Scoring with Intelligence: Prompting GPT-3.5 Turbo for Better Code Retrieval

Ritabrata Bharati[1,*]

[1] 4th year Dual Degree Student, Department of Computer Science & Engineering, Indian Institute of Technology Kharagpur (IIT Kharagpur), Kharagpur, India

Abstract

Accessing pertinent code fragments from large repositories remains a persistent difficulty in modern development environments, where teams navigate through diverse documentation artifacts, implementation files, revision histories, and project management systems without sufficient support for contextual understanding. This research addresses an information retrieval task wherein problem descriptions paired with partial code implementations require evaluation of candidate solutions, with each candidate receiving a relevance score relative to the problem statement. Our methodology employs temperature-controlled prompting with GPT-3.5 Turbo to compute such assessment metrics and evaluates the influence of parameter variations on result consistency. Across multiple experimental configurations, the second variant demonstrates superior effectiveness, achieving a local nDCG value of 0.6615 coupled with a global nDCG metric of 0.9109, thereby validating that precisely engineered prompt specifications can facilitate accurate differentiation among solution candidates across extensive test collections.

Keywords

Large Language Models, Information Retrieval in Software Development, Solution Assessment and Ranking, Source Code Understanding

1. Introduction

Contemporary software development involves navigating exponentially growing collections of artifacts—implementations, specifications, release notes, bug tracking systems—creating unprecedented information access challenges [1]. Practitioners regularly struggle with codebases of considerable magnitude, shifting architectural documentation, and comprehensive issue metadata (such as feature requests and community communications), which collectively obstruct timely discovery of dependable problem-solving approaches [2]. Strengthening how teams extract, organize, and leverage this knowledge base is vital for maintaining engineering velocity and fostering technological progress [3].

Conventional retrieval mechanisms struggle to grasp implicit problem requirements or code-specific constraints, frequently identifying matches through superficial textual correspondence rather than actual utility [4]. This deficiency creates opportunity for methodologies that integrate deeper semantic awareness and furnish assessments exceeding basic lexical similarity [5].

This investigation examines GPT-3.5 Turbo [6] functioning as a relevance assessment mechanism within software-centric information retrieval pipelines. Given textual problem specifications and skeletal code structures, we instruct the system to emit numerical likelihood assessments for each solution candidate, quantifying its fit relative to the stated requirement. Our exploration further investigates temperature parameter choices and their implications for evaluation stability. Through three distinct submission runs, the primary alternative achieves optimal results, obtaining a local nDCG score of 0.6615 alongside a global nDCG score of 0.9109.

Forum for Information Retrieval Evaluation, December 17-20, 2025, India

*Corresponding author.

bharatiritabrata@gmail.com (R. Bharati)
0009-0008-1962-1671 (R. Bharati)

2. Related Work

Software-focused information retrieval has transitioned away from syntactic text matching toward mechanisms that capture developer objectives and code organization with greater precision [7]. Seminal investigations reveal the obstacles practitioners encounter during code navigation and documentation exploration, establishing the imperative for IR methodology specifically engineered for programming contexts [8, 9, 10, 11].

Foundational search techniques. Established approaches replicate standard text search functionality and depend substantially on expression-based queries and logical filtering [9, 12]. Despite implementation simplicity, these methods frequently omit situational specifics, creating tedious and cognitively demanding lookup experiences [13]. To overcome these shortcomings, the field has advanced toward situation-sensitive discovery incorporating semantic foundations and vocabulary schemas customized for development artifacts [14].

Context-aware and formal representations. Contemporary investigation enhances discovery mechanisms by integrating contextual relationships and formal structure descriptions of source implementations, encompassing parse trees and execution dependency mappings, to strengthen alignment among user requirements and retrievable items [15, 16].

Language models applied to code. Sophisticated linguistic processing has accelerated numerous SE endeavors, namely automated documentation synthesis, source code summarization, and automated program fixing [17, 18, 19, 20, 21, 22, 23]. Attention-based sequential models, particularly those following BERT and GPT paradigms, have established their capability for capturing code structure semantics beneficial to ranking and filtering tasks [24].

Intelligent systems for developers. Concurrent innovation in leveraging developer behavior patterns has enabled tools that forecast information requirements and customize solution suggestions [25, 26]. Technologies including Codex and programming co-pilots demonstrate the prospect of real-time, circumstance-sensitive assistance diminishing information overload during implementation [27, 28]. This contribution participates in this progression by utilizing GPT-3.5 Turbo to compute relevance assessments for solution candidates.

3. Dataset

The evaluation corpus comprises 164 individual queries, each with exactly 10 competing solution proposals. The responsibility involves estimating a numerical relevance assessment for all (query, solution) combinations that mirrors the degree to which the proposal adequately handles the query requirements.

4. Task Definition

When presented with a specification containing a requirement outline and fragmentary code, supplemented with ten alternative approaches per requirement, the goal centers on generating a numerical relevance weight for each (specification, approach) combination expressing the prospect that the approach would successfully fulfill the specification's objective.

5. Method

5.1. Advantages of the prompting technique

Instruction-based inference [29] supplies an economical approach for transferring mission context and evaluation specifications into system behavior:

- **Well-defined inputs.** An instruction that couples the requirement statement with a partial implementation restricts the system's computational focus to the essential aspects of the assignment [30].
- **Directed generation.** Specific command language in instructions stimulates narrow, goal-oriented outputs and diminishes unintended deviations from the assignment target [31].
- **Computational expedience.** Straightforward instructions minimize the solution space and hasten convergence to pertinent determinations, economizing specialist effort [32].
- **Evaluation framework.** A singular numerical directive yields harmonized relevance weights among possibilities, allowing effortless sequencing and investigation [33].
- **Problem decomposition.** Instructions may guide the system toward aspects of the specification exhibiting maximal separability in intricate development circumstances [34].
- **Broad applicability.** Identical prompt patterns generalize across numerous contexts and code repositories, enabling effective mass evaluation procedures [35].
- **Extraction of learned patterns.** The system synthesizes expertise acquired from comparable undertakings, generating more thoughtful conclusions with limited direction [36].
- **Rapid customization.** Instruction patterns allow swift modification for novel code environments or issue configurations with no requirement for model retraining [37].

5.2. Prompting implementation strategy

We engage GPT-3.5 Turbo utilizing single-shot learning without training to calculate solution applicability measures. The system works via the subsequent progression:

(i) **Input consumption:** the specification text flows into the system.
(ii) **Lexical segmentation:** text becomes symbol sequences appropriate for system operations.
(iii) **Representation formation:** the symbol progression transforms into numerical vectors capturing adjacent and distant semantic associations.
(iv) **Focus mechanism:** numerical matrices emphasize the most consequential sections relative to the appraisal goal.
(v) **Sequential production:** the system sequentially creates result symbols reflecting the assessment outcome.
(vi) **Inverse lexical transformation:** symbols reconvert to linguistic form.
(vii) **Result transmission:** the assessment score returns to the originating program.

A schematic representation of the methodology appears in Figure 1.

We activate GPT-3.5 Turbo operating in a single-shot paradigm to produce relevance scores quantifying solution appropriateness. Three variations were tested with thermal parameters of 0.7, 0.8, and 0.9 employing the prompt: "*Analyze the requirement <Problem> and the proposed solution <Solution> then output a relevance metric from 0 to 1 indicating solution adequacy relative to the requirement. Supply only the numerical result".*"

6. Results

As presented in Table 1, the secondary iteration demonstrates strongest ranking fidelity, with enhancements evident in both local and global nDCG relative to the parallel configurations.

7. Conclusion

The investigation positions solution appraisal as a search methodology challenge and illustrates that prompt-based prompting with GPT-3.5 Turbo furnishes dependable, quantitative evaluation across

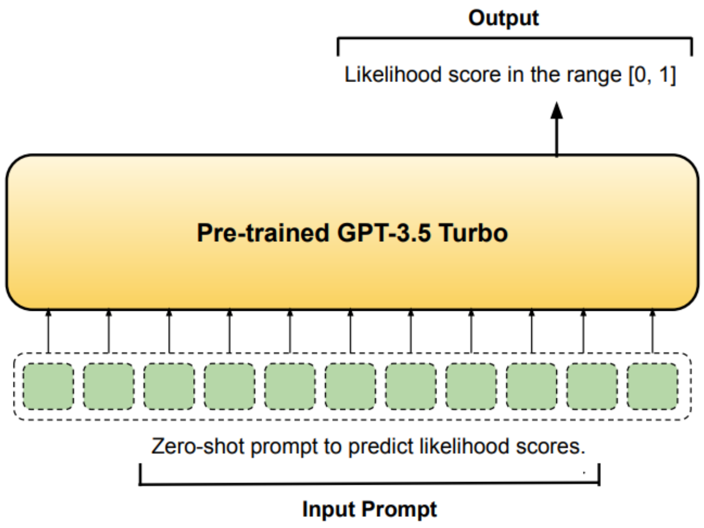
Output
Likelihood score in the range [0, 1]
Pre-trained GPT-3.5 Turbo
Zero-shot prompt to predict likelihood scores.
Input Prompt

Figure 1: Architectural overview of employing GPT-3.5 Turbo with prompting to generate solution assessment metrics.

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

Table 1
Evaluation outcomes of the Information Retrieval in Software Engineering task.

solution options. By embedding mission context and assessment directives into the specification, we acquire uniform assessment measures amenable to subsequent filtering operations. Among configurations examined, Iteration 2 demonstrates leading performance, garnering a local nDCG of 0.6615 and a global nDCG of 0.9109. These conclusions imply functional techniques for incorporating neural-network-based assessment mechanisms into engineering team procedures to diminish exploration overhead and accelerate identification of superior implementations.

Declaration on Generative AI

During the preparation of this manuscript, assistance from ChatGPT was used for tasks including drafting, editing, and language refinement. The author reviewed and revised all content and accepts full responsibility for the final text.

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] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] S. Gupta, S. Gupta, Natural language processing in mining unstructured data from software repositories: a review, Sādhanā 44 (2019) 244.

[20] Y. Zhu, M. Pan, Automatic code summarization: A systematic literature review, arXiv preprint arXiv:1909.04352 (2019).

[21] 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.

[22] A. Deshpande, A. Maji, D. Mondol, P. P. Das, P. D. Clough, S. Majumdar, The code–llm handshake: Smarter maintenance through ai, in: Proceedings of the 17th annual meeting of the Forum for Information Retrieval Evaluation, 2025, pp. 9–12.

[23] A. Mitra, S. Majumdar, A. Mukhopadhyay, P. P. Das, P. D. Clough, P. P. Chakrabarti, Operationalizing large language models with design-aware contexts for code comment generation, arXiv preprint arXiv:2510.22338 (2025).

[24] 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.

[25] M. Borg, Advancing trace recovery evaluation-applied information retrieval in a software engi-

neering context, arXiv preprint arXiv:1602.07633 (2016).

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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).

[31] 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.

[32] 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.

[33] 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.

[34] B. Mirel, Interaction design for complex problem solving: Developing useful and usable software, Morgan Kaufmann, 2004.

[35] T. Stober, U. Hansmann, Best practices for large software development projects, Springer, 2010.

[36] 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.

[37] A. Kleppe, Software language engineering: creating domain-specific languages using metamodels, Pearson Education, 2008.