Analysis of automated code comment generation with zero-shot learning, in-context learning and prompt tuning on LLMs

1st Tanmay Vakare
Master's in Computer Science
University of Texas at Dallas
thv200000@utdallas.edu

2nd Harichandana Neralla
Master's in Computer Science
University of Texas at Dallas
hxn210036@utdallas.edu

***Abstract*—This project proposal aims to investigate and enhance the automated code comment generation process using Large Language Models (LLMs). Leveraging techniques such as zero-shot learning, in-context learning, and prompt tuning, the project seeks to improve the quality and relevance of generated code comments. Through comprehensive experimentation and evaluation, the research aims to provide valuable insights into the capabilities of LLMs in the context of software development, ultimately contributing to more efficient and effective code documentation practices.**

***Index Terms*—prompt tuning, zero-shot, comment generation, in-context**

I. Problem Statement

Automated code comment generation is a critical aspect of modern software development, aiming to enhance code understanding and maintainability. However, several challenges and issues persist in this domain, necessitating research and innovation:

- **Quality and Consistency**: Existing automated code comment generation tools often struggle to consistently produce high-quality comments. Comments may vary in clarity, relevance, and correctness, leading to inconsistencies in code documentation.
- **Code Complexity**: Complex code structures, nested functions, and intricate algorithms pose a significant challenge for automated comment generation. Current systems may fail to capture the intricacies of such code, resulting in inadequate or misleading comments.
- **Contextual Understanding**: Automated tools frequently lack the ability to fully comprehend the context in which the code operates. This limitation hampers their capacity to generate comments that accurately reflect the code's purpose and functionality.
- **Codebase Variability**: Software projects frequently involve multiple programming languages, frameworks, and libraries. Automated comment generation systems must adapt to diverse codebases, making it challenging to ensure universal applicability.
- **Informativeness vs. Conciseness**: Striking the right balance between providing informative comments and maintaining code readability is a persistent challenge. Automated systems often produce comments that are either overly verbose and complex or overly simplistic and uninformative.

Addressing these challenges is crucial to improving the effectiveness and adoption of automated code comment generation systems. Solutions that enhance code documentation while accommodating the complexities of real-world software development scenarios are essential for advancing this field.

A. Motivation

The rapid advancement of Large Language Models (LLMs) has transformed various natural language processing tasks. They are characterized by their massive scale and size, often containing hundreds of millions to billions of parameters (learnable weights). The scale of these models allows them to capture complex patterns and nuances in language. LLMs are typically pre-trained on large and diverse text corpora, such as the entire internet. While conventionally all surpervised deep learning(DL) based systems relied on large amount of data to learn patterns, currently significant research is being done in the direction of efficiently leveraging LLMs capabilities for unseen the downstream tasks.

In software development, generating accurate and informative code comments is crucial for enhancing code comprehensibility and maintainability. We propose a comprehensive analysis of automated code comment generation techniques leveraging the capabilities of LLMs, with zero-shot learning, in-context learning, and prompt tuning methodologies. The capabilities LLMs will enable it to capture complex patterns in the code and generate comprehensive comments. The methodologies mentioned before will help to efficiently leverage the LLMs for conditional generation, without requiring large amount of resources or data.

B. Objective

The primary objective of this research is to explore and assess the effectiveness of zero-shot learning in training LLMs to generate code comments without the need for explicit annotation or fine-tuning. Zero-shot learning will be applied

to enable LLMs to understand code context and generate meaningful comments based on surrounding code snippets.

In addition to zero-shot learning, in-context learning will be investigated to enhance the LLMs' capability to generate code comments that are contextually relevant and aligned with the programmer's intent. By incorporating contextual information from the surrounding code, and comment samples we aim to improve the accuracy and specificity of generated comments.

Prompt tuning [1], another crucial aspect of this project, involves designing effective prompts or learning prompts to elicit desired behaviors from LLMs. We will explore various prompt tuning strategies to fine-tune LLMs behaviour for code comment generation tasks, emphasizing the optimization of prompt design to achieve better results.

II. Related Work

Automated code comment generation has evolved significantly, driven by advancements in machine learning and natural language processing:

- **Supervised Learning**: Early efforts relied on supervised learning, and training models on annotated code-comment pairs, but faced limitations due to data availability [3].
- **Large Language Models (LLMs)**: LLMs like T5 and successors have transformed code comment generation. Fine-tuning LLMs has shown promise in generating contextually relevant comments [2] [4].
- **Contextual Understanding**: Recent studies incorporate code context to align comments with programmer intent, although striking a balance between context and conciseness remains a challenge.
- **Bias Mitigation**: Efforts are underway to mitigate biases in comment generation, ensuring fairness and unbiased comments.

Existing work underscores the potential for innovation in leveraging LLMs and related techniques to improve code comment quality, relevance, and context awareness [5], [7].

III. Technical Approach

The research methodology will involve extensive experimentation using diverse samples from coding languages like JAVA and Python, and LLM architectures, including but not limited to GPT, LLaMA their successors. Every model architecture would be tested on samples from the code base along with implementation of each methodology, zero-shot learning, prompt-tuning and in-context learning. Evaluation metrics will encompass comment relevance, clarity, conciseness, and informativeness. Additionally, the research will consider practical aspects such as the computational resources required and potential biases in generated comments.

IV. Expected Results

The outcomes of this project have the potential to significantly advance the field of automated code comment generation, benefiting developers by simplifying the documentation process and improving code readability. The results will also shed light on the capabilities and limitations of LLMs in understanding and generating code comments, ultimately contributing to the broader understanding of LLMs' capabilities in software development tasks.

V. Conclusion

In conclusion, this project proposal outlines a systematic investigation into the integration of zero-shot learning, in-context learning, and prompt engineering techniques to enhance the performance of LLMs in automated code comment generation. The findings from this research will provide valuable insights and methodologies that can be applied in real-world software development scenarios, ultimately leading to more efficient and effective code documentation practices.

References

[1] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Comput. Surv. 55, 9, Article 195 (September 2023), 35 pages. https://doi.org/10.1145/3560815

[2] Pearce, Hammond, Benjamin Tan, Baleegh Ahmad, Ramesh Karri, and Brendan Dolan-Gavitt. "Examining zero-shot vulnerability repair with large language models." In 2023 IEEE Symposium on Security and Privacy (SP). 2023.

[3] Iyer, Rajasekar, et al. "Summarizing Source Code using a Neural Attention Model." *Proceedings of the 34th International Conference on Machine Learning.* 2017.

[4] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22). Association for Computing Machinery, New York, NY, USA, Article 332, 1–7. https://doi.org/10.1145/3491101.3519665

[5] Svajlenko, Jeffrey, and Timothy Menzies. "When Is a Liability a Benefit? An Empirical Study of Java Program Commenting." *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering.* 2015.

[6] Hu, Heng, et al. "Deep code comment generation." *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers).* 2018.

[7] Hindle, Abram, et al. "On the naturalness of software." *2012 34th International Conference on Software Engineering (ICSE).* 2012.