Coding Smarter, Not Harder: Leveraging GPT-3.5 Turbo for Automated Code Solution Scoring

Aniket Deroy[1,*], Subhankar Maity[2,*]

[1]IIT Kharagpur, Kharagpur, India

Abstract

Information retrieval (IR) in software engineering is a critical area that focuses on the effective extraction, organization, and utilization of information from diverse software artifacts, including source code, documentation, and issue tracking systems. As software systems grow in complexity and volume, traditional search methodologies struggle to meet the demands of developers and engineers seeking relevant information. The task is, given a prompt (which includes a problem definition along with an incomplete code snippet) and ten corresponding solutions for each problem, we need to assign a predicted likelihood score to each (problem, solution) pair to indicate how likely the solution is to effectively address the problem. This paper explores GPT-3.5 Turbo via prompting to assign scores to solutions for programming tasks describing how good the solution is wrt to the problem. We experiment by differing the temperature values. Submission 2 scores the highest amongst the three submission runs with local nDCG of 0.6615 and Global nDCG of 0.9109.

Keywords

GPT, Software Engineering, Likelihood, Programming

1. Introduction

Information retrieval (IR) in software engineering has emerged as an essential discipline in addressing the challenges posed by the vast and complex landscape of software artifacts [1]. With the rapid growth of software systems, developers and engineers are inundated with an overwhelming amount of information, including source code, technical documentation, and issue tracking data [2]. The ability to effectively extract, organize, and utilize this information is paramount for enhancing productivity and fostering innovation in software development [3].

Traditional search methodologies often fall short in meeting the nuanced needs of software professionals, who require not just relevant information, but also contextual understanding and evaluative insights into potential solutions [4]. This gap in capability highlights the necessity for more sophisticated IR techniques that can provide a deeper analysis of programming tasks and solutions [5].

In this paper, we explore the application of GPT-3.5 Turbo [6] as a powerful tool for improving IR in software engineering. Specifically, we investigate how this advanced language model can be prompted to assign scores to various programming solutions based on their relevance and effectiveness concerning specific problems. By leveraging the model's natural language processing capabilities, we aim to enhance the retrieval process, enabling developers to quickly identify the most suitable solutions and streamline their workflow. This research not only contributes to the ongoing discourse on IR in software engineering but also proposes practical methodologies for integrating AI-driven insights into everyday software development practices. We experiment by differing the temperature values. Submission 2 scores the highest amongst the three submission runs with local nDCG of 0.6615 and Global nDCG of 0.9109.

Forum for Information Retrieval Evaluation, December 12-15, 2024, India

*Corresponding author.

roydanik18@kgpian.iitkgp.ac.in (A. Deroy); subhankar.ai@kgpian.iitkgp.ac.in (S. Maity)

0000-0001-7190-5040 (A. Deroy); 0009-0001-1358-9534 (S. Maity)

2. Related Work

The field of Information Retrieval (IR) in software engineering has gained significant traction as researchers and practitioners strive to manage the increasing complexity and volume of software artifacts [7]. Various studies have highlighted the challenges developers face in navigating vast repositories of source code, documentation, and issue tracking systems, underscoring the need for advanced IR techniques tailored to the unique characteristics of software engineering [8, 9, 10].

Traditional IR Techniques: Early approaches to IR in software engineering predominantly utilized keyword-based search methodologies, akin to traditional text retrieval systems [9, 11]. These methods, while straightforward, often fell short of addressing the nuanced queries posed by software developers. The limitations of Boolean searches and keyword matching have been well-documented, with studies showing that developers frequently struggle to find relevant information quickly, leading to decreased productivity and increased cognitive load [12]. As a response, researchers have advocated for context-aware retrieval systems that leverage semantic understanding, such as those incorporating domain-specific ontologies [13].

Semantic and Contextual IR: Recent advances in semantic search techniques have sought to bridge this gap by incorporating contextual information and understanding user intent [14]. For instance, techniques leveraging formal representations of code, such as abstract syntax trees (ASTs) and program dependence graphs, have been explored to enhance the retrieval process [15]. These methods facilitate more precise querying capabilities, allowing for improved matching of developer needs with relevant artifacts.

Natural Language Processing in Software Engineering: The intersection of natural language processing (NLP) and software engineering has garnered increasing attention, particularly with the advent of deep learning models [16]. Research has demonstrated that NLP can enhance code summarization, documentation generation, and even automated bug fixing [17, 18, 19, 20]. The application of transformer-based models, such as BERT and GPT, has shown promising results in understanding code semantics and providing contextualized retrieval solutions [21].

AI-Driven Insights for Enhanced IR: As AI technologies continue to evolve, researchers have begun to explore their potential in improving IR processes in software engineering [22]. Notably, studies have shown how machine learning algorithms can learn from past developer interactions to predict relevant solutions and provide personalized recommendations [23]. This trend aligns with our exploration of GPT-3.5 Turbo, where we investigate its ability to score programming solutions based on relevance and effectiveness, building on the foundational work of AI-assisted development environments.

Integrating AI in Development Workflows: The integration of AI-driven tools into software development practices represents a significant shift towards enhancing developer productivity [24]. Tools such as Codex and other AI pair programmers have demonstrated the potential for real-time assistance and contextual recommendations, addressing the challenges of information overload [25]. Our research seeks to expand on these capabilities by providing a structured approach to IR, enabling developers to leverage AI insights in their decision-making processes effectively.

In summary, while traditional IR methodologies have provided a foundation for information retrieval in software engineering, the rapid advancement of AI and NLP technologies presents an opportunity to redefine these approaches. By leveraging models like GPT-3.5 Turbo, this research aims to contribute to the evolution of IR techniques that not only retrieve relevant information but also offer actionable insights, ultimately fostering innovation and efficiency in software development practices.

3. Dataset

There are 164 queries in the test set along with 10 solutions corresponding to every query.

4. Task Definition

The task is, given a prompt (which includes a problem definition along with an incomplete code snippet) and ten corresponding solutions for each problem, we need to assign a predicted likelihood score to each (problem, solution) pair to indicate how likely the solution is to effectively address the problem.

5. Methodology

5.1. Why Prompting?

Prompting [26] is used for the following reasons:

- **Structured Context:** By providing a clear problem definition and an incomplete code snippet, prompting establishes a structured context that helps the model understand the specific requirements and nuances of the task at hand [27].
- **Guided Responses:** Prompts help guide the model in generating relevant and focused responses, ensuring that the solutions provided align closely with the problem described [28]. This helps mitigate ambiguity and enhances the quality of the output.
- **Efficiency:** Effective prompting can streamline the information retrieval process by enabling the model to quickly hone in on relevant solutions, reducing the time developers and engineers spend searching for pertinent information [29].
- **Assessment Framework:** Prompting sets up a framework for evaluating solutions based on the likelihood scores, making it easier to analyze how well each solution addresses the problem [30].
- **Complexity Management:** Given the increasing complexity of software systems, prompting allows the model to better handle diverse artifacts and contexts by providing specific cues that focus its attention on relevant aspects of the problem and potential solutions [31].
- **Scalability:** As software projects grow, the ability to use prompting to efficiently evaluate multiple solutions against specific problems scales well, accommodating the demands of larger teams and more complex systems [32].
- **Enhanced Learning:** Using prompts enables the model to draw on previous knowledge and learning from similar problems, allowing for more nuanced scoring and better-informed predictions [33].
- **Flexibility:** Prompts can be adapted to different types of programming tasks or domains, making the approach versatile and applicable to a wide range of software engineering challenges [34].

Overall, prompting enhances the effectiveness of information retrieval in software engineering by providing clarity, structure, and guidance, ultimately improving the relevance and accuracy of the solutions generated.

5.2. Prompt Engineering-Based Approach

We used the GPT-3.5 Turbo model via prompting to solve the code quality estimation problem in Zero-shot mode. We now summarize the internal steps in the prompting approach of GPT-3.5 Turbo:

(i) **Input Reception:** The model receives a text input (the prompt) from the user.
(ii) **Tokenization:** The input text is broken down into smaller units called tokens. This process involves converting words and punctuation into numerical representations that the model can understand.
(iii) **Context Encoding:** The model takes the sequence of tokens and encodes them into a contextual representation. This involves capturing the relationships and meanings between the tokens based on the model's training.

(iv) **Attention Mechanism:** Using an attention mechanism, the model weighs the importance of different tokens relative to each other. This allows it to focus on relevant parts of the input when generating a response.
(v) **Decoding:** The model generates a response by predicting the next token in the sequence, based on the encoded input and its learned patterns. It continues to generate tokens until it reaches a specified length or an end condition.
(vi) **Detokenization:** The generated tokens are converted back into human-readable text.
(vii) **Output Delivery:** The final text response is presented to the user.

An overview of GPT-3.5 Turbo to generate likelihood scores in Figure 1.

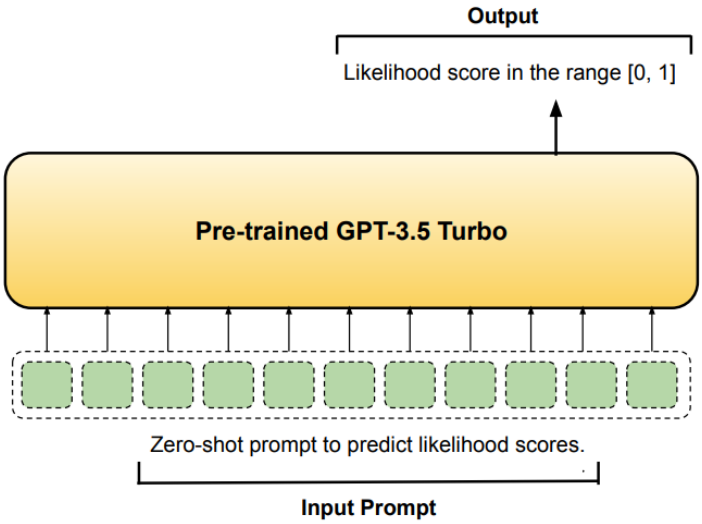

Figure 1: An Overview of GPT-3.5 Turbo for predicting likelihood scores.

We used GPT-3.5 Turbo in zero-shot mode at temperatures 0.7,0.8, and 0.9 with the following prompt: "*Given the problem <Problem> and the solution <Solution> generate a likelihood score between 0 and 1 stating how relevant is the solution wrt the problem. Only state the score*".

6. Results

Run	Local nDCG	Global nDCG
Submission 1	0.6595	0.9108
Submission 2	0.6615	0.9109
Submission 3	0.6602	0.9107

Table 1
Results for Task-2 of Information retrieval in software engineering Track

Table 1 shows the result for Task-2 of Information retrieval in software engineering Track. Results for Submission-2 of Information retrieval in software engineering Track is better than that of Submission-1 and Submission-3 for both local nDCG and Global nDCG.

7. Conclusion

This research highlights the critical role of advanced Information Retrieval (IR) techniques in software engineering, addressing the increasing complexity and volume of software artifacts developers face

daily. Traditional search methods often lack the contextual depth required by software professionals, underscoring the need for more refined IR solutions capable of delivering relevant, evaluative insights. Our study demonstrates the potential of GPT-3.5 Turbo as a valuable tool in this domain, showcasing how its language processing capabilities can be used to enhance the relevance and effectiveness of retrieved programming solutions.

By experimenting with different model parameters, such as temperature, we observed that Submission 2 consistently scored the highest, achieving a local nDCG of 0.6615 and a global nDCG of 0.9109. These results suggest that leveraging AI-driven models like GPT-3.5 Turbo not only improves retrieval accuracy but also supports software engineers in navigating complex development tasks more efficiently. This research contributes practical methods for integrating AI into software development workflows, enhancing productivity, and supporting innovation in the field.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Drafting content, Grammar and spelling check, etc. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] J. Cleland-Huang, O. C. Gotel, J. Huffman Hayes, P. Mäder, A. Zisman, Software traceability: trends and future directions, in: Future of software engineering proceedings, 2014, pp. 55–69.

[2] C. H. David, J. S. Famiglietti, Z.-L. Yang, F. Habets, D. R. Maidment, A decade of rapid—reflections on the development of an open source geoscience code, Earth and Space Science 3 (2016) 226–244.

[3] S. Nambisan, R. Agarwal, M. Tanniru, Organizational mechanisms for enhancing user innovation in information technology, MIS quarterly (1999) 365–395.

[4] M. L. Drury-Grogan, K. Conboy, T. Acton, Examining decision characteristics & challenges for agile software development, Journal of Systems and Software 131 (2017) 248–265.

[5] A. Sadeghi, H. Bagheri, J. Garcia, S. Malek, A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software, IEEE Transactions on Software Engineering 43 (2016) 492–530.

[6] B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, et al., Language models are few-shot learners, arXiv preprint arXiv:2005.14165 1 (2020).

[7] V. Garousi, M. Borg, M. Oivo, Practical relevance of software engineering research: synthesizing the community's voice, Empirical Software Engineering 25 (2020) 1687–1754.

[8] W. Scacchi, Understanding the requirements for developing open source software systems, IEE Proceedings-Software 149 (2002) 24–39.

[9] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, A systematic literature review on the usage of eye-tracking in software engineering, Information and Software Technology 67 (2015) 79–107.

[10] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology (2023).

[11] D. Binkley, D. Lawrie, P. Laplante, Applications of information retrieval to software development, Encyclopedia of Software Engineering (P. Laplante, ed.),(to appear) (2010).

[12] A. Abogdera, Exploring Information-Seeking Strategies College Students Use to Improve the Relevance of Retrieval from Online Information Retrieval Systems, Ph.D. thesis, Colorado Technical University, 2022.

[13] A. D. Dave, N. P. Desai, A comprehensive study of classification techniques for sarcasm detection on textual data, in: 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), IEEE, 2016, pp. 1985–1991.

[14] M. Fernández, I. Cantador, V. López, D. Vallet, P. Castells, E. Motta, Semantically enhanced information retrieval: An ontology-based approach, Journal of Web Semantics 9 (2011) 434–452.
[15] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu, A novel neural source code representation based on abstract syntax tree, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE, 2019, pp. 783–794.
[16] C. Watson, N. Cooper, D. N. Palacio, K. Moran, D. Poshyvanyk, A systematic literature review on the use of deep learning in software engineering research, ACM Transactions on Software Engineering and Methodology (TOSEM) 31 (2022) 1–58.
[17] S. Panichella, A. Panichella, M. Beller, A. Zaidman, H. C. Gall, The impact of test case summaries on bug fixing performance: An empirical investigation, in: Proceedings of the 38th international conference on software engineering, 2016, pp. 547–558.
[18] S. Gupta, S. Gupta, Natural language processing in mining unstructured data from software repositories: a review, Sādhanā 44 (2019) 244.
[19] Y. Zhu, M. Pan, Automatic code summarization: A systematic literature review, arXiv preprint arXiv:1909.04352 (2019).
[20] E. Dehaerne, B. Dey, S. Halder, S. De Gendt, W. Meert, Code generation using machine learning: A systematic review, Ieee Access 10 (2022) 82434–82455.
[21] D. Drain, C. Wu, A. Svyatkovskiy, N. Sundaresan, Generating bug-fixes using pretrained transformers, in: Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming, 2021, pp. 1–8.
[22] M. Borg, Advancing trace recovery evaluation-applied information retrieval in a software engineering context, arXiv preprint arXiv:1602.07633 (2016).
[23] Z. Batmaz, A. Yurekli, A. Bilge, C. Kaleli, A review on deep learning for recommender systems: challenges and remedies, Artificial Intelligence Review 52 (2019) 1–37.
[24] S. Tatineni, K. Allam, Ai-driven continuous feedback mechanisms in devops for proactive performance optimization and user experience enhancement in software development, Journal of AI in Healthcare and Medicine 4 (2024) 114–151.
[25] M.-F. Wong, S. Guo, C.-N. Hang, S.-W. Ho, C.-W. Tan, Natural language generation and understanding of big code for ai-assisted programming: A review, Entropy 25 (2023) 888.
[26] L. Wang, X. Chen, X. Deng, H. Wen, M. You, W. Liu, Q. Li, J. Li, Prompt engineering in consistency and reliability with the evidence-based guideline for llms, npj Digital Medicine 7 (2024) 41.
[27] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review, ACM Transactions on Software Engineering and Methodology (2023).
[28] E. A. Siverling, T. J. Moore, E. Suazo-Flores, C. A. Mathis, S. S. Guzey, What initiates evidence-based reasoning?: Situations that prompt students to support their design ideas and decisions, Journal of Engineering Education 110 (2021) 294–317.
[29] L. Belzner, T. Gabor, M. Wirsing, Large language model assisted software engineering: prospects, challenges, and a case study, in: International Conference on Bridging the Gap between AI and Reality, Springer, 2023, pp. 355–374.
[30] H. A. Diefes-Dux, J. S. Zawojewski, M. A. Hjalmarson, M. E. Cardella, A framework for analyzing feedback in a formative assessment system for mathematical modeling problems, Journal of Engineering Education 101 (2012) 375–406.
[31] B. Mirel, Interaction design for complex problem solving: Developing useful and usable software, Morgan Kaufmann, 2004.
[32] T. Stober, U. Hansmann, Best practices for large software development projects, Springer, 2010.
[33] L. Reynolds, K. McDonell, Prompt programming for large language models: Beyond the few-shot paradigm, in: Extended abstracts of the 2021 CHI conference on human factors in computing systems, 2021, pp. 1–7.
[34] A. Kleppe, Software language engineering: creating domain-specific languages using metamodels, Pearson Education, 2008.