RAG Based Documentation Generation from code files, A Case Study on LLM for open-source projects

Sujaykumar Reddy M[1]

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India

November 25, 2024

Abstract

In the year 2022, developers initiated a total of 52 million novel open-source projects on the GitHub platform. Furthermore, the developer community on GitHub actively contributed to open-source initiatives, accumulating a noteworthy sum of 413 million contributions during the same period. As the number of open-source projects grows, so too does the number of issues that arise. These issues require a considerable amount of time and effort to resolve. However, what if a Language Model with extensive knowledge of the project, including its libraries and programming languages, was at your disposal ? Utilizing such an LLM for a specific project could help developers resolve issues with ease and eventually automate the entire process. Established Language Models (LLMs), such as Microsoft Co-Pilot, represent valuable assets in the pursuit of automation for this purpose. Our primary objective is to forge a seamless interface between developers and the system, thereby enhancing developer agility, project scalability, vigilance over potential security vulnerabilities and majorly providing documentation to the developers. It is crucial to note that formulating a solution for this problem statement transcends the conventional methods of Fine-tuning or Prompt Engineering.

1 Introduction

With the rise of Large Language Models (LLMs), developers are using these advanced tools to improve different aspects of software development. Companies like Accenture and Bloomberg are creating customized LLMs for their developers. This shift signals a change in the tech landscape, showing how LLMs empower developers with better language processing, making software development more efficient. Documentation is crucial in Software Engineering, especially in Agile Development, a standard that covers the Software Development Life Cycle (SDLC) (3). Agile Documentation acts as a comprehensive resource, explaining the details of systems, products, or services. It helps users understand, use, and maintain these components and facilitates communication among developers, users, and maintainers.

In software documentation, two main stakeholders are essential – End-Users and Developers. End-Users directly interact with the software, relying on documentation to optimize their usage. Developers use documentation to understand the codebase, APIs, and design principles. This dual

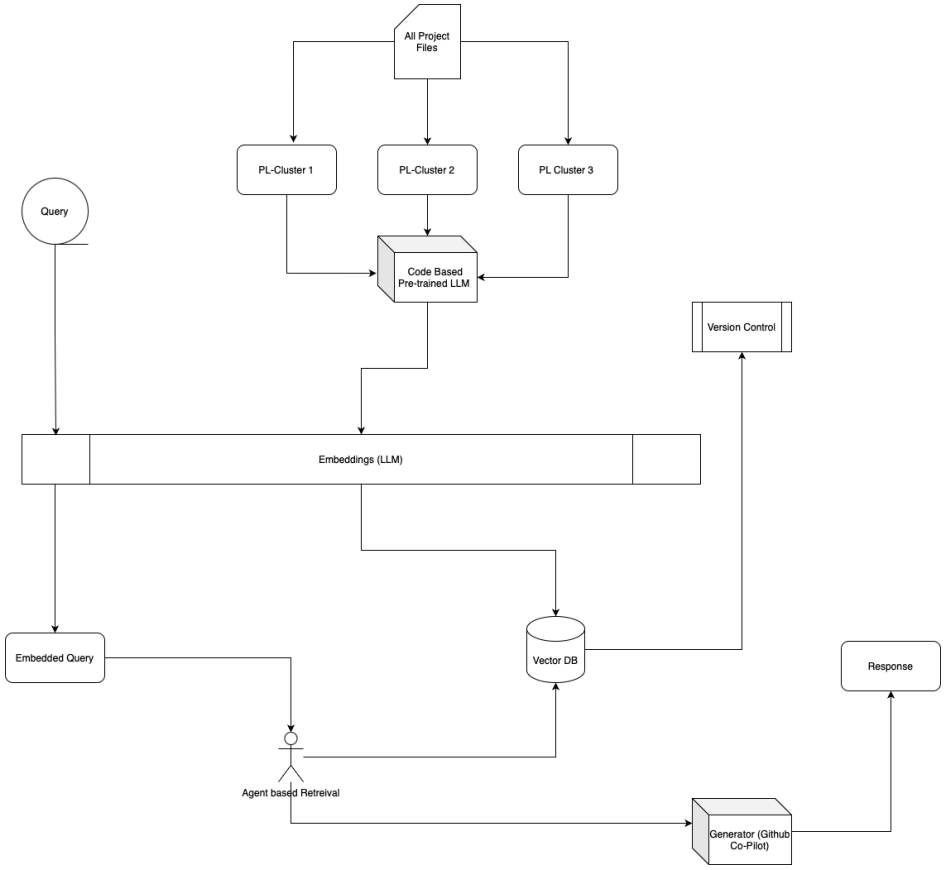

Figure 1: Depicts the Proposed Architecture of the RAG based LLM using Code files

role of documentation acts as a bridge between those who use and those who create the software. From a cost perspective, more than 68% of respondents reported that creating and maintaining an in-house documentation system costs over $10,000 (4). To simplify and automate the process, we propose a new Retrieval-based Code Documentation system for various open-source projects and programming languages. This approach helps developers build new software versions on existing codebases, providing users easy access to both code and documentation. The goal is to enhance software development efficiency with a more user-friendly interface for both developers and end-users.

2 Proposed Methodology

This paper introduces a novel RAG (Retrieval-Augmented Generation) system, designed to minimize hallucinations and deliver answers within a well-defined context. Figure 1 illustrates the architecture of the proposed model specifically employed for documentation purposes.

2.1 Retreival Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) is a retrieval system for the input prompt to augment the output generated by the LLM. This technique allows us to bypass fine-tuning as we can easily expose the model to external data (non-parametric), instead of having to retrain it on our domain-specific data. Here we use input files as all the programming files in the project with the metadata (which can be commented):

`{"name":"main.py","loc":"Desktop/project","datecreated": etc ...}`

This helps the Dense Passage Retrieval model (or) Document Retriever to understand the project structure. When the Developer or the User updates the code or starts a new project with our LLM, the model will have a vague knowledge of the project and then once he/she commits the code using the model, we compute the Document index and centroids for that particular commit. Once the following computations are performed, when the user issues a query, documents are retrieved utilizing the centroids. These retrieved documents are then passed on to the generator, which generates new sequences conditioned to fall within the range between the project. However, in contemplating this problem statement and exploring the solution space, several potential disadvantages emerge. As we delve into the solution space, we encounter a shift towards reliance on language models as opposed to traditional developer documentation.

2.2 Case Study on Open-Source Projects

Embarking on our exploration of the RAG-based Large Language Model (LLM) in the context of open-source projects, we initiate this journey with a focus on its applicability. In the project's early stages, where developers may not require extensive data, the vector database remains devoid of embeddings. Notably, the project files, situated within the project folder hierarchy, play a pivotal role in optimizing retrieval and RAG capabilities over intricate document sets (5). This unique approach leverages the inherent hierarchy and structure of the project folder, eliminating the need for the explicit creation of semantic or syntactic hierarchies.

Original Code

```
class NeuralNetwork(nn.Module):
    def __init__(self,input_size,
    hidden_size):
        super(NeuralNetwork,self).
    __init__()
        # defining layers
        self.lin1 = nn.Linear(input_size,
    hidden_size)
        self.lin2 = nn.Linear(hidden_size
    ,1)

        # define Activation Function
        self.relu = nn.ReLU()

        # define Output layer
        self.sigmoid = nn.Sigmoid() # (or
    ) nn.Softmax()


    def forward(self, x):
        out = self.lin1(x)
        out = self.relu(out)
        out = self.lin2(out)
        out = self.sigmoid(out)
        return out
```

After Pretrained Model Generation

```
class NeuralNetwork(nn.Module):
    """
    Neural Network model with specified input size and hidden size.

    Args:
        input_size (int): The size of the input layer.
        hidden_size (int): The size of the hidden layer.

    Attributes:
        lin1 (nn.Linear): The first linear layer.
        lin2 (nn.Linear): The second linear layer.
        relu (nn.ReLU): The ReLU activation function.
        sigmoid (nn.Sigmoid): The sigmoid activation function.

    """

    def __init__(self, input_size, hidden_size):
        super(NeuralNetwork, self).__init__()
        # defining layers
        self.lin1 = nn.Linear(input_size, hidden_size)
        self.lin2 = nn.Linear(hidden_size, 1)

        # define Activation Function
        self.relu = nn.ReLU()

        # define Output layer
        self.sigmoid = nn.Sigmoid()  # (or) nn.Softmax()

    def forward(self, x):
        """
        Forward pass of the neural network.

        Args:
            x (torch.Tensor): The input tensor.

        Returns:
            torch.Tensor: The output tensor.

        """
        out = self.lin1(x)
        out = self.relu(out)
        out = self.lin2(out)
        out = self.sigmoid(out)
        return out
```

The project folder inherently delineates the use of various programming languages in an open-source project, facilitating clustering, such as Python_Cluster 1, Python_Cluster 2, Bash_Cluster 3, and so forth. Now, considering the project hierarchy and segmented data with the multi-programming Language Model (LLM) capability, our LLM is adept at understanding user queries related to specific programming languages. This proves invaluable for tasks like updating Docker files at precise locations in the hierarchy or addressing modifications in markdown files. In the documentation domain, we enhance context using a Code-based Pre-trained LLM to generate function documentation. Figure 2 illustrates the Copilot pretrained LLM, showcasing its effectiveness in generating documentation for a simple PyTorch Neural Network before and after the documentation generation process. Following this, we proceed to compute embeddings and store them in a vector database. This database includes clusters (also known as centroids) and document indices (storing specific documents or code snippets generated by the RAG-based LLM).

3 Conclusion

When a developer queries the model, it reliably produces responses without hallucinations. However, challenges arise in a multi-user scenario, necessitating re-running the model for each commit and maintaining version control (excluding the need for the previous model). Focusing on developers exclusively, we can bolster the model's strength by incorporating multiple Code Language

Models (LLMs), and the generator can range from GPT-2 to a fine-tuned GPT-4. Previous research indicates that utilizing fewer parameters can yield faster and more dependable results. Additionally, incorporating self-reflection mechanisms allows continual improvement of the answers generated by the model. This approach ensures adaptability and efficiency, particularly in collaborative development environments with multiple users.

References

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, Ming-Wei Chang. *REALM: Retrieval-Augmented Language Model Pre-Training*. 2020. arXiv:2002.08909 [cs.CL].

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv:2005.11401 [cs.CL].

How Do You Create and Manage SDLC Documentation? Available: https://www.linkedin.com/advice/0/how-do-you-create-manage-sdlc-documentation

Documentation Website Costs., Available: https://www.archbee.com/blog/documentation-website-costs

The Untold Side of RAG: Addressing its Challenges in Domain-Specific Searches., Available: https://towardsdatascience.com/the-untold-side-of-rag-addressing-its-challenges-in-domain-specific-searches-808956e3ecc8