Investigating the Impact of Synthetic Data on Code Comment Quality Prediction: A Logistic Regression Study

Subhajit Dutta[1,*]

[1]Indian Institute of Technology, Kharagpur, 721302

Abstract

The importance of code comment quality in software development can vary, which emphasizes the need for trustworthy evaluation methods. By combining manually annotated datasets with artificially generated data, this work aims to improve the classification of comment usefulness. We used GPT-4.1 to label extra comments in order to augment the data. The baseline classifier, a logistic regression model, had an F1 score of roughly 0.81 and showed little improvement when the synthetic data was added. The study looks into the benefits and drawbacks of using artificial data to assess the relevance of code comments.

Keywords

LLMs, GPT-4.1, Comment Classification, Logistic Regression, Qualitative Analysis, Data Augmentation

1. Introduction

Software is now essential in many vital industries, including banking, healthcare, and transportation, in the current digital landscape. Organizations must regularly update their current systems and create new applications in order to stay up with changing requirements. Code complexity inevitably rises as a result of this continuous evolution, which is necessary to support new features. Thus, a key component of the Software Development Life Cycle (SDLC) is efficiently managing these sizable and intricate codebases.

Quick fixes, code additions, and updates to already-existing applications are frequently the outcome of rapid development cycles. These hurried schedules, though, occasionally result in less-than-ideal coding techniques. Supporting documentation, such as requirement specifications and design documents, may become obsolete as software develops further, and the original developers are frequently unavailable for consultation. Program comprehension is a crucial tactic for preserving and enhancing current codebases, and this scenario emphasizes the significance of implementing organized, quality-focused processes in software development.

Because software is always changing, test execution logs, static code analysis, and code comments are all trustworthy sources of information. Code comments are emphasized in this study as important software design indicators for automated tools and developers alike. They help with understanding and upkeep by encapsulating the goals and reasoning behind the code. Their uneven quality, however, emphasizes the necessity for automated techniques to assess their value.

The lack of well-annotated datasets that represent a variety of programming contexts presents a significant challenge when assessing comment utility. In order to get around this, new methods are needed to increase the size of current datasets and improve model performance on actual data. In order to tackle this, our work combines synthetic data augmentation produced by GPT-4.1 with manual annotation.

The challenge of dividing source code comments in the C programming language into two groups—useful and not useful—is the focus of this paper. As the baseline classifier, we trained a logistic regression model on a dataset of 11,500 manually annotated comments. We added more than

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

*Corresponding author.

duttasubhajit050@gmail.com (S. Dutta)

0009-0004-6793-7639 (S. Dutta)

300 GPT-labeled samples to the dataset in order to investigate possible enhancements. With an F1 score of 0.81 on both the original and augmented datasets, the model's performance stayed consistent.

By combining synthetic data augmentation with manual annotations, our study improves our understanding of code comment utility classification. Our goal is to solve current issues and facilitate the creation of more flexible models for contemporary software engineering.

The structure of this paper is as follows: Section 2 reviews related work on comment classification. Section 3 describes the task and dataset, while Section 4 outlines the proposed methodology. Section 5 presents the results, and Section 6 provides the conclusion.

2. Background and Literature Review

Within the field of software engineering, automatic program comprehension is a well-established research topic. Using sources like runtime execution traces and structural code attributes, a number of tools have been developed to make it easier to extract knowledge from software metadata. [1, 2, 3, 4, 5, 6, 7, 8].

In order to comprehend the code flow, novice programmers usually rely on the comments that are already there. Not all comments, though, successfully aid in program comprehension, so it's important to consider their applicability before using them. Numerous scholars have investigated the automatic categorization of source code comments, emphasizing the assessment of their quality. For instance, Omal et al. [9] suggested a hierarchical structure for the variables affecting software maintainability. They made it possible to evaluate software features that can be combined into a single maintainability index by introducing quantifiable attributes for each factor in the form of metrics.Fluri et al. [10] looked into whether source code and related comments change together over time. They discovered that 97% of comment changes happened in the same revision as the corresponding code changes in three open-source projects: *ArgoUML, Azureus*, and *JDT Core*. The quality model was transformed into a structured knowledge base appropriate for industrial applications by Deissenboeck et al. [11], who proposed a two-dimensional maintainability model that explicitly links system properties with maintenance activities.An empirical study on task annotations embedded in source code was carried out by Storey et al. [12], with a focus on their function in task management for developers. They noted that task management entails striking a balance between the manual annotations developers add to their code and official issue-tracking systems. In order to improve the readability of PL/I programs, Tenny et al. [13] conducted a 3×2 experiment comparing procedure formatting with comments. Programs without comments were the least readable, according to the results of a survey given to student participants after they had read the program.

Yu et al. [14] categorized source code comments into four groups: unqualified, qualified, good, and excellent. They also showed that classification performance was enhanced by combining several basic classifiers. Majumdar et al. [15] presented CommentProbe, an automated classification system for assessing the quality of comments in C codebases.Even though there has been a lot of progress in examining source code comments from a variety of angles [15, 16, 17, 18, 19, 20, 21, 22, 8, 20, 23, 24, 19, 25, 26], automated quality evaluation of comments is still an important and developing field that needs more investigation.

It is crucial to compare the quality assessment of code comments by standard models such as GPT 4.1 or LLM with human interpretation in light of the emergence of large language models [27]. The IRSE track at FIRE 2025 [28, 22] expands on the approaches put forth in [15, 23, 29] to explore different vector space models [30] and features for binary classification and evaluation of comments in relation to code comprehension. Incorporating GPT-generated labels for code quality and comment snippets taken from open-source software, this track also evaluates the predictive model's performance.

#	Comment	Code	Label

1	/*Transform a minor status code
	(the underlying routine error) into text.*/
		-10. break;
		-9. }
		-8. gss_release_buffer(&min_stat, &status_string);
		-7. }
		-6. if(sizeof(buf) > len + 6) {
		-5. strcpy(buf + len, ".\n");
		-4. len += 2;
		-3. }
		-2. msg_ctx = 0;
		-1. while(!msg_ctx) {
		/*Transform a minor status code
		(the underlying routine error) into text.
			Useful

2	/*cr to cr,nul*/	-1. else
		/*cr to cr,nul*/
		1. newline = 0;
		2. }
		3. else {
		4. if(test->rcnt) {
		5. c = test->point[0];
		6. test->point++;
		7. test->rcnt−;
		8. }
		9. else
		10. break;
			Not Useful

3	/*test 637*/	-10. int res = 0; -9. CURL *curl = NULL; -8. FILE *hd_src = NULL; -7. int hd; -6. struct_stat file_info; -5. CURLM *m = NULL; -4. int working; -3. start_test_timing(); -2. if(!libtest_arg2) { -1. #ifdef LIB637 /*test 637*/ 1. fprin	Not Useful

Table 1
An example of data

3. Classification Goal and Data Characteristics

The task of developing a binary classification system to differentiate source code comments as either *useful* or *not useful* is the focus of this paper. After processing a comment and the code that goes with it, the system forecasts a label that accurately reflects the comment's significance. For this classification, conventional machine learning techniques like logistic regression can be used. The following is a definition of the comment categories:

- *Useful* - The comment offers precise or pertinent code information.
- *Not Useful* - The comment doesn't provide any useful details about the code.

11,500 C code-comment pairs make up our primary dataset. A team of 15 people annotated the pairs to indicate whether or not the comments were helpful. Additionally, by gathering code-comment pairs from GitHub and labeling them with GPT, we produced an extra dataset. Our primary dataset is supplemented by this secondary dataset, which is formatted identically to the original.

4. Model Implementation and Experimental Setup

The binary classification functionality is implemented using logistic regression. The system accepts surrounding code snippets and comments as input. We use a pre-trained Universal sentence encoder to generate embeddings of each code segment and the corresponding comment. Both machine learning models are trained using the output of the embedding process. 80% of the data instances and their labels are included in the training dataset. In both experiments, the remainder is used for testing. The next section discusses the model's description.

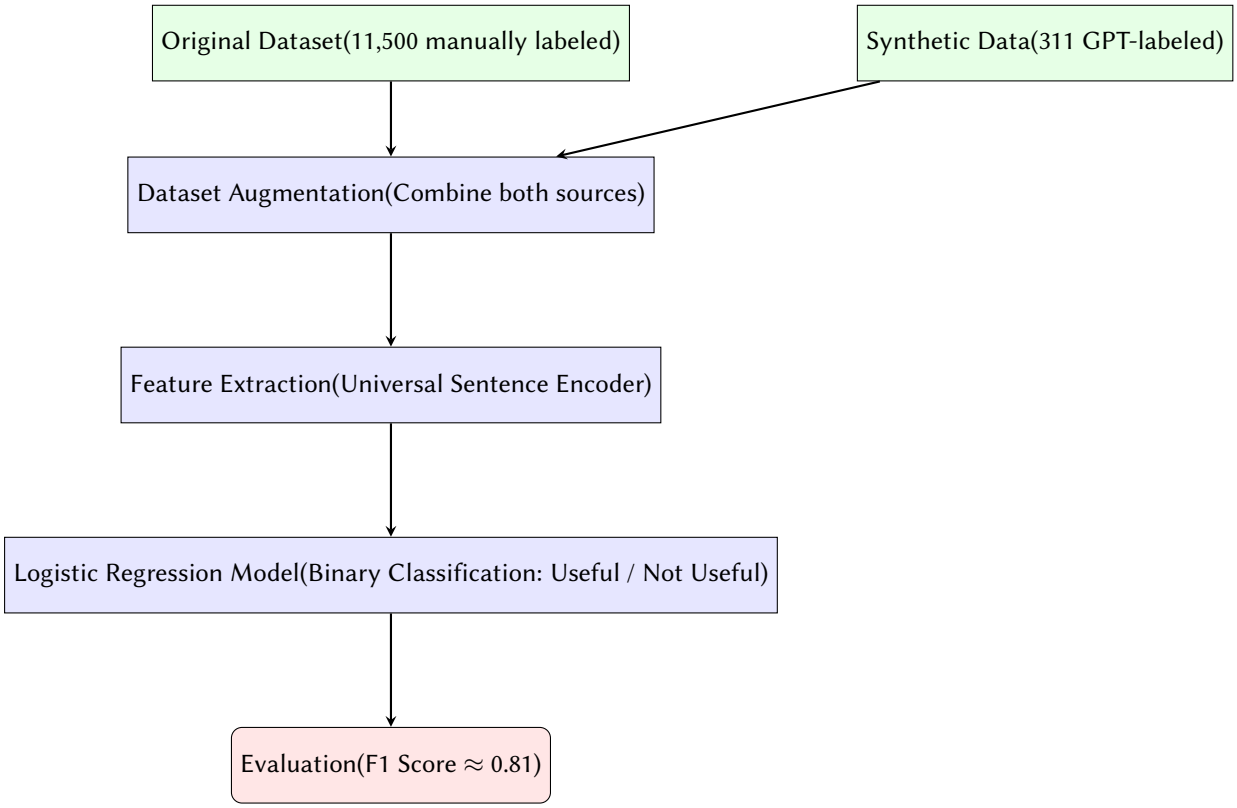

Figure 1: Methodology Block Diagram for Comment Quality Prediction

4.1. Logistic Regression

For the binary comment classification task, we employ logistic regression, which maintains the regression output between 0 and 1 by using a logistic function. The following is the definition of the logistic function:

$$Z = Ax + B \tag{1}$$

$$logistic(Z) = \frac{1}{1 + exp(-Z)} \tag{2}$$

The logistic function (see equation 2) receives the output of the linear regression equation (see equation 1). Based on the acceptance threshold, the logistic function's probability value is used to predict binary classes. For the *useful* comment class, we maintain the threshold value of 0.6. Every training instance yields a three-dimensional input feature that is fed into the regression function. The hyper-parameter tuning is trained using the Cross entropy loss function.

5. Classification Performance Metrics

We use both datasets to train our logistic regression model. There are 311 samples in the GPT-generated data compared to 11,500 samples in the original dataset. The first experiment yields the following scores using only the original data.

The following outcomes were observed after adding the GPT-generated data to the original dataset.

	Accuracy	Precision	Recall	F1 Score
Original Dataset	82.6374	0.80726	0.81938	0.81327
Augmented Dataset	81.7362	0.80184	0.82211	0.81185

Table 2
Binary classification outcomes for both datasets

The validity of using GPT-generated data for data augmentation is demonstrated by the very slight change in scores across metrics, which implies that the newly generated data was essentially indistinguishable from the original dataset.

6. Conclusion

This study looked into the classification of code comment quality using synthetic data. By combining pre-existing manually annotated data with synthetic samples, the main goal was to enhance the classification of comment utilities. For the binary classification task, the researchers used a baseline logistic regression model. An F1 score of approximately 0.81 was successfully established by this baseline model. One of the main conclusions of the study was that classification performance was not improved by the augmentation of synthetic data. Even after the augmentation, the model's results held steady, retaining the F1 score of 0.81, suggesting that the addition of the synthetic data had little effect. This result implies that in some situations, traditional annotated data remains highly effective. The study suggests investigating different strategies for future research, like employing more intricate models or integrating domain-specific expertise.

Declaration on Generative AI

In the course of preparing this manuscript, the author(s) employed the generative AI tool ChatGPT. Its use was limited to performing checks for grammar and spelling. Following this, the author(s) conducted a thorough review and revision of the text and assume full responsibility for the final published content.

References

[1] S. C. B. de Souza, N. Anquetil, K. M. de Oliveira, A study of the documentation essential to software maintenance, Conference on Design of communication, ACM, 2005, pp. 68–75.

[2] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Smartkt: a search framework to assist program comprehension using smart knowledge transfer, in: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2019, pp. 97–108.

[3] N. Chatterjee, S. Majumdar, S. R. Sahoo, P. P. Das, Debugging multi-threaded applications using pin-augmented gdb (pgdb), in: International conference on software engineering research and practice (SERP). Springer, 2015, pp. 109–115.

[4] S. Majumdar, N. Chatterjee, S. R. Sahoo, P. P. Das, D-cube: tool for dynamic design discovery from multi-threaded applications using pin, in: 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2016, pp. 25–32.

[5] S. Majumdar, N. Chatterjee, P. P. Das, A. Chakrabarti, A mathematical framework for design discovery from multi-threaded applications using neural sequence solvers, Innovations in Systems and Software Engineering 17 (2021) 289–307.
[6] S. Majumdar, N. Chatterjee, P. Pratim Das, A. Chakrabarti, Dcube_ nn d cube nn: Tool for dynamic design discovery from multi-threaded applications using neural sequence models, Advanced Computing and Systems for Security: Volume 14 (2021) 75–92.
[7] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, A. Brechmann, Measuring neural efficiency of program comprehension, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 140–150.
[8] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access 11 (2023) 73599–73612.
[9] P. Oman, J. Hagemeister, Metrics for assessing a software system's maintainability, in: Proceedings Conference on Software Maintenance 1992, IEEE Computer Society, 1992, pp. 337–338.
[10] B. Fluri, M. Wursch, H. C. Gall, Do code and comments co-evolve? on the relation between source code and comment changes, in: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE, 2007, pp. 70–79.
[11] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard, An activity-based quality model for maintainability, in: 2007 IEEE International Conference on Software Maintenance, IEEE, 2007, pp. 184–193.
[12] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, J. Singer, Todo or to bug, in: 2008 ACM/IEEE 30th International Conference on Software Engineering, IEEE, 2008, pp. 251–260.
[13] T. Tenny, Program readability: Procedures versus comments, IEEE Transactions on Software Engineering 14 (1988) 1271.
[14] H. Yu, B. Li, P. Wang, D. Jia, Y. Wang, Source code comments quality assessment method based on aggregation of classification algorithms, Journal of Computer Applications 36 (2016) 3448.
[15] S. Majumdar, A. Bansal, P. P. Das, P. D. Clough, K. Datta, S. K. Ghosh, Automated evaluation of comments to aid software maintenance, Journal of Software: Evolution and Process 34 (2022) e2463.
[16] S. Majumdar, S. Papdeja, P. P. Das, S. K. Ghosh, Comment-mine—a semantic search approach to program comprehension from code comments, in: Advanced Computing and Systems for Security, Springer, 2020, pp. 29–42.
[17] S. Majumdar, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Overview of the irse track at fire 2022: Information retrieval in software engineering., in: FIRE (Working Notes), 2022, pp. 1–9.
[18] S. Majumdar, A. Bandyopadhyay, P. P. Das, P. Clough, S. Chattopadhyay, P. Majumder, Can we predict useful comments in source codes?-analysis of findings from information retrieval in software engineering track@ fire 2022, in: Proceedings of the 14th Annual Meeting of the Forum for Information Retrieval Evaluation, 2022, pp. 15–17.
[19] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).
[20] P. Chakraborty, S. Dutta, D. K. Sanyal, S. Majumdar, P. P. Das, Bringing order to chaos: Conceptualizing a personal research knowledge graph for scientists., IEEE Data Eng. Bull. 46 (2023) 43–56.
[21] S. Majumdar, A. Deshpande, P. P. Das, P. P. Chakrabarti, Comprehending c codes with llms: Effective comment generation through retrieval and reasoning, Pattern Recognition Letters (2025).
[22] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D. Clough, et al., Overview of the "information retrieval in software engineering"(irse) track at forum for information retrieval 2024, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024, pp. 18–21.
[23] S. Paul, S. Majumdar, A. Bandyopadhyay, B. Dave, S. Chattopadhyay, P. Das, P. D. Clough, P. Majumder, Efficiency of large language models to scale up ground truth: Overview of the irse track at forum for information retrieval 2023, in: Proceedings of the 15th Annual Meeting of the Forum

for Information Retrieval Evaluation, 2023, pp. 16–18.
[24] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Tool assisted agile approach for legacy application migration, International Journal of System Assurance Engineering and Management (2025) 1–16.
[25] 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.
[26] 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).
[27] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.
[28] S. Paul, S. Majumdar, R. Shah, S. Das, M. Ghosh, D. Ganguly, G. Calikli, D. Sanyal, P. P. Das, P. D Clough, A. Bandyopadhyay, S. Chattopadhyay, Generative ai for code metadata quality assessment, in: Proceedings of the 16th Annual Meeting of the Forum for Information Retrieval Evaluation, 2024.
[29] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, S. Chattopadhyay, P. P. Das, P. D. Clough, P. Majumder, Generative ai for software metadata: Overview of the information retrieval in software engineering track at fire 2023, arXiv preprint arXiv:2311.03374 (2023).
[30] S. Majumdar, A. Varshney, P. P. Das, P. D. Clough, S. Chattopadhyay, An effective low-dimensional software code representation using bert and elmo, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2022, pp. 763–774.