with two attributes 'fileContent' and 'explain'. Set 'fileContent'='none' AND 'explain' must start with 'No changes needed.' if no changes are needed. Give me the full updated file only if comments in the file need changes. 'explain' will be appended to the body of a pull request to explain the changes. Your output will be parsed by the JSON.parse() JavaScript code and will not be seen by users.`;	comments 14.md	Still responses that no change is needed

Minor changes by reordering instruction and include inaccurate comments in the code	systemInstruction = `Help me make sure that the code ${file.path} is well documented by adding comments and checking the accuracy of existing comments. Format your response in JSON with two attributes 'fileContent' and 'explain'. **If no changes are needed,** Set 'fileContent'='none' AND 'explain' must start with 'No changes needed.'. Give me the full updated file only if comments in the file need changes. 'explain' will be appended to the body of a pull request to explain the changes. Your output will be parsed by the JSON.parse() JavaScript function and will not be seen by users.`;
	
	The inaccurate comment:
	// Array to store outputs for each file
	// only contains filePath and fileContent
	const outputs: {
	filePath: string;
	fileContent: string;
	explain: string;
	}[] = [];
		comments 15 - partial response.md
		comments 15 - api response.json
		
		Backup of file contents are in Repository_File_Contents 15.txt
			Api call stopped due to max_tokens.
			
			But still managed to see that the planted comment got updated to
			
			const outputs: {
			filePath: string;
			fileContent: string;
			explain: string;
			}[] = []; // Array to store outputs for each file

Attempt to get the LLM to add more comments and added inaccurate comment	systemInstruction = `Help me **add comments to the code ${file.path} to make it well documented and check the accuracy of existing comments**. Format your response in JSON with two attributes 'fileContent' and 'explain'. If no changes are needed, Set 'fileContent'='none' AND 'explain' must start with 'No changes needed.'. Give me the full updated file only if comments in the file need changes. 'explain' will be appended to the body of a pull request to explain the changes. Your output will be parsed by the JSON.parse() JavaScript function and will not be seen by users.`; (this function uses axios to make a get request instead of octokit) // Function to get the repositories for the given username using octokit const fetchRepos = async () => {...	comments 16.md Backup of file contents are in Repository_File_Contents 16.txt	Did not add comments Did not detect the wrong comment

Attempt with simpler prompt and using tool declaration (figure 31) to help with formatting the response and tested with inaccurate comment	systemInstruction = `You are an API agent. Your response will be consumed directly by code and parsed as a JSON object. Just return a raw JSON object. Help me make sure that the code ${file.path} is well documented.`
	
	The inaccurate comment:
	// Function to get the repositories for the every repository that the token has access to
	const fetchRepos = async () => {
		comments 17.md
		
		comments 17 - #26 submitted pull request.md
		
		Backup of file contents are in Repository_File_Contents 17.txt
			Still unable to detect the slightly inaccurate comment.
			Able to format the output nicely to be parsed by my code.
			Also able to add extra comments to explain the code (see figure 32).

Validate with larger repo Owen-Choh/SC4052-Cloud-Computing-Project	Same prompt as above	Response: "Okay, I'm ready. Please provide the content of the file `chatbot-app/src/pages/LoginPage.tsx`.\n"	Does not seem to work when there are 70 files worth of content in the prompt

		Backup of file contents are in Repository_File_Contents 18.txt	
pass the file directly in the system prompt since LLM seem to lose track of it	systemInstruction = `You are an API agent. Your response will be consumed directly by code and parsed as a JSON object. Just return a raw JSON object. Help me make sure that the code ${file.path} is well documented.\n\n${repoFileContentMap.get(file.path)}`;	comments 18 - #5.md Backup of file contents are in Repository_File_Contents 18.txt	Submitted pull request 5 to github but there are code fences in the file content
Added code to clean the file contents before submitting pull request	Same prompt as above	comments 19 - #6.md	Submitted pull request with the changes. However during usage of the app, the LLM sometimes gives the "diff" or edits to apply instead of the full file which is not supported by my app
Amend function declaration (figure 33) to obtain full file content	Same prompt as above	comments 20 - #8.md	Able to get comments and documentation for both files, also gave the full file content instead of the "diff"
Amended prompt to improve chance of detecting inaccurate comments and returning responses in the correct format. Tested on Owen-Choh/	systemInstruction = `You are an API agent. Your response will be consumed directly by code and parsed as a JSON object with attributes fileContent and explain. **Help me amend ${file.path} comments to be accurate. Also help me make sure that the code ${file.path} is well documented. Give me the entire updated file in the json output.**\n\n${repoFileContentMap.get(file.path)}`; The inaccurate comment (function returns an error message instead of throwing an error):	comments 21 - #27.md Backup of file contents are in Repository_File_Contents 21.txt	Able to detect inaccuracies in code comments and add documentation

SC4052-Cloud-Computing-Assignment-2	// submits pull request, will throw an error if api call fails export const submitPullRequest = async (		

```
export const parseFileObjectFunctionDeclaration = {
  name: "parse_file_object",
  description: "Parse the file JSON object to be used in the pull request.",
  parameters: {
    type: Type.OBJECT,
    properties: {
      fileContent: {
        type: Type.STRING,
        description: "The content of the file.",
      },
      explain: {
        type: Type.STRING,
        description: "Explaination of the changes.",
      },
    },
    required: ["fileContent", "explain"],
  },
};
```

Figure 31: Code added to extract the JSON object from the LLM response

Figure 32: Comments added to document code in pull request 26

```
export const parseFileObjectFunctionDeclaration = {
  name: "parse_file_object",
  description: "Parse the file JSON object to be used in the pull request.",
  parameters: {
    type: Type.OBJECT,
    properties: {
      fileContent: {
        type: Type.STRING,
        description: "The full content of the file after the changes.",
      },
      explain: {
        type: Type.STRING,
        description: "Explanation of the changes.",
      },
    },
    required: ["fileContent", "explain"],
  },
};
```

Figure 33: Amended function declaration to obtain full content of files for pull request

Figures for Custom Prompts

Link to files for Custom Prompts

https://github.com/Owen-Choh/SC4052-Cloud-Computing-Assignment-2/tree/main/documents/experiments%20and%20exploration/custom%20prompt%20examples

- custom prompt 1.md: Output of the AI with prompt given in figure 34
- Repository_File_Contents 1.txt: Backup of all files in the repository

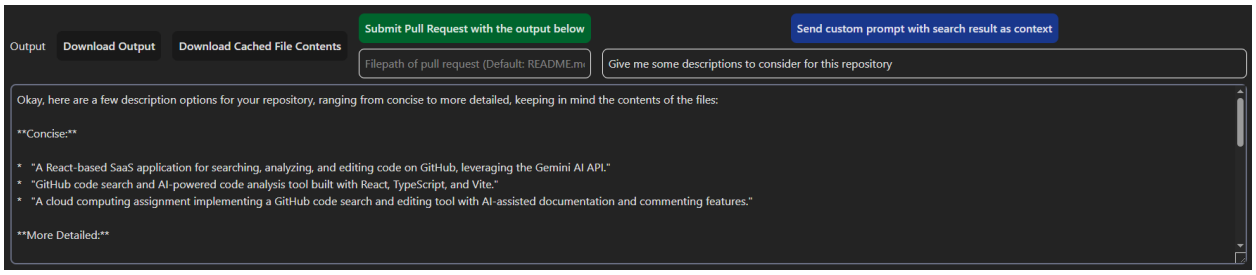


Figure 34: Input box implemented to allow users to send custom prompts to the LLM