TheCommentors
Pranav Kannan Hariharane
Richard Steve Cruz-Silva

Overview: With the rise in popularity of LLMs, almost everyone uses them now for a wide range of applications. Our experiment came from the struggle that LLMs face in performing well in the code generation application. Through prior research, we know that LLMs struggle to accurately produce code for complex and difficult programming problems that represent realistic software development scenarios. Thus, we chose to explore how well LLMs can generate comments for realistic software development code since we believe that may provide some insight into how well LLMs can understand and reason about these more complicated code scenarios. By prompting the GPT-4 LLM to generate comments for class-level code examples from the ClassEval benchmark [1], we can see exactly how well LLMs are able to understand complex code and all the contextual dependencies involved in it such as third-party libraries, other class methods, class attributes, etc. We hope the results of our experiment and research can provide insights into the future steps that can be taken to improve how LLMs perform in code generation for software development tasks.

Research Questions

1. How well can a pre-trained code LLM generate high-quality useful comments for methods in class-level code?
    a. We will evaluate the comments generated by the LLM with the following methods
        i. Human Evaluation: We will each assign a score to the output comments generated by the LLMs
            1. Score will be 1, 2 or 3
                a. 1 = poorly generated comments that don't capture the intent of the code in the method
                b. 2 = decently generated comments that do an okay job of capturing the intent of the code in the method
                c. 3 = well generated comments that do a great job of capturing the intent of the code in the method
            2. We will average the scores assigned by both of us and use this averaged score as a metric to evaluate the performance
        ii. BLEU score

$$\text{BLEU} = BP \times \exp \frac{1}{N} \sum_{n=1}^{N} \log p_n$$

Where:

$$p_n = \frac{\text{Number of ngram tokens in system and reference translations}}{\text{Number of ngram tokens in system translation}}$$

And:

The brevity penalty = $\exp(1 - r/c)$, where c is the length of the hypothesis translation (in tokens), and r is the length of the *closest* reference translation.

1.

        a. Word and phrase similarity checking metric
        b. Since this metric doesn't really measure correctness of the comments, we will place more weight on the Human Evaluation metric
2. How well can LLMs capture the contextual dependencies present in the code through their comments?
    a. Contextual Dependencies = things that the class methods depends on to behave correctly but are also not passed into the method explicitly
        i. Class attributes, third party libraries, other methods in the class, etc.
    b. We will produce qualitative results analyzing how well the LLM generated comments are able to capture the contextual dependencies present in the classes that have them.
        i. We will look through the LLM generated comments manually to see how well the LLMs are able to notice and document these contextual dependencies in their comments

Value to User Community: We believe evaluating exactly how well LLMs are able to generate high-quality comments on our provided class-level code can indicate how well LLMs are able to understand and reason about more complex and realistic code. Thus, we hope the results of our experiment and research can provide insights into the future steps that can be taken to improve how LLMs perform in code generation for software development tasks. Also, we can comment on legacy code to increase its maintainability. By commenting on legacy code, it will provide examples for those unfamiliar with the legacy language that will be quickly understood. It will be able to comment on tasks such as opening and editing a file or traversing some list or map object. These examples will aid the project maintainer in learning the language and the specific project. Commenting on legacy code can also aid in the process of modernizing it while increasing the likelihood of preserving all the original functionality. The developer updating the code should not only use code comments to ensure full functionality conversion but it will be a useful tool in the updating process. Documenting large software projects will also be useful to onboard new developers faster. Organizations will be able to quickly comment on software projects if they lack the existing documentation as a starting point to make the process of creating the documentation more efficient. Documentation is crucial to having useful projects so this will be a good indicator of whether LLMs can be reliably used to generate comments for software development projects.

Demo

1. Brief overview of the topic of our experiment
    a. Why we are doing it
    b. Value this adds to the community
    c. Overview of the experimental setup
2. Either
    a. Showing an LLM generating comments for a class-level code example in real-time with our code scripts

    b. Showing a comparison between LLM generated comments for a class-level code example and the ground-truth comments for the example and providing a brief analysis
        i. Getting the LLM to generate comments for a class-level code example beforehand

Delivery:

1. Create the dataset for the experiment from the dataset provided in the existing ClassEval benchmark(https://github.com/FudanSELab/ClassEval/blob/master/data/ClassEval_data.json) [1]
    a. New dataset will be a JSON file that represents a list of items
    b. Each item in the dataset will contain 3 fields: id, code, and annotated code
        i. id: id for the example
        ii. code: class–level code that contains code solutions for the methods
            1. Copied over from ClassEval benchmark [1]
        iii. annotated_code: class–level code that contains code solutions for the methods and comments for each method generated by us
            1. Created by us
2. Develop python code scripts that
    a. Take in the examples from the dataset
    b. Prompts the GPT-4 LLM for high quality comments for each example
        i. Prompt: k examples of <class-level code, class-level code annotated with comments> + target class-level code + “Annotate the target class-level code guided by the examples that are provided in the prompt”
        ii. We decided to use the OpenAI GPT-4 LLM as it’s pre-trained on a large dataset that includes both text and code and is very easy to use
            1. LLM Parameters:
                a. Temperature: 0.3, Top_p: 0.2
                    i. Ideal parameters for code commenting tasks per https://community.openai.com/t/cheat-sheet-mastering-temperature-and-top-p-in-chatgpt-api/172683
    c. Stores the outputs of the LLM-generated comments for each class-level code example in some file for evaluation later
3. Evaluate the comments generated by the LLMs
    a. Develop python code scripts that can generate the BLEU(described earlier) scores for each LLM-generated example
    b. Manually conduct Human Evaluation(described earlier) on each LLM-generated example

This is the Github repository we are using to save all of our code and data to.
https://github.com/hariharanep/LLMCodeCommenting

References

[1] Du, X. *et al.* (2024) ‘Evaluating large language models in class-level code generation’, *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, pp. 1–13. https://dl.acm.org/doi/10.1145/3597503.3639219