How Large Language Models can be used to automatically solve GitHub-Issues

Jimmy Neitzert[1,2] Alpay-Kaan Erbay[1,2] Alperen Dagli[1,2] Albert Makdisi[1,2]
Taha Ilgar[1,2]

[1] University of applied Sciences Ruhr West, Lützowstr. 5, Bottrop, 46236, Germany.
[2] These authors contributed equally to this work.

Abstract. GitHub, as a central hub for software developers worldwide, is confronted with an ever-growing number of issues, the efficient handling of which is essential. Despite the importance that issues can have in the development process, many of them are addressed slowly or not at all, which could lead to delays and possible quality degradation. The introduction of Large Language Models (LLMs), could provide an approach to address these challenges by autonomously generating solutions for handling and suggesting possible solutions to issues. This thesis evaluates the potential of the GPT-4 model for automated processing of GitHub Issues. Technical aspects as well as practicality in a real-world application scenario are considered. The results show how LLMs can potentially transform the software development process and increase efficiency in issue handling. The evaluation is based on four automated issues, testing whether the proposed solutions are accepted by the contributors.

Keywords: GPT, Github, Issues, LLM, Issue Solving

1 Introduction

1.1 Importance of issues and problem definition

GitHub, as a central repository for countless software projects, has established itself as a central hub for developers from all over the world in recent years [1]. Within this platform, issues are an essential mechanism not only to report problems or bugs, but also to discuss and coordinate the further development of a project [2].
Efficient handling of these issues is critical to project progress. Unaddressed or slowly addressed issues can inhibit the flow of development and lead to delays in release cycles [3]. In addition, the quality and speed of issue handling directly influenced community and stakeholder satisfaction [4].
As of September 2023, there are over 100,000,000 open issues on GitHub [5]. Interestingly, this number also roughly corresponds to the number of users on GitHub [6]. The sheer volume of open requests can overwhelm the platform and its developer community. The question is how to efficiently manage and prioritize this massive amount of feedback without compromising the quality and speed of

problem resolution.
Thus, the challenge lies not only in the quantity, but also in the complexity of the issues to be resolved. It is not enough to simply assign more developers to issue handling. Instead, innovative approaches and tools are needed to optimize the process while ensuring the quality of the solutions.

1.2 Importance of issues and problem definition

With the introduction of Large Language Models such as the GPT series of OpenAI, a new approach to automate various processes in IT has emerged. These models not only provide the ability to generate code and text, but could also be used to automate the processing of issues in GitHub repositories. A successful application model would speed up the development process by reducing or eliminating repetitive and time-consuming tasks. However, effective implementation requires LLMs to fully understand the context and logic behind each issue. This work will therefore investigate how well GPT-4 is able to process GitHub Issues in an automated manner and propose solutions.

1.3 Paper Structure

The following sections of this paper follow this scheme: section 2 discusses existing tools for generating code and solving Github Issues and previous research on using Large Language Models in generating code. Section 3 focuses on the architecture of the designed program. The architecture of the system follows a careful approach, starting with the selection of relevant Github repositories, through data organization and vector analysis, to interaction with language models and final code integration, and ultimately pull requests. This section highlights the conceptual and technical decisions that underpin this process. Section 4 outlines the evaluation framework, including strategic planning and implementation. Finally, Section 5 briefly summarizes the results and outlines future research directions.

2 State of the art

In the ongoing evolution of software development using large language models, continuous innovation and automation are essential components that have the potential to revolutionize the entire development and refactoring process. With the goal of exploring the feasibility of a system capable of effectively resolving arbitrary issues of repositories of any size, the current state of the technology is highlighted.

2.1 Embeddings and vector databases

Embeddings generate quantitative representations for text entities and summarize their semantic essence in a numerical format. This is particularly relevant

for identifying and measuring similarities between texts or documents by comparing their embeddings in vector space [7].
The generated embeddings are stored in vector databases. When a user query is received, the input text is converted into an embedding and sent to the vector database to find the closest textual and semantic similarity. Vector database management systems such as "Deep Lake" have become indispensable in modern data management. Their relevance is increasing with the growing need to describe large-scale data such as text, images, and program code in a computer-aided manner in areas such as similarity search and chatbots. These data are captured as numerical vectors that are both cost-effective to store and compare [8].
There are various methods for passing the contents of documents and processing them using LLM. A common method is the "stuff" method. It is particularly effective when the text segments returned by the vector store fit well within the context size of the LLM. This method allows efficient integration of similar documents in a single request to the LLM [9]. Retrievers such as those from "LangChain" can be used to query a vector database. These are able to find and output matching documents from the database based on a query [10].

2.2 Existing projects

The "GPT-Codemaster" project is central to automating work with GitHub Issues. It utilizes the Octokit library for communication with GitHub and the NUnit framework for code quality assurance. The main objective is to offer automated solutions for analyzing and addressing GitHub issues. Not only can it suggest code modifications, but it also auto-generates pull requests, drastically reducing manual effort for developers. The system provides developers with progress updates and operates incrementally. Each step signifies a distinct action in a GitHub issue's resolution path, whether it's selecting files for review, modifying code, or initiating a pull request. The system's integration with OpenAI's GPT-4 is utilized for coding tasks, suggesting code alterations, and generating solutions. Additionally, the system incorporates robust error handling and logging mechanisms to ensure all actions are correctly recorded, and any unexpected outputs or errors are addressed appropriately [11].
An evolved tool in this area is AutoPR. Originally developed to generate pull requests (PRs) based on issues, AutoPR has evolved into a comprehensive framework specialized in orchestrating workflows within repositories. This system allows users to implement specified actions using Python and integrate them into coordinated workflows. Features include executing Bash commands, automatically generating ReadMe files, and interacting with the GitHub platform, among other things. The language model used in this project is GPT-3 from OpenAI [12].
In addition to the technologies already mentioned, there is a project called "Chat-with-your-codebase". This web application allows users to interactively communicate with their codebase via a chat interface. The whole process is enabled by a combination of different libraries and tools, including OpenAI,

DeepLake, git and Streamlit. "Chat-with-your-codebase" serves as a medium for users to connect directly to their GitHub repository and ask questions about their code. The system then analyzes the code, extracts relevant information, and provides answers based on the analyzed code. The Streamlit web application collects user input by having users enter their OpenAI GPT-4 key and link to their GitHub repository. The system then clones the repository and creates a vector database of code content. Users can then ask questions about their code, and the system provides answers based on the analyzed code. Chat history is maintained, allowing continuous dialog between users and the system [13].
Although these tools symbolize remarkable progress, they are not free of limitations. A significant limitation in the current technological state is the token limit of LLMs, which is currently limited to up to 8000 tokens (GPT-4) [14].
In conclusion, despite the significant recent progress in automating software development processes through LLMs, significant challenges still exist. Future developments in this sector will inevitably depend on the ability to overcome these hurdles and continuously expand the capabilities of LLMs.

3 Software-Architecture

The development of an automated system for processing issues from Github repositories using large language models requires a careful and methodical approach. The architecture of this system is divided into different phases, which are detailed in the following sections.

3.1 Github repository selection and criteria

To ensure the integrity and impartiality of the system, only repositories where researchers have no influence on solution acceptance or rejection will be selected. Care is taken to include a wide range of projects, from simple to complex, to ensure a comprehensive evaluation of system effectiveness. Only issues with sufficient information content will be considered. Redundant or sparsely described Issues could lead to distortions in system performance.

3.2 Initialization and data input

An initial console input is used to enter the URL of the repository to be edited. The project is then forked, creating an independent copy. This allows the contents of the repository to be modified without affecting the original. Console input:

- Specific Issue Selection: The interface allows users to specifically select a particular issue. Introduction of additional tags: To improve the response quality of the Language Model, users can provide specific tags via console input. These tags are used to provide additional context or specific directions to the model to increase the accuracy and relevance of the generated solutions.

– Specifying relevant files: To allow the system to know exactly which files or resources are relevant to the processing of the selected issue, users can specifically enter these file paths or names. This allows the system to focus on the actual resources.

3.3 Automatic detection of references in issues

In the description of an issue, it is common to refer directly to files or external resources using paths or URLs. These references often contain critical information that is necessary for the analysis and resolution of an issue. Therefore, a special detection module has been implemented in the system:
Using rule-based algorithms and pattern matching techniques, the system automatically identifies all file paths and URLs contained in the content of an issue. Once identified, these references are extracted and stored in a temporary data structure to keep them available for subsequent processing.
Replacing generic paths and URLs with concrete file names is a critical step to improve AI generation performance. Using file names directly instead of abstract paths allows the AI to better understand which files need to be customized.

3.4 Code Chunking

Breaking documents into smaller sections, known as chunking, is a fundamental strategy for efficiently organizing and retrieving information in documents.
The main goal of chunking is to make data easier to retrieve. When documents, especially voluminous ones, are divided into smaller units, it allows for targeted and quick retrieval of specific information. Instead of searching an entire document for a piece of information, one can access the relevant chunk directly.
Breaking code into chunks not only streamlines searching, but can also speed up processing. Small, manageable chunks of data can usually be processed faster than large, monolithic documents, especially when it comes to traversing, analyzing or modifying data [15]. LangChain's CharacterTextSplitter was used for this particular use case. This specialized tool allows for granular chunking of text data based on character count, while taking care to preserve the integrity and context of the text. In addition, each chunk is given metadata such as file name, path, and ID to make it easier to find in a subsequent search.
These chunks of data are systematically stored in a Deep Lake vector database for efficient query and retrieval.

3.5 Code extraction through vector analysis and data retrieval

In order to efficiently transmit the code to the LLM, only the essentials need to be sent due to the token constraint. For this purpose, we have developed a two-step procedure.
By analyzing the issue content, Deeplake performs a similarity search in the vector database to extract the paths of the thematically relevant documents.

OpenAI's embeddings are used for this purpose. These allow the creation of a vector representation of a piece of text. In the context of this work, a custom retriever was implemented to extract all relevant documents based on the identified file paths. This retriever was created using the LangChain library. The retriever takes a list of paths as input. Only those documents from the vector database whose source (path) is included in the path list are considered essential. Thus, the paths resulting from manual file input, file filtering, and similarity search are passed to the retriever to obtain the associated documents.

3.6 Prompt structure

The prompt, designed to interact with the LL-Model, serves as a concise guide to issue handling and code editing. This carefully worded prompt considers both the issue presented and the relevant code files. The central instructions and expectations of the prompt are:
The AI is instructed to solve the issue given in the text to the best of its ability using the provided code. It is also to process the existing code, using the functions and methods already in use, unless it is necessary to develop new ones. It should also maintain imports from other files in the project to ensure the integrity and functionality of the project. Output code should represent the entire document without abbreviations, even if changes are made to multiple files. At the end of the output, the AI is instructed to add a JSON object that contains the source from the original file's metadata. This facilitates later integration and tracking of changes.

```
Try to solve the issue in the given text as best you can with the given code.
Edit the existing code and also use the functions and methods already used, unless you need to develop new ones.
It is also very important that you keep imports from other files in the project.
Your output should show the whole file without truncations when modifying a file, even if changes are made in
multiple files.
Please make sure you add a JSON object at the end in the following format:
{
'source': '<source>'
}
Where 'source' is the source from the metadata of the original file.

{issue}

{code}
```

Fig. 1. The prompt used to solve the Github issue

3.7 Initialization and data input

An initial console input is used to enter the URL of the repository to be edited. The project is then forked, creating an independent copy. This allows the contents of the repository to be modified without affecting the original. Console input:

- Specific Issue Selection: The interface allows users to specifically select a particular issue.
- Introduction of additional tags: To improve the response quality of the Language Model, users can provide specific tags via console input. These tags are used to provide additional context or specific directions to the model to increase the accuracy and relevance of the generated solutions.
- Specifying relevant files: To allow the system to know exactly which files or resources are relevant to the processing of the selected issue, users can specifically enter these file paths or names. This allows the system to focus on the actual resources.

Langchain's LLMChain is used for communication with GPT 4. For this purpose, the prompt including the issue and the relevant code documents is transmitted.

3.8 Post-processing and code integration

A specially developed parser separates the generated code from the associated JSON. The extracted data is used to replace the original code with the revised code, using the source information in the JSON object as a reference.

3.9 Commit and Pull Request Workflow

This phase is critical in transitioning automatically generated code into a real-world context, serving as the capstone of the automated issue resolution system. Once the code is generated and parsed, it needs to be committed to the forked repository. This allows users to customize a commit message and branch via console input, providing an additional layer of control.
After committing changes, the system offers an option to create a pull request. Users can input specific details like title and body text before the pull request is automatically initiated, linking the forked repository to the original one.

4 Evaluation

4.1 Strategy

To assess the practical effectiveness and real-world utility of the novel code generation tool, a comprehensive evaluation was conducted based on an iterative and open-source approach.
The evaluation process involved submitting pull requests with code generated

by the tool to various open-source software repositories across diverse domains within computer science. Successful integration of a pull request into the main repository served as concrete evidence of the tool's capability to produce high-quality, functional code that met the project's requirements.
In instances where a pull request was not integrated, feedback from the project contributors was leveraged to continuously refine the tool. This iterative approach not only measured the tool's current performance but also ensured its continued relevance in real-world software development scenarios.
Using the criterion of successful pull request integration ensures that the generated code seamlessly fits into the existing codebase and adheres to project standards. This emphasizes the tool's practical applicability in real-world software development scenarios where seamless code integration is paramount.

4.2 Results

During the evaluation of the Github-Issue-Solver tool, a total of nine pull requests were submitted to various open source software repositories.
During the evaluation of the Github issue solver tool, it was found that the structure and presentation of a Github issue has a significant impact on the effectiveness of the solutions generated by GPT-4. These findings are not based solely on the pull requests analyzed, but also on the processes used to select appropriate issues. The resulting best practice recommendations for structuring a Github issue are as follows:

- An introductory section should describe the underlying problem or requirement in detail, yet concisely.
- Method handling: this should indicate if and how existing methods should be modified or if new methods should be created.
- File references: Concrete file names or direct links to the relevant lines of code in the repository are essential for targeted editing.
- Resource details: Detailed information on required libraries, frameworks or specific functions.
- Generation details: Clear instructions about what exactly is to be generated by GPT-4.
- Implementation specifics: details of any special requirements, such as the creation of helper functions or special implementation details.
- Additional notes: Any other relevant information that might contribute to the generation of a goal-oriented solution.

The first pull request encountered a challenge: the tool's proposed solution for the Github issue did not use the project-specific function, but generated its own. This constraint stemmed from LangChain's retriever, which uses maximum marginal relevance (MMR) to find the right documents. Thus, the retriever was not able to retrieve the relevant documents that could clarify the meaning of the functions used.
To address this, two major adaptations were undertaken:

- The Retriever was upgraded from LangChain to the more efficient Deep Lakes Similarity Search, enhancing its capability to fetch pertinent documents.
- The prompt was refined to emphasize the importance of retaining and leveraging project-specific functions.

When looking at the repository of sipgate.io, sipgateio-google-deployer, the first limitations of this program were noticed. Despite the fact that the issues are not particularly complex, the use of external tools such as AWS, Azure, GCloud CLI, etc. is disadvantageous because they rarely provide useful results. Here the implementation of the tools can vary from project to project and things like updating versions, or adjustments in the implementations become inaccurate, since for an accurate generation of such codes often a lot of context must be provided, which is currently not possible.
Subsequent to these modifications, eight pull requests, including to the project with the failed PR, were successfully integrated into their respective repositories. The addressed Github-Iissues spanned diverse domains, such as replacing models in deep learning contexts, making React components responsive and extending the context menu in a React Electron app.
Two of the successfully addressed issues originated from a big project, namely invisal's query-master. The remaining 4 solutions catered to smaller, private projects.
However, the results also highlighted inherent challenges due to GPT-4's token limit.
If the context provided to the LLM already includes several thousand tokens, the response returned by GPT-4 could be a truncated version of the actual code. This is a significant limitation and illustrates the difficulty of obtaining a complete and coherent code output from GPT-4.
The result of this truncated code output is that inserting the new code into existing code bases becomes a complex task. It becomes particularly problematic if the generated code does not cover the entire context. There could be overlaps or remnants of old methods left in the code that lead to errors.
The high token consumption during input ultimately also affects the quality of the output, and in some cases no code is output. As a result, the tool's outputs should be reviewed by people to ensure accuracy and contextual relevance.

5 Summary and outlook

This paper examined the automated solving of GitHub-Issues through the usage of the LLM GPT–4. The study's objective was to assess how well LLMs like GPT-4 perform in tasks like solving GitHub-issues and to determine if their provided solutions match the quality and criteria of human provided solutions. Initially, the significance of efficiently managing issues reported in GitHub-repositories and its influence on the quality of software development was elucidated. It was greatly emphasized that the automated handling and solving of GitHub-Issues would accelerate the software developing process as the time for interacting

with user issues could be delegated to an LLM-powered tool. Furthermore the potential occurrence of human error and potentially long waiting periods would be reduced as the LLM-powered tool would automatically handle the GitHub-Issues. Working with LLMs showed that it is necessary to provide a software context that is specific to the issue as the results proved to be more precise and helpful. Additionally this approach was used to evade a token limit problem when providing a proper software context, as the usage of the GPT-4-model is currently restricted to a token limit of 8.000. Additionally, it was found that the quality of the responses strongly depends on the quality of the issue description. Consequently, the LLM provides better results when a well-founded issues template is used for the repositories. The chapter "State of the art" describes similar LLM-powered tools that are being used for comparable use cases and highlights alternative approaches that were utilized by the tool providers. In the chapter "Software-Architecture" it is emphasized that only GitHub-repositories, which are not linked to the researchers regarding the selections of fitting repositories, were chosen. This was done to prevent any personal bias, which could possibly distort the results. The evaluation approach consisted of using the LLM-powered tool to work on and resolve GitHub-Issues from open-source repositories, ultimately achieving a successful pull request followed by a merge. The results have shown that none of the eight pull requests were rejected, however xy were successfully accepted . This demonstrates that the LLM-model GPT-4, can be successfully utilized in handling GitHub-Issues. Further advances like the deployment of the GPT-4-model, which enables the usage of 32.000 tokens, will likely improve the results of solving GitHub-Issues, as a larger software context could be provided to the LLM. The examination of possible advantages through the utilization of larger context-models of GPT-4 should be conducted in future studies to assess whether the size of the context indeed makes a difference.

References

1. Author, F.: Article title. Journal **2**(5), 99–110 (2016)
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. Author, F., Author, S., Author, T.: Book title. 2nd edn. Publisher, Location (1999)
4. Author, A.-B.: Contribution title. In: 9th International Proceedings on Proceedings, pp. 1–2. Publisher, Location (2010)
5. LNCS Homepage, http://www.springer.com/lncs. Last accessed 4 Oct 2017