Evaluating the Efficacy of Synthetic Data for Source Code Comment Classification

Lakshya Yadav[1,*]

[1]Indian Institute of Technology, Kharagpur, India, 721302

Abstract

A developer's ability to understand code is directly tied to the quality of its comments-a critical but often inconsistent aspect of software projects. We assessed the efficacy of synthetic data in the automated classification of source code comments by their utility. A foundational dataset, comprising human-labeled comments, was supplemented with examples generated by the GPT-4o language model. The primary objective was to measure any performance uplift in a baseline random forest classification model. The results demonstrated that the data augmentation strategy did not yield an improvement, as the model's F1 score was maintained at a consistent 0.78. This stability implies that the features present in the synthetic data were not sufficiently distinct or valuable for the classifier, underscoring the complexities of creating high-fidelity training data for software engineering tasks.

Keywords

Random Forests, Data Augmentation, Comment Classification, Qualitative Analysis

1. Introduction: The Challenge of Code Comprehension

The ever-growing pace of modern software development often leaves crucial documentation, like code comments, underdeveloped. As a codebase grows in size and new components are introduced, its documentation can become outdated, and the original authors may no longer be available to explain or even understand their work. This creates a significant challenge for software maintenance and evolution, highlighting a necessity for tools that can seamlessly automate the comprehension of the programs.

Code comments are one of the most vital and direct forms of code documentation, offering crucial insights into a programmer's intent, logic, and decisions behind the code. However, not all comments are helpful, ranging from highly insightful to completely irrelevant.

This inconsistency makes it essential to develop automated methods for judging the quality of comments. A primary roadblock to building such tools is the lack of high-quality, large enough datasets of annotated comments from diverse projects. This study addresses this gap by exploring the potential of synthetic data. We began with a manually annotated dataset of C language comments and augmented it with newly generated instances by GPT-4o. We then trained a random forest model to classify comments into one of two categories: "useful" or "not useful". Our findings show that the synthetic data did not noticeably alter the model's performance, raising important questions about how and when such data is truly effective.

By exploring the dynamic between manually annotated and AI-generated synthetic data for classifying comment usefulness, our work offers a clearer understanding of this interaction and provides new perspectives for subsequent inquiry. We explore a novel perspective on the interplay of synthetic data with manual data.

Following this introduction, we survey the relevant literature in Section 2. We then define our classification task and describe the datasets used in Section 3. Our methodology is detailed in Section 4, followed by a presentation of our results in Section 5. The paper concludes in Section 6 with a summary of our findings and thoughts on future work.

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

*Corresponding author.

lakshyayadav3009.ly@gmail.com (L. Yadav)

0009-0009-3051-4436 (L. Yadav)

2. Literature Review

Code comments are an essential aspect of software maintenance and understanding. Many tools [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] have been developed to extract and analyze information from source code [14], including runtime traces and structural features.

Previous studies [15, 16, 17, 18, 19, 20] have explored several methods for filtering and judging the effectiveness of source code comments. As an example, the work of Rahman et al. [21] distinguished the key features that make code review comments helpful or unhelpful, based on their findings on developer surveys conducted at Microsoft [22]. Recent advances in the field of large language models (LLMs) [23] have introduced new techniques for this task, with models being employed to evaluate comments and determine their relevance to the associated code [24, 25, 26, 27].

This study contributes to that ongoing conversation by focusing specifically on the impact of using synthetic data from GPT-4o to augment a human-annotated dataset for classification.

3. Classification Task and Data Sources

This paper outlines the development of a binary classification system for categorizing comments within the source code. The system's function is to accept a code comment and its related lines of code as input and to generate an output label designating the comment as either "useful" or "not useful". The purpose of this automated classification is to provide a tool that helps developers understand the source code effectively. We employ established machine learning algorithms, including random forests, to build the classifier. The categories of comments considered are the following.

- *Not Useful* - Content of the comment is not relevant to the linked code block.
- *Useful* - Content of the comment pertains directly to the associated code block.

This research utilizes a dataset composed of more than 11,000 pairs of code comments aligned with corresponding code snippets in the C language. Each entry includes the text of the comment, the associated code snippet, and a label denoting the perceived utility of the comment. Gathered from GitHub, this dataset was annotated by a team of 14 annotators. An example of this dataset entry can be found in Table 1.

To complement the primary dataset, we developed an analogous dataset for our research purposes. This supplementary dataset was constructed by extracting pairs of code and comments from GitHub repositories, and each pair was evaluated by GPT to determine its usefulness, classifying them as either not useful or useful. The design of this additional dataset closely mirrors that of the initial dataset, and its purpose is to augment the original dataset, enabling more in-depth analyses in subsequent stages of our investigation.

4. Experimental Design and Model

For the binary classification task, we utilize the Random Forest (RF) algorithm. It is an ensemble model chosen for its robustness against overfitting and its ability to handle complex feature interactions. The model processes both the comment text and the code, which are first converted into dense vector embeddings by using an off-the-shelf embedding model, the Universal Sentence Encoder.. The output embeddings are used to train the model on two datasets, original and augmented. Our experimental workflow is illustrated in Figure 1.

4.1. Random Forest Classifier

Binary classification is achieved through the Random Forest (RF) algorithm. The approach relies on an ensemble of decision trees, which simultaneously improves the model's predictive power and reduces overfitting. The core strategy of Random Forest is to grow a large number of individual decision trees

	Comment	Code	Label
1	/*test 516*/	-10. CURL *curl_handle = NULL; -9. int result = -1; -8. int file_descriptor; -7. struct_stat file_stats; -6. FILE *source_file = NULL; -5. int transfer_active; -4. CURLM *curl_multi_h = NULL; -3. init_test_timer(); -2. if(libtest_argc < 2) { -1. #ifdef TEST_CASE_516 /*test 516*/ 1. fwprintf	Not Useful

2	/* unpack the minor status code from the routine error into a text buffer for display*/	-10.
		-9. break;}
		-8. gss_release_buffer(&routine_status_code, &error_text_buffer);
		-7. }
		-6. if(sizeof(msg_buffer) > length + 4) {
		-5. strcat(buffer + length, ";\r\n");
		-4. length += 3;
		-3. }
		-2. message_context = NULL;
		-1. while(message_context != NULL) {
		/* unpack the minor status code from the routine error into a text buffer for display*/
			Useful

3	/*cur to cur,ptr*/	-1. else
		/*cur to cur,ptr*/
		1. new_line_char = '\0';
		2. }
		3. else {
		4. if(test_data->remaining_count > 0) {
		5. current_char = *(test_data->read_ptr);
		6. test_data->read_ptr++;
		7. test_data->remaining_count−;
		8. }
		9. else
		10. break;
			Not Useful

Table 1
Example data sample

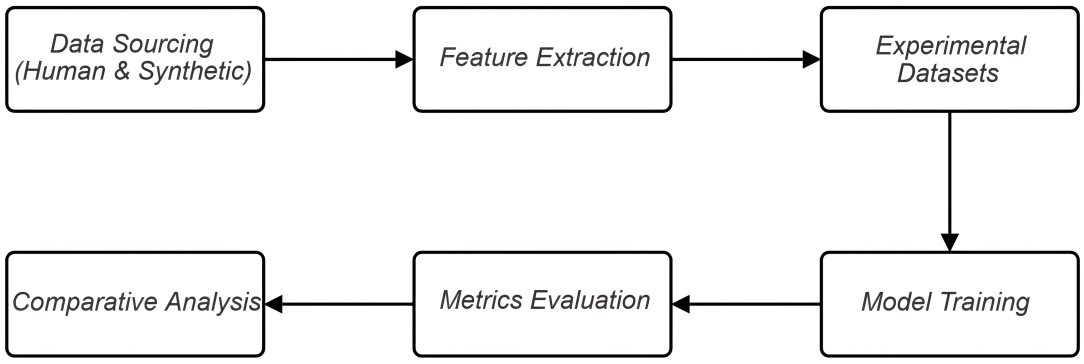

Figure 1: The overall workflow of the experimental design

during the training process. During inference, the model outputs the class that is the mode/mean or the most common prediction among all the individual trees.

The construction of each tree in the Random Forest follows these steps:

1. First, a unique training set for the tree is created by drawing a bootstrap sample from the original training data (sampling with replacement).
2. As the tree is constructed, at each node or decision point, only a random subset of features is considered for the split.
3. From that random subset, the algorithm determines the best possible split using a criterion like entropy for effective data partition or Gini impurity.
4. These last two steps are then repeated recursively for each new node until the tree is complete.

The model's final prediction is determined by aggregating the outputs from every tree in the forest and conducting a majority vote, formally expressed as:

$$RF(x) = \text{majority}\left(\{T_i(x)\}_{i=1}^{N}\right) \tag{1}$$

where $T_i(x)$ represents the output generated by the i-th decision tree for x, the input vector, and N denotes the total count of decision trees in the ensemble. By default, a standard probability threshold of 0.5 is applied to make the final decision; however, this threshold can be adjusted to make the model more sensitive to a particular class, such as the "useful" comment category.

One of the significant practical advantages of Random Forest is its innate ability to handle high-dimensional feature spaces without requiring prior feature scaling. Furthermore, the algorithm is robust in the presence of missing data. It addresses this issue by either making decisions based on the non-missing values or by imputing the missing entries with a calculated substitute, based on mean/mode, or the majority class.

During training, the model calculates an Out-of-Bag (OOB) error using the data not included in each tree's bootstrap sample. The OOB error serves a dual purpose: it offers an impartial measure of the model's ability to generalize to new data and provides a mechanism for tuning hyperparameters without needing a separate validation dataset.

5. Results

To evaluate the model's performance, two experiments using a Random Forest classifier were conducted. The first involved training the model exclusively on the original dataset, which contains 11,452 samples. For the second experiment, we created an augmented dataset by adding 233 synthetically generated samples from GPT to the original dataset.

The dataset augmented with GPT-curated data yielded the following performance metrics:

	Accuracy	Precision	F1 Score	Recall
Original Dataset	0.8201	0.7850	0.7881	0.7912
Augmented Dataset	0.8198	0.7801	0.7895	0.7992

Table 2
Comparison of Model Performance on Original and Augmented Datasets

The performance metrics showed minimal deviation between the two experiments, suggesting that the GPT-generated synthetic data was comparable in quality to the original, manually annotated data. This stability highlights the viability of synthetic data augmentation as a method for expanding datasets without degrading model performance.

6. Conclusion

In this paper, we developed a binary classification system that uses a random forest model to determine the usefulness of source code comments in C programs. The key experimental finding is that synthetic

data from GPT-4o performs on par with manually annotated data. This result highlights the value of synthetic data augmentation for expanding machine learning datasets, particularly when resources for manual annotation are scarce.

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. 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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).

[8] 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.

[9] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access 11 (2023) 73599–73612.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.
[15] L. Tan, D. Yuan, Y. Zhou, Hotcomments: how to make program comments more useful?, in: Conference on Programming language design and implementation (SIGPLAN), ACM, 2007, pp. 20–27.
[16] Y. Wang, H. Le, A. D. Gotmare, N. D. Bui, J. Li, S. C. Hoi, Codet5+: Open code large language models for code understanding and generation, arXiv preprint arXiv:2305.07922 (2023).
[17] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.
[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, 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: Forum for Information Retrieval Evaluation, ACM, 2022.
[20] J. L. Freitas, D. da Cruz, P. R. Henriques, A comment analysis approach for program comprehension, Annual Software Engineering Workshop (SEW), IEEE, 2012, pp. 11–20.
[21] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 215–226.
[22] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, Working Conference on Mining Software Repositories, IEEE, 2015, pp. 146–156.
[23] 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.
[24] 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.
[25] S. Majumdar, S. Paul, D. Paul, A. Bandyopadhyay, B. Dave, 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, in: Forum for Information Retrieval Evaluation, ACM, 2023.
[26] 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.
[27] 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).