Producing software documentation from code snippets using generative AI

University of Pisa

MARCO PARDINI

A.Y. 2022-2023

OBJECTIVE

The objective of this paper is to explore the use of generative AI techniques to automate the creation of software documentation from code snippets. Traditionally, software documentation has been a time-consuming and error-prone process that requires a significant amount of manual effort. However, recent advances in machine learning and natural language processing have opened up new possibilities for using generative AI to automatically produce high-quality documentation that accurately describes the functionality of software code. By leveraging these techniques, the objective is streamlining the documentation process and improve the efficiency and accuracy of software development.

WHY PRODUCING DOCUMENTATION IS SO EXPENSIVE?

Companies typically spend a significant amount of time and resources on hiring individuals to write documentation for the code they produce. This can be a challenging and time-consuming process, as it requires finding individuals who have a strong technical background, good writing skills, and the ability to effectively communicate complex technical concepts. Furthermore, documentation is often viewed as a lower priority task compared to coding, and as a result, documentation efforts can sometimes be neglected or overlooked entirely. However, high-quality documentation is critical for ensuring that software is maintainable, understandable, and usable over the long term.

Here are some examples of expenses that companies may incur when hiring people to write documentation:

1. Salaries and benefits: Companies typically pay salaries and benefits to technical writers or other individuals responsible for writing documentation. These costs can add up quickly, particularly if multiple writers are needed to produce documentation for a large software project.
2. Training and onboarding: Even experienced technical writers may require training and onboarding when starting a new job, which can be time-consuming and expensive for companies.
3. Editing and review: Documentation typically goes through multiple rounds of editing and review to ensure accuracy and clarity, which can involve additional expenses for companies.
4. Delays in software development: If documentation is not completed in a timely manner, it can delay the release of software, which can result in lost revenue and additional expenses.
5. Translation costs: If documentation needs to be translated into multiple languages for international audiences, this can be an additional expense for companies.

By using generative AI to automate the documentation process, companies can potentially save money on these expenses while also improving the overall quality and efficiency of their documentation efforts.

There have been several studies and surveys that have shown the importance of documentation in software development.

IEEE released a paper in 2003, interviewing 12 corporate sites and one government site, along with a large telecommunications company. The paper highlights the fact that *“Most software documentation is not updated consistently, but out-of-date documentation might remain useful. We must find powerful yet simple documentation strategies and formats that software engineers will likely maintain.”*[1]

Generative AI

Generative AI is a branch of artificial intelligence that focuses on creating machines capable of generating original content autonomously. Rather than being explicitly programmed, generative AI models are trained on large amounts of data to learn patterns and generate new data based on those patterns. This can include generating text, images, music, and even videos.

Generative AI has many potential applications, including in the fields of natural language processing, computer vision, and robotics. For example, generative AI can be used to generate realistic-looking images or videos, or to generate creative writing such as poetry or short stories. It can also be used to develop more advanced chatbots and virtual assistants, which can provide more natural and human-like interactions with users.

However, there are also potential ethical concerns with the use of generative AI, particularly around issues of bias and the potential misuse of the technology for malicious purposes. As such, it is important to carefully consider the potential implications of using generative AI and to take steps to mitigate any potential risks.

Models used in this project

Text-Davinci-002 is a language model developed by OpenAI that uses the latest generative AI technology to generate human-like text. It is based on the GPT-3 architecture, which is known for its ability to generate high-quality natural language text. Text-Davinci-002 has been trained on a vast amount of data, which allows it to produce highly coherent and contextually relevant text.

Other than generative AI to produce software documentation, other techniques will also be employed. One of these techniques is the cosine similarity measure, which will be used to compare the similarity between the code and the generated documentation. This technique is often used in natural language processing to compare the similarity between two documents.

Another technique that will be used is a language tool to check the number of grammar errors in the generated text. This is important for ensuring that the generated documentation is not only coherent and contextually relevant but also grammatically correct. By combining these techniques with

[1] https://ieeexplore.ieee.org/document/1241364

generative AI, it is possible to produce high-quality software documentation that is both accurate and easy to understand.

Example of documentation production

Consider the following piece of code, written in Python language:

```
def factorial(n):
if n == 1:
return 1
else:
return n * factorial(n-1)
```

Using generative AI and natural language processing techniques, it is possible to automatically generate documentation for this code. For example, the following documentation was generated using Text-Davinci-002:

The `factorial` function takes a positive integer `n` and returns the factorial of `n`. If `n` is equal to 1, the function returns 1. Otherwise, it multiplies `n` by the factorial of `n-1` and returns the result. This implementation uses recursion to compute the factorial, which is a common approach for this problem.

We can see that the generated documentation accurately describes the purpose and functionality of the code. The language is coherent, contextually relevant, and grammatically correct, which is a testament to the capabilities of generative AI and natural language processing.

Example of text evaluation

Consider the following two sentences:

1. "The cat sat on the mat."
2. "The moon is made of cheese."

We can easily understand that the first sentence is semantically correct, as it describes a common and plausible scenario that can be easily visualized. On the other hand, the second sentence is semantically incorrect, as it describes a scenario that is not supported by scientific evidence.

If we ask to ChatGPT to perform a semantic evaluation of these 2 sentences, this would be the answer:

"Based on the semantical correctness, the first sentence obtained a mark of 9 out of 10, while the second one obtained a mark of 2 out of 10. The reason for the difference in marks is that sentence 1 accurately describes a plausible scenario that conforms to our understanding of the world, while sentence 2 makes a claim that is not supported by any empirical evidence. Semantic correctness is determined by how well a sentence reflects the reality that it describes, and in this case, sentence 1 does a much better job than sentence 2."

If we propose ChatGPT to evaluate the documentation produced for the factorial example, the result would be again a 9 out of 10, and if we ask for the reason for such choice, this would be the answer:

*"The documentation accurately describes the purpose and functionality of the **factorial** function, and the language used is clear, coherent, and grammatically correct. The only minor issue is that the documentation does not explicitly mention that the **factorial** function is only defined for positive integers. However, this can be inferred from the fact that the function uses recursion, which requires a base case of 1 to terminate the recursion. Overall, the documentation is of high quality and provides a clear explanation of the **factorial** function."*

WORKFLOW OF THE PROJECT

In order to evaluate the effectiveness of the generative AI model for producing documentation from code snippets, the decision was to conduct a series of tests using a selected dataset and a testing pipeline. In this next paragraph there will be a description of the details of the testing pipeline, including the dataset used, the process of data cleaning and preparation, and the specific evaluation metrics employed to assess the performance of the model.

DATASET CHOICE

The dataset used for our tests was the code_blocks_21.csv file from Zenodo, which contains a collection of code snippets written in various programming languages. This file is approximately 100 MB in size and contains over 3 million rows of code. The code snippets in this dataset are written in a variety of languages, including Python, Java, C++, and JavaScript. I chose this dataset because it provided a diverse range of code examples that allowed us to evaluate the performance of the generative AI model across multiple programming languages. Additionally, the size of the dataset ensured that the tests were comprehensive and representative of real-world scenarios. Prior to running tests, a process of data cleaning and preparation was performed to ensure that the code snippets were of high quality and suitable for evaluation. In particular, the decision was to focus exclusively on Python code snippets from the code_blocks_21.csv dataset, as Python is one of the most commonly used programming languages in industry and academia.

DOCUMENTATION GENERATION USING DAVINCI MODEL

In the next phase of our evaluation, the decision was to use the Davinci model from OpenAI to generate software documentation for the Python code snippets in our selected dataset. The Davinci model is one of the most advanced language models currently available, capable of generating high-quality natural language text in a wide range of domains and styles. By leveraging the capabilities of this generative AI model, I aimed to produce software documentation that was both informative and semantically accurate, providing a valuable resource for developers and engineers alike.

To conduct our evaluation of generative AI for software documentation, a free trial of the Davinci model offered by OpenAI was used. Due to the limitations of the free trial, I decided to set a maximum number of 500 tokens for the generated documentation, ensuring that the output was both concise and informative. After running the model on the selected Python code snippets, the obtained dataset consisted of 691 pairs of code and corresponding generated documentation.

Evaluation

To evaluate the quality of the generated software documentation produced by the ChatGPT model, a combination of methods was used. First, the semantic correctness of the generated text was assessed by using the OpenAI APIs to automatically evaluate the output and assign a semantic score based on its accuracy in conveying the intended meaning of the corresponding code snippet. Second, a grammar tool was used to evaluate the grammatical correctness of the generated text, looking for errors in spelling, punctuation, and syntax. Finally, the cosine similarity between the code snippet and the corresponding generated documentation, to measure the degree to which the generated text accurately captured the essence of the code. By using these methods in combination, we aimed to provide a comprehensive evaluation of the quality of the generated software documentation, taking into account both its semantic and grammatical accuracy, as well as its relevance to the original code.

Data cleaning post evaluation

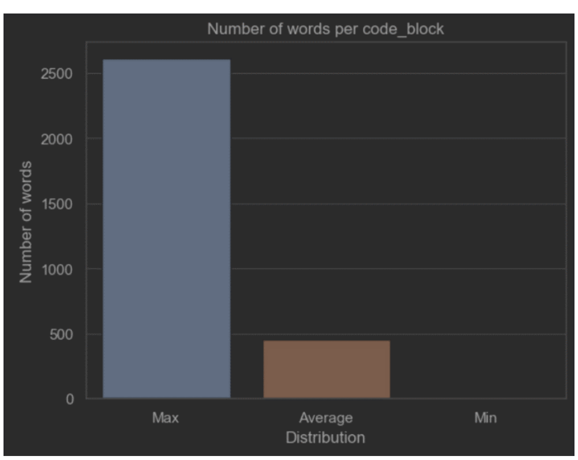
During the evaluation process of the generated documentation, some of the evaluations provided by the OpenAI API were in the form of text instead of numerical values. As a result, these evaluations could not be used in the analysis, and therefore were discarded. The reason for this was that numerical values were needed to compute the accuracy of the generative AI model and its comparison with the cosine similarity and grammar tool evaluation. This data cleaning process allowed us to ensure that only valid numerical evaluations were used in the analysis, making the results more reliable and accurate.

Results

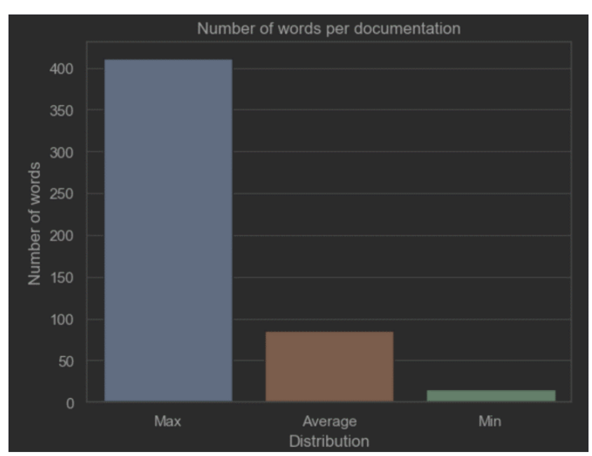
After performing generative AI-based documentation on the code snippets dataset and conducting various evaluations, some interesting results were obtained. In this section, the overall performance of the generated documentations will be discussed, the accuracy of the grammar tool used, and the effectiveness of the cosine similarity evaluation method.

Some insights

After analyzing the code snippets and their corresponding documentation, some insights can be drawn.



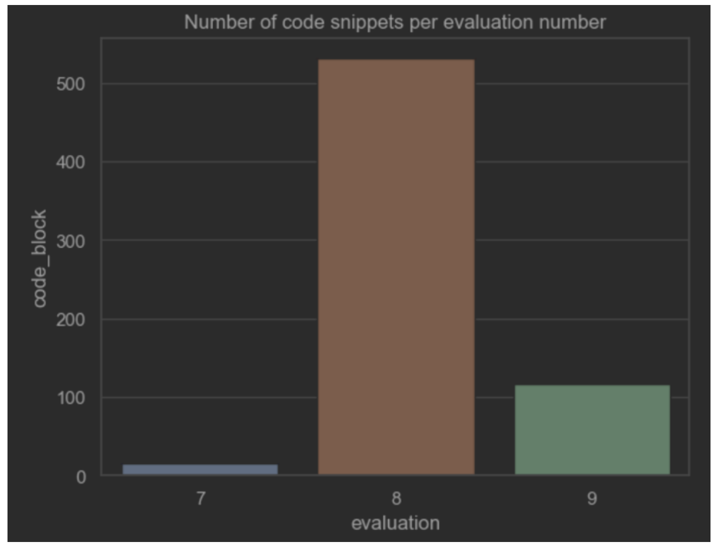
In



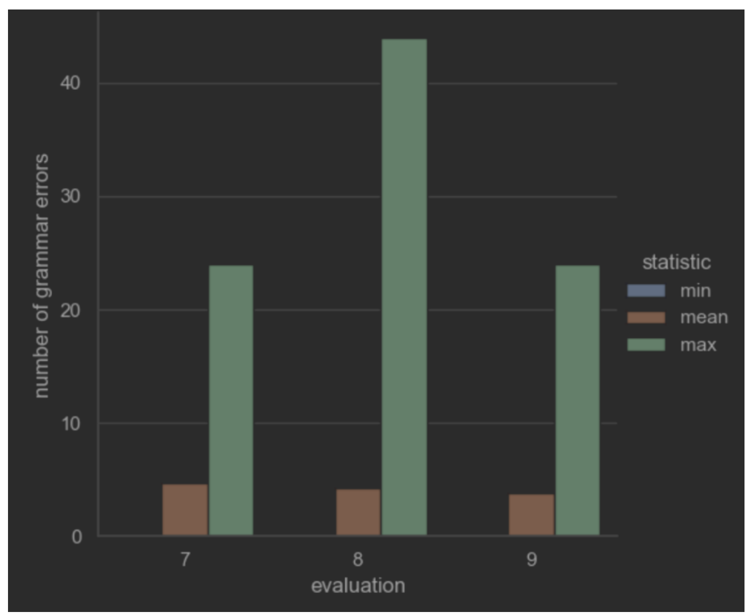
terms of the number of words per code block, the maximum number of words is 2609, while the minimum is 4, with an average of 450 words. On the other hand, the number of words per documentation ranges from 16 to 411, with an average of 85 words. These findings suggest that the produced documentation tends to be shorter than the code blocks themselves, which may indicate a need for more concise and precise documentation. Let's take a deeper look at this.

EVALUATION RESULTS

In terms of the semantic evaluation, we found that 531 documentations obtained a mark of 8, 15 code blocks obtained a mark of 7, and 116 obtained a mark of 9. It is worth noting that all of the marks given were above 6, which is considered sufficient, and not even a single documentation received a perfect score of 10. While there were some variations in the marks assigned, the majority of the documentations performed well in the semantic evaluation, indicating that the generative AI model produced fairly accurate and coherent documentation for the given code snippets.

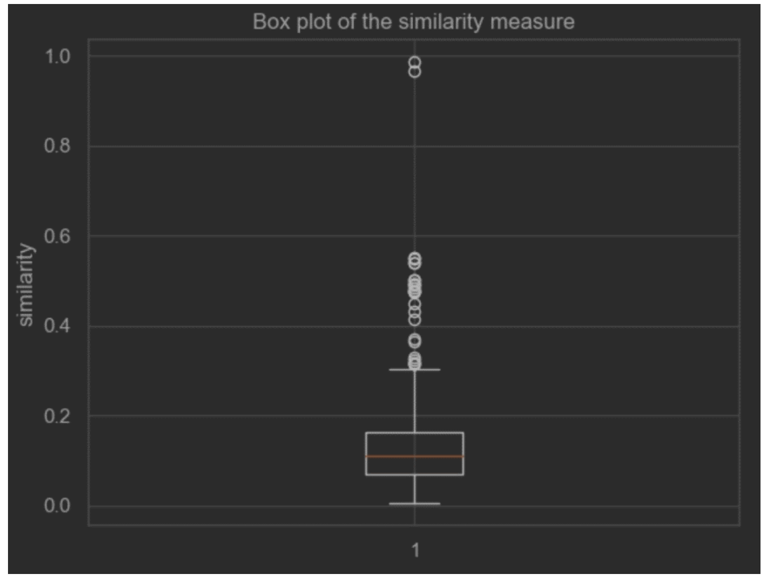


For the grammatical evaluation, the results were grouped by the semantic evaluation vote. The min, mean, and max values were calculated for each group. For the group with a semantic evaluation vote of 7, the min was 0, the mean was 4.73, and the max was 24. For the group with a semantic evaluation vote of 8, the min was 0, the mean was 4.25, and the max was 44. For the group with a semantic evaluation vote of 9, the min was 0, the mean was 3.77, and the max was 24. These results suggest that there is not a clear correlation between semantic correctness and grammatical correctness.



For instance, some code blocks with a low semantic correctness score had a higher grammatical correctness score, and vice versa. This indicates that even though the code documentation may be grammatically correct, it may not convey the intended meaning or be accurate. Additionally, there may be other factors at play that affect grammatical correctness, such as the complexity of the code or the style of the programming language used. It is important to note that the grammar tool used only checks for basic grammatical errors and does not take into account the context of the documentation (for example all function calls were considered to be grammar errors, since they're written line function_call().)

Overall, the results suggest that while grammatical correctness is important, it is not the sole factor in determining the quality of code documentation.



The evaluation through cosine similarity between code snippet and documentation was not found to be a good way of evaluating the documentation. The results showed that the mean similarity measure was only 0.15, with a maximum of 0.25 in the boxplot, even if some outliers went up to 1. This low similarity score can be attributed to the fact that the code snippets and documentation use different vocabularies, making it difficult to obtain high similarity scores. Additionally, even if the documentation is semantically correct, it can still use different vocabulary or style than the code snippet, and this would affect the similarity score. Therefore, cosine similarity should not be used as the sole measure of the quality of the documentation produced by ChatGPT.

SOME EXAMPLES OF CODE BLOCKS AND RELATIVE DOCUMENTATION

In order to provide a more in-depth analysis of the produced documentations, a closer look will be given to three specific examples. One of them has obtained a vote of 7 in the semantic correctness evaluation, one of them obtained a vote of 8 and the last one obtained a vote of 9. For the sake of brevity, we won't include the entire code block in this analysis as it would be too long.

“This code imports the necessary libraries for data analysis and visualization, then loads the data from the CSV files. Next, it defines a function for visualizing data in a bar plot. Finally, it groups the data by target label and displays the counts in a bar plot.”

This generated documentation obtained a semantic correctness mark of 7 out of 10. If we ask to ChatGPT the reason, this would be the answer:

“I gave a mark of 7 out of 10 for the semantic evaluation of this documentation because it accurately describes the purpose of the code and the steps taken in the analysis. However, it could be improved by providing more specific details, such as the names of the libraries being imported, the file paths of the CSV files being loaded, and more information about the visualization function being defined. Additionally, it could benefit from a brief explanation of what the target label represents and why it is important to group the data by this label.”

Let’s take a loot at the next produced documentation, that obtained a mark of 8 out of 10.

“This code defines a MLP class which inherits from the nn.Module class. The class has three methods:

__init__ : This method initializes the class and defines the three linear layers of the MLP.

forward : This method defines the forward pass of the MLP. It takes an input tensor and passes it through the three linear layers. The output of the last linear layer is passed through a log softmax function.

fit : This method is used to train the MLP. It takes as input the train loader and the model. It defines the optimizer and the error function. It then loops over the training data for a number of epochs and saves the model with the lowest loss."

The reason of the semantic evaluation of this documentation are the following ones:

"There are a few possible reasons why the documentation obtained a vote of 8 out of 10 for semantic evaluation. One reason could be that the documentation provides a clear and concise summary of what the code does and how it works. It defines the MLP class, explains its three methods, and describes how the fit method is used to train the model."

Now, the last documentation to be commented:

"This code imports the necessary libraries for data analysis and visualization, then loads the training and testing data sets into pandas dataframes. Next, it checks for any missing values in the data sets and exits the code if any are found. Finally, it assigns the label column from the training data set to the variable "y" and the remaining columns to the variable "x"."

This documentation obtained a mark of 9 out of 10. Let's see how ChatGPT justifies this:

"The documentation obtained a mark of 9 out of 10 in terms of semantic evaluation because it provides a clear and concise explanation of the code's purpose and functionality. The documentation describes each step of the code, from importing necessary libraries to assigning variables, in a logical and organized manner. The only possible reason for not achieving a perfect score is that it does not mention the purpose of each imported library and its role in the code, which could be helpful for users who are not familiar with the libraries. However, this is a minor issue, and overall the documentation effectively communicates the purpose and functionality of the code."

CONCLUSIONS

In conclusion, this paper highlights the potential of large language models, such as GPT-3.5, to automate and streamline software documentation processes. The experiment conducted in the paper shows that GPT-3.5 is capable of producing accurate and comprehensive documentation for code blocks in a matter of seconds, which would take humans hours to produce.

A little less than 700 code snippets were analyzed, and the mean of the semantic mark was a little below 9. It's safe to conclude that the produced documentation likely obtained a high score in terms of semantic evaluation. The fact that the AI model was able to produce such high-quality documentation in just a matter of seconds is truly remarkable and could potentially revolutionize the software industry by saving developers hours of time in the documentation process.

However, it's important to note that there may be some limitations to the use of large language models in software documentation. For example, the model may struggle with highly technical or domain-specific language, leading to inaccuracies or omissions in the produced documentation. Additionally, the model may not be able to capture the nuances and insights that human developers can provide when documenting their own code.

Another problem may be the fact that a software is not only represented by the written code, but also by the idea behind it, the framework chosen, the language, and other important aspects that GPT may not be capable of conveying in the documentation, since it only analyses the written code.

It would be interesting to give to GPT an entire project of several lines of code and different classes, to understand if it’s capable of working also with large projects and not only snippets of code, understanding the relation between different modules of the application. Unfortunately this can’t be done with the trial version offered by OpenAI.

Overall, the results of the experiment in the paper suggest that large language models like GPT-3.5 can be a valuable tool for automating and streamlining software documentation processes. While there may be limitations and challenges to their use, they offer a promising solution for companies looking to save time and resources on documentation tasks.