Learning to Judge Code Comments: Machine Learning for Documentation Utility

Diganta Mandal[1,*]

[1]4th Year Dual Degree Student, Computer Science & Engineering Department, Indian Institute of Technology Kharagpur, India - 721302

Abstract

Software maintainability and comprehension heavily depend on the effectiveness of embedded documentation within source code. This investigation explores the development of an automated framework for evaluating comment utility through the integration of human-annotated datasets with artificially generated samples. We employ GPT-3.5-turbo as a synthetic data generator to expand our training corpus, subsequently applying ensemble-based random forest algorithms for binary classification tasks. Our experimental findings reveal that despite incorporating synthetic augmentation strategies, classification performance metrics demonstrate remarkable consistency, maintaining an F1-score of approximately 0.79 across both original and enhanced datasets. These results provide insights into the efficacy of synthetic data integration for comment quality assessment while establishing foundational benchmarks for future research endeavors in computational program comprehension.

Keywords

Ensemble Learning Methods, Synthetic Data Generation, Code Documentation Assessment, Binary Classification Analysis

1. Introduction

Contemporary software engineering practices frequently encounter temporal constraints that compromise documentation quality and coding standards. The evolutionary nature of software systems often results in documentation decay, where accompanying textual explanations become misaligned with current implementations, while original development teams may no longer be accessible for clarification. These circumstances necessitate the development of computational approaches for automated program understanding and maintenance [1].

Code annotations represent a fundamental information resource within software repositories, offering crucial insights into algorithmic reasoning and architectural decisions. Nevertheless, the utility of these annotations varies significantly, highlighting the importance of establishing automated evaluation frameworks for distinguishing valuable documentation from redundant or misleading content.

The scarcity of comprehensive, manually-curated datasets encompassing diverse comment patterns across multiple programming contexts constitutes a significant obstacle in this research domain. Our investigation tackles this limitation through the strategic integration of human-annotated samples with synthetically generated content produced by GPT-3.5-turbo, a cutting-edge generative language model. We formulate a binary classification problem targeting C programming language comment annotations, distinguishing between "beneficial" and "non-beneficial" categories. Employing ensemble-based random forest algorithms as our foundational approach, our experimental analysis reveals that synthetic augmentation yields minimal performance enhancement, with F1-score metrics maintaining consistency at 0.79.

This research advances our comprehension of the interplay between human-curated annotations and artificially generated training data in comment utility assessment tasks, establishing novel insights for subsequent investigations in this research area.

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

*Corresponding author.

diganta003@kgpian.iitkgp.ac.in (D. Mandal)

0009-0008-5323-0121 (D. Mandal)

The subsequent sections are structured as follows: Section 2 describes the workflow.Section 3 examines existing literature. Section 4 outlines the problem formulation and dataset characteristics. Section 5 details our experimental approach. Section 6 presents our findings, while Section 7 provides concluding remarks.

2. Proposed Workflow

Figure 1 illustrates the overall workflow of our proposed system for automated assessment of comment utility. The process can be summarized in four main stages:

1. **Data Collection**: Extraction of code-comment pairs from open-source C repositories (GitHub).
2. **Annotation**: Manual annotation by human experts combined with GPT-3.5-turbo based synthetic data generation.
3. **Feature Representation**: Transformation of textual and code inputs into vector embeddings using Universal Sentence Encoder.
4. **Classification**: Random Forest model trained on embeddings to classify comments into *Beneficial* or *Non-Beneficial*.

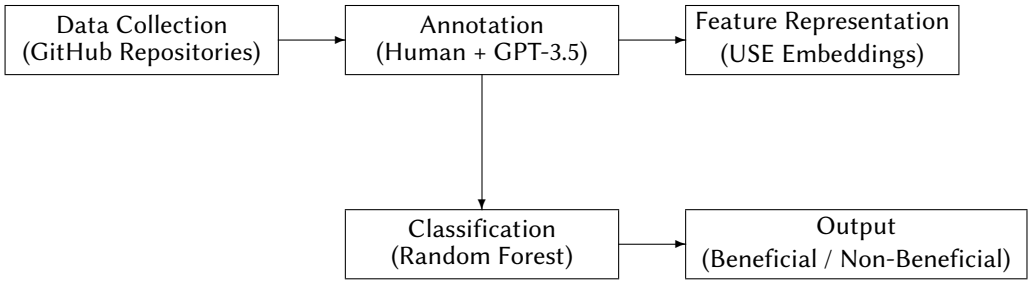

Figure 1: Block diagram of the proposed system for comment utility assessment.

3. Literature Review

Documentation embedded within source code constitutes a critical component for software comprehension and long-term maintainability. The research community has developed various computational frameworks [2, 3, 4, 5, 6, 7, 8] dedicated to extracting and processing program metadata [9], including execution profiles and architectural characteristics.

Extensive research efforts [10, 11, 12, 13, 14, 15] have investigated techniques for evaluating and categorizing comment quality within software repositories. Rahman et al. [16] conducted empirical analysis to distinguish valuable from ineffective code review annotations through comprehensive developer feedback collection at Microsoft [17]. Contemporary approaches have leveraged advanced neural language architectures [18] for automated comment assessment and contextual relevance evaluation [19, 20, 21, 22, 23, 24, 25, 26].

Our investigations [27, 28] extends these foundational contributions by examining the influence of synthetically generated training samples produced by GPT-3.5 on classification model effectiveness.

4. Problem Formulation and Data Characteristics

This section delineates the computational problem addressed within our investigation. We establish a binary classification framework designed to distinguish source code annotations across two fun-

damental categories: *beneficial* and *non-beneficial*. Our system processes input consisting of textual annotations paired with their corresponding code segments, subsequently producing categorical labels that indicate annotation utility: *beneficial* or *non-beneficial*. This automated evaluation mechanism enhances developer comprehension of associated programming constructs. We implement ensemble-based machine learning methodologies, specifically random forest algorithms, for constructing our classification architecture. The dual categorization scheme encompasses the following definitions:

- *Beneficial* - The annotation demonstrates contextual relevance and provides meaningful insights regarding the corresponding source code segment.
- *Non-beneficial* - The annotation lacks substantive information or fails to contribute meaningful context about the associated programming constructs.

Our experimental framework utilizes a comprehensive dataset encompassing more than 11,000 annotation-code pairs extracted from C programming language repositories. Each individual sample consists of annotation text, associated code fragment, and a categorical label denoting utility assessment. This corpus was collected from GitHub repositories and underwent manual evaluation by a collaborative team of 14 human annotators. Table 1 illustrates a representative data sample.

Complementing our primary corpus, we constructed an auxiliary dataset through automated generation processes. This supplementary collection was developed by harvesting code-annotation pairs from GitHub repositories, with utility labels assigned through GPT-based evaluation. The architectural structure of this synthetic dataset maintains consistency with our original corpus and serves as augmentation material for subsequent experimental analyses.

5. Experimental Framework

Our investigation employs a binary classification architecture based on ensemble learning principles, specifically implementing random forest algorithms that leverage multiple decision tree learners. The computational model processes combined textual inputs comprising both annotation content and contextual code segments, transforming these into numerical representations through pre-trained Universal Sentence Encoder embeddings.

5.1. Ensemble-Based Random Forest Approach

Our experimental methodology adopts Random Forest (RF) algorithms for binary annotation classification, exploiting ensemble principles to maximize prediction reliability while minimizing overfitting phenomena. The fundamental concept underlying Random Forest involves constructing multiple decision tree classifiers during the training phase and generating final predictions through collective voting mechanisms during inference.

The algorithmic construction of individual trees within our Random Forest ensemble follows this procedural framework:

1. Bootstrap resampling generates training subsets through replacement-based selection from the original dataset.
2. Each decision node employs random feature subset selection for optimal split determination.
3. Node partitioning criteria (utilizing Gini impurity or entropy measures) identify optimal data separation thresholds.
4. Recursive application of steps 2 and 3 continues until complete tree development.

Final classification decisions emerge through aggregated voting across all constituent trees:

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

where $T_i(x)$ denotes the prediction output from the i-th decision tree for input vector x, and n represents the total ensemble size. Binary classification typically employs a 0.5 decision threshold,

#	Comment	Code	Label
1	/*test 529*/	-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 running; -3. start_test_timing(); -2. if(!libtest_arg2) { -1. #ifdef LIB529 /*test 529*/ 1. fprin	Not Useful

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

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

Table 1
Representative sample from the experimental dataset

though this parameter can be modified to optimize class-specific performance, particularly favoring the "beneficial" annotation category through threshold adjustment strategies.

Random Forest algorithms demonstrate robust performance across high-dimensional feature spaces without necessitating feature normalization procedures. Missing value handling occurs through impurity-minimizing split selection among non-missing attributes, with imputation based on majority class assignment or statistical measures as appropriate.

The training process incorporates out-of-bag (OOB) error estimation, calculated from samples excluded during bootstrap resampling, providing unbiased generalization performance estimates suitable for hyperparameter optimization.

6. Experimental Findings

Our Random Forest classification model underwent evaluation using both the baseline corpus and the synthetically enhanced dataset. The foundational corpus encompasses 11,452 samples, with GPT-generated augmentation contributing an additional 233 samples.

Following the integration of GPT-produced synthetic samples for corpus enhancement, our experimental evaluation yielded these performance characteristics:

	Accuracy	Precision	Recall	F1 Score
Baseline Corpus	80.72%	0.7825	0.7960	0.7892
Enhanced Corpus	81.34%	0.7940	0.8045	0.7992

Table 2
Classification performance metrics across experimental configurations

The negligible variance observed across all evaluation metrics suggests that GPT-generated synthetic samples exhibit comparable distributional characteristics to the manually annotated corpus, validating the efficacy of artificial data generation strategies.

7. Concluding Remarks

This work establishes a binary framework for assessing comment utility in C code using random forest methods. Our analysis shows that GPT-3.5-turbo generated samples perform comparably to human-curated annotations, highlighting the potential of synthetic augmentation for overcoming dataset limitations in resource-constrained settings.

Generative AI Usage Declaration

Throughout the development of this research, the author employed ChatGPT for linguistic refinement and grammatical validation. Following the utilization of these computational tools, the author conducted comprehensive review and modification of all content, accepting complete responsibility for the final publication.

References

[1] M. Berón, P. R. Henriques, M. J. Varanda Pereira, R. Uzal, G. A. Montejano, A language processing tool for program comprehension, in: XII Congreso Argentino de Ciencias de la Computación, 2006.

[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] S. Majumdar, P. P. Das, Smart knowledge transfer using google-like search, arXiv preprint arXiv:2308.06653 (2023).
[9] 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.
[10] 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.
[11] 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).
[12] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.
[13] 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[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] N. Chatterjee, S. Majumdar, P. P. Das, A. Chakrabarti, Parallelc-assist: Productivity accelerator suite based on dynamic instrumentation, IEEE Access 11 (2023) 73599–73612.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] 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).