Enhancing Code Comment Classification Using Language Models

Jaivin Barot

Kadi Sarva Vishwavidyalaya, LDRP Campus, Sector-15, KH-5, Gandhinagar-382015

Abstract
In the realm of software development, collaboration among development teams is vital, and comments play a pivotal role in maintaining and improving software quality. Comments serve diverse purposes, from elucidating complex code logic to aiding in debugging and offering insights into design decisions. However, distinguishing between valuable and redundant comments can be a formidable challenge. This paper explores the potential of Language Model-based (LLM) approaches, particularly advanced models such as GPT-3, to automate the classification of comments and evaluate their effectiveness. By harnessing the contextual comprehension and generation capabilities of these models, this research postulates significant advancements in comment analysis. Through extensive experiments utilizing both real-world human-labeled comments and synthetic comments generated by ChatGPT, we demonstrate that LLMs can classify comments with remarkable accuracy, surpassing previous methods that rely on surface-level features. Additionally, this study critically examines factors like pre-training data, comment coverage, and model architectures, shedding light on their impact on comment analysis. In summary, this research makes several substantial contributions. It thoroughly explores the application of cutting-edge LLMs for comment classification across various contexts, provides a benchmark dataset of human-annotated comments, and highlights that LLMs can greatly enhance codebase documentation by automatically identifying low-quality comments. These techniques hold the potential for integration into Integrated Development Environments (IDEs) to provide developers with continuous feedback. Finally, this paper opens up new possibilities for leveraging advanced Natural Language Processing (NLP) in software engineering tasks that require deep code comprehension, despite lingering questions about model robustness and the nature of human-AI collaboration. This work underscores the enormous potential of LLMs in revolutionizing programming by mastering language understanding and generation in the context of software development.

CEUR-WS.org/Vol-3681/T7-10.pdf

1. Introduction

In the landscape of modern software development, where large, collaborative teams construct intricate codebases, code comments serve as a lifeline for developers. Well-crafted comments offer insights into the reasoning behind the code, complex implementations, edge cases, and issues that may elude static analysis. This documentation proves invaluable for ongoing maintenance, facilitating the onboarding of new team members, debugging, and preserving institutional knowledge. While numerous studies have emphasized the benefits of comments in software development, sheer quantity alone does not guarantee improved code quality or comprehension.

[*] Forum for Information Retrieval Evaluation, December 9-13, 2022, India
[*] Corresponding author.
barotjaivin244@gmail.com (J. Barot)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

The real challenge lies in differentiating between helpful comments and those that obfuscate understanding. Low-quality comments that merely restate the code, delve into trivial implementation details, or contain outdated information can introduce confusion and noise. The identification and management of comment quality pose significant challenges, particularly in large-scale projects. This paper delves into the possibility of automating the classification of code comments as either useful or non-useful using Language Model-based (LLM) approaches, such as GPT-3. These models excel in understanding and generating language, making them ideal candidates for assessing comments. Our research includes experiments involving two sources of comments: human-curated comments from real-world code and synthetic comments generated by the LLM ChatGPT. Through rigorous analysis and comparisons, our study demonstrates that LLMs can achieve accuracy rates exceeding 90% in comment classification, outperforming previous methods reliant on surface-level code features. These techniques hold the potential for seamless integration into developer workflows, aiding in the identification of unhelpful comments, enhancing documentation practices, and ultimately improving the maintainability of codebases in the long term.

1.1. The Role of Comments in Software Comprehension

Before delving into automatic comment classification, we first review the established literature regarding the role of comments in program comprehension. Effective documentation is widely acknowledged as crucial in software development and maintenance. Code tells you "what," while comments tell you "why." Comments clarify reasoning, capture design decisions, elucidate complex sections, and prevent the loss of critical knowledge over time. Several seminal studies have empirically demonstrated the advantages of high-quality comments. Tenny (1988) found that comments significantly improved code understanding, even more so than identifier names [1]. Woodfield et al. (1981) reported similar findings, indicating that comments enhanced modification tasks conducted by experienced programmers [2]. Further research has reinforced these findings, demonstrating improvements in comprehension [3-5]. Nevertheless, the quantity of comments does not necessarily correlate with quality or usefulness. The addition of unhelpful comments introduces unnecessary documentation overhead. Lawrie et al. (2007) discovered that approximately 20% of comments provided no additional meaningful information beyond identifiers [6]. Steidl et al. (2013) manually analyzed over 4,500 comments and determined that 28% offered negligible value [7]. Time spent creating and maintaining such ineffective comments is a waste of developer resources. Differentiating useful explanations from uninformative or unnecessary ones poses a significant challenge. Manual inspection does not scale, and simply quantifying comment length or counting keywords ignores semantic content. Automated techniques are needed to assess comment utility effectively, separating valuable insights from the noise.

1.2. Applications of Language Models in Software Engineering

Recent advancements in Natural Language Processing (NLP) have yielded powerful Language Models (LMs) with remarkable capabilities. Models like GPT-3 exhibit proficiency in understanding, generating, and reasoning about natural language. While primarily designed for conversational tasks, LMs also demonstrate strengths in dealing with programming languages. Multiple studies have explored the potential applications of LMs in software engineering, including code search and retrieval, automated documentation generation, code summarization, bug detection, security vulnerability identification, and improved code completion. These applications underscore the potential of LMs to assist developers by leveraging their substantial knowledge of code and mastery of natural language for explaining it. We hypothesize that LMs can significantly enhance the analysis of code comments, given their strengths in both language and code understanding. Surprisingly, prior to our research, no comprehensive study had examined LMs for classifying comment utility. Our research conducts extensive experiments using powerful LMs like GPT-3, applied to both real and synthetic comments on a large scale.

2. Related Work

Software metadata [1] plays a crucial role in the maintenance of code and its subsequent understanding. Numerous tools have been developed to assist in extracting knowledge from software metadata, which includes runtime traces and structural attributes of code [2, 3, 4, 5, 6, 7, 8, 9, 10].

In the realm of mining code comments and assessing their quality, several authors have conducted research. Steidl et al. [11] employ techniques such as Levenshtein distance and comment length to gauge the similarity of words in code-comment pairs, effectively filtering out trivial and non-informative comments. Rahman et al. [12] focus on distinguishing useful from non-useful code review comments within review portals, drawing insights from attributes identified in a survey conducted with Microsoft developers [13]. Majumdar et al. [14, 15, 16, 17] have introduced a framework for evaluating comments based on concepts crucial for code comprehension. Their approach involves the development of textual and code correlation features, utilizing a knowledge graph to semantically interpret the information within comments. These approaches employ both semantic and structural features to address the prediction problem of distinguishing useful from non-useful comments, ultimately contributing to the process of decluttering codebases

In light of the emergence of large language models, such as GPT-3.5 or llama [18], it becomes crucial to assess the quality of code comments and compare them to human interpretation. The IRSE track at FIRE 2023 [19] expands upon the approach presented in a prior work [14]. It delves into the exploration of various vector space models [20] and features for binary classification and evaluation of comments, specifically in the context of their role in comprehending code. Furthermore, this track conducts a comparative analysis of the prediction model's performance when GPT-generated labels for code and comment quality, extracted from open-source software, are included.

2.1. Rule-based Methods

Early approaches relied on manually defined rules and heuristics to identify unhelpful comments. For instance, Ratzinger et al. (2007) specified rules such as overly lengthy comments, the presence of code tokens, or excessively short comments [21]. Tan et al. (2007) designed 197 regex patterns to match non-informative phrases [22]. de Souza et al. (2005) also defined rules based on comment length, special characters, and keywords [23]. These rule-based systems required extensive input from domain experts and suffered from limited generalizability across different contexts and difficulties in handling semantic variations. In contrast, our LM-based approach overcomes these challenges through automated inductive capabilities and contextual understanding.

2.2. Feature Engineering with Classifiers

More recent efforts have focused on extracting linguistic features to train traditional machine learning classifiers. Steidl et al. (2013) computed lexical features such as comment length, terms used, readability, and punctuation to train Support Vector Machines (SVMs) [7]. Dat

apathaa and Nicholson (2018) combined word embeddings and grammar complexity metrics as inputs for regressors and forests [24]. The performance of these methods heavily relies on the crafting of features and is limited by the representational power of manually designed features. In contrast, our techniques leverage deep contextual embeddings within LMs that capture semantic relationships.

2.3. Neural Models

Several studies have explored neural networks for comment analysis, albeit with key differences from our approach. For instance, Hu et al. (2018) employed Long Short-Term Memory (LSTM) networks on sequential comment text for classification [25]. Jiang et al. (2017) combined Recurrent Neural Networks (RNNs) for text with Convolutional Neural Networks (CNNs) for source code [26]. While promising, these models do not benefit from the extensive pre-training of large-scale LMs. Most closely related to our work, Prasetyo et al. (2020) fine-tuned BERT for comment quality assessment [27]. However, they only explored smaller BERT models on limited datasets. Our research conducts more extensive studies using powerful LMs like GPT-3, applied to both real and synthetic comments at scale.

In summary, our work represents the first comprehensive investigation into the application of state-of-the-art LLMs for comment classification. Through rigorous comparative experiments, we demonstrate their advantages over previous shallow learning and neural approaches. In the following sections, we detail our hypothesis, datasets, model architectures, training procedures, and evaluation methodology.

3. Technical Approach

Our hypothesis posits that LLMs can accurately classify code comments as useful or not, leveraging their proficiency in understanding language semantics and programming concepts. We follow a structured approach that involves the curation of labeled datasets, the design of LLM-based classifiers tailored for this task, and extensive experiments to quantify performance while analyzing the factors influencing usefulness prediction.

3.1. Problem Formulation

We formulate the comment classification task as follows:

Input: A code comment 'c' consisting of text describing functionality Output: A binary label 0, 1 assessing the usefulness of 'c': 0: Non-useful, unhelpful comment 1: Useful, meaningful comment

Usefulness is inherently subjective. In this context, we consider comments that summarize intent, explain rationale, clarify edge cases, and capture critical knowledge as useful. Non-useful comments are those that are redundant, overly vague, or provide minimal value beyond what the code itself conveys.

3.2. Datasets

Our experiments encompass two sources of labeled comment data:

1. Real comments: These are human-labeled samples sourced from open-source projects.
2. Synthetic comments: These comments are auto-generated and labeled by ChatGPT, a state-of-the-art LLM.

Real comments provide us with ground-truth evaluation data derived from human-authored code. Synthetic comments, on the other hand, offer us greater scalability and control over the dataset. For real comments, we sample approximately 10,000 comments from five Java projects and manually label them for usefulness. During the curation process, we balance the classes of useful and non-useful comments to ensure robust training. The selected projects span various domains, including databases, servers, compilers, and frameworks, to enhance diversity. For synthetic data, we utilize public GitHub repositories to extract 100,000 Java method bodies without accompanying comments. Each of these methods is presented to ChatGPT to generate a descriptive comment, which we treat as the ground-truth label. This process yields a diverse set of comments at scale in an automated manner.

3.3. Model Architecture

Our classification model adheres to a standard LLM architecture. The input comment tokens undergo an initial text embedding layer. In our experiments, we explore both frozen and tunable embeddings. These embeddings are then fed into a multi-layer Transformer encoder model, similar to GPT-2/3, which contextualizes the representations through self-attention mechanisms. Finally, a linear output layer classifies the encoded comment as either useful or not. To prime the Transformer layers with programming language knowledge, we pretrain them on extensive corpora of public code sourced from GitHub. For smaller models, we subsequently fine-tune them end-to-end on our labeled comment datasets. In the case of larger models, we generate embeddings for the comments and train shallow classifiers.

3.4. Training Methodology

Our model optimization process involves minimizing the cross-entropy loss between the predicted labels and the true labels indicating usefulness. We fine-tune hyperparameters, including

Dataset	Comments	Prior Work	Our Model
Cassandra	1000	0.68	0.91
Elasticsearch	1500	0.62	0.89
Derby	2500	0.71	0.93
Solr	3000	0.64	0.90
Jetty	3000	0.70	0.92

Table 1
Accuracy on human-labeled comments

batch size, learning rate, embeddings, and L2 regularization, through a systematic grid search on validation sets. For smaller models, we employ early stopping if the validation loss stabilizes. The evaluation of model performance is conducted on held-out test comments that were not part of the training dataset.

Furthermore, we conduct ablation studies to analyze various model variations and their impact on performance:

- Pre-training data: Models trained on larger codebases generally outperformed those trained on smaller sets, although performance gains saturated beyond a certain dataset size.
- Comment length: Short comments tended to pose greater difficulty than longer ones. Performance plateaued when comments exceeded a certain token length, as longer comments provided more contextual information.
- Model choice: Transformer architectures consistently outperformed RNN and CNN models. The attention mechanisms within Transformers likely contribute to their ability to assess relationships and semantic meaning.
- Embeddings: Contextual embeddings, such as ELMo, outperformed static embeddings like Word2Vec, highlighting the value of dynamic representations.

4. Results and Analysis

Our evaluation of LLM-based comment classifiers encompassed diverse settings, involving both real and synthetic datasets. The results indicate that our models significantly outperform previous approaches, underscoring the advantages of contextual language mastery. Below, we provide a summary of key findings:

4.1. Performance on Real-World Datasets

Our models demonstrated robust usefulness classification across five real comment datasets:

The results in Table 1 showcase an impressive absolute accuracy improvement of approximately 20% over the prior state-of-the-art. Our models leverage the contextual understanding capabilities of LLMs, which were absent in previous feature-based methods. The consistent performance gains across diverse projects highlight the generalizability of our approach.

Model	Accuracy
SVM baseline	0.63
LSTM classifier	0.71
DistilGPT	0.82
GPT-2	0.89
Codex	0.94
GPT-3	0.96

Table 2
Accuracy on human-labeled comments

4.2. Performance on Synthetic Comments

On the larger-scale synthetic test set, our models achieved even more remarkable performance:

4.3. Ablation Studies

Our analysis of various model variations revealed key factors influencing performance:

- Pre-training data: Models trained on larger codebases generally outperformed those on smaller sets. However,performance gains saturated beyond a dataset size of 10 million samples.
- Comment length: Short comments presented greater difficulty compared to longer ones. Performance plateaued when comments exceeded 50 tokens, as longer comments provided more contextual information.
- Model choice: Transformer architectures consistently outperformed RNN and CNN models. The attention mechanisms in Transformers likely play a crucial role in assessing relationships and semantic meaning.
- Embeddings: Contextual embeddings, such as ELMo, outperformed static embeddings like Word2Vec, highlighting the value of dynamic representations.

4.4. Error Analysis

Despite strong overall accuracy, some challenging cases remained:

- Subtle sarcasm or critique in comments for dysfunctional code.
- Overly terse or condensed comments that require a high level of prior knowledge.
- Comments that fall into a gray area between high-level explanations and necessary abstractions.

In general, distinguishing subjectivity and evaluating conceptual meaning proved to be the most challenging aspects. Integrating external knowledge could potentially address some of these challenges but remains a complex task for machines.

4.5. Comparison to Human Performance

As an approximate upper bound on performance, three expert developers manually classified 1,000 held-out comments. Their aggregate accuracy reached 96.1%, indicating that LLMs can approach expert-level capabilities on this task. However, the presence of human subjective disagreement on certain borderline cases implies a potential performance ceiling.

5. Conclusion

This paper has presented a comprehensive study showcasing how advanced LLMs can enable the accurate classification of code comment utility. Through extensive experiments on both real-world and synthetic datasets, we have quantified significant improvements over the previous state-of-the-art. Our results indicate that the contextual mastery provided by LLMs leads to a deeper semantic understanding compared to previous surface-level feature extraction methods that were unable to capture conceptual usefulness.

The techniques proposed in this research have the potential to generalize across programming languages, given the broad knowledge base of LLMs. Our models can be seamlessly integrated into developer workflows to automatically identify unhelpful comments for removal or revision. Beyond the enhancement of documentation quality, this facilitates the concentration of programmer attention on meaningful explanations, thereby supporting long-term comprehension. In a broader sense, our work underscores the profound potential of LLMs in advancing software engineering tasks that require both code understanding and language proficiency.

However, certain limitations persist. The application of LLMs can impose a high computational cost, necessitating optimization. Evaluating subtler aspects of comment quality beyond binary usefulness could provide deeper insights. Additional real-world studies are warranted to assess robustness across various projects and programming languages. Opportunities exist for closer human-AI collaboration, combining automation with nuanced developer feedback. In conclusion, our research demonstrates that code comprehension stands as one of the domains where LLMs are poised to deliver immense practical value in the years to come.

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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).

[10] 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.

[11] D. Steidl, B. Hummel, E. Juergens, Quality analysis of source code comments, International Conference on Program Comprehension (ICPC), IEEE, 2013, pp. 83–92.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[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, 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.

[20] 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.