Applications of LLMs for Code Analysis

Adwita Deshpande[1,*]

[1]Indian Institute of Technology Goa,India - 403401

Abstract

The Software Engineering Information Retrieval (IRSE) track specializes in the construction of automated methods to assess code comments with a machine learning paradigm. In 2022, the track consisted of two main tasks: (i) estimating code comment usefulness, and (ii) estimating code quality. The first task is to classify code comments as useful and not useful. We created a dataset consisting of 9,048 pairs of code comments from open-source, C-based projects hosted on Github, as well as a second dataset created with people's help using large language models (LLMs). Specifically, 12 teams from universities completed this work; some of these teams planned and executed experiments with both quantitative and qualitative measurements. Code comments, while created with LLMs, introduce bias into the prediction model, but help reduce overfitting, resulting in more generalizable results. The second sub-track code quality estimation was created this year. The goal of the task is to auto-estimate the functional correctness of each code when given a problem description, and a set of code generated by large language models. For the purpose of evaluation, each problem-solution pair is then ranked by these estimated probabilities of functional correctness, the quality of which is then reported with standard ranking performance measures.

Keywords

Large Language Models, Comment Usefulness Prediction, Code Quality Estimation, bert, GPT-2

1. Introduction

Effective software maintenance relies heavily on code comprehensibility, where high-quality comments play a pivotal role. Well-written, informative comments can significantly improve code readability and ease the maintenance burden. However, the "usefulness" of a comment is often subjective and context-dependent. Previous research, such as Bosu et al. [1], has focused on evaluating code review comments from external tools. Yet, a clear need remains for models that can assess the quality of inline source code comments, which are fundamental to day-to-day maintenance activities.

Building on the work of Majumdar et al. [2], who proposed a framework for classifying comments based on their contribution to code comprehension, the IRSE track has evolved. The inaugural track at FIRE 2023 broadened the investigation into comment quality with diverse machine learning methods. In 2024, the focus shifted to incorporating "silver standard" quality labels generated by Large Language Models (LLMs).

This year, the IRSE track continues this exploration with two distinct sub-tasks. The first task challenges participants to build a binary classifier for comment usefulness, with an emphasis on using LLMs to augment the training data. The second, a new pilot sub-task, addresses the emerging challenge of automatically estimating the quality of LLM-generated code. Given a programming problem, the goal is to predict the functional correctness of multiple code solutions generated by an LLM. This task is analogous to query performance prediction (QPP) in traditional IR [3], where "relevance" is replaced by "functional correctness."

This paper summarizes the design, participation, and outcomes of both sub-tasks, offering insights into the current state of automated analysis for software engineering artifacts.

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

*Corresponding author.

adwita5b@gmail.com (A. Deshpande)

0009-0003-9442-3441 (A. Deshpande)

2. Related Work

The automatic analysis of software metadata is a well-established research area, with numerous tools developed to extract knowledge from artifacts like runtime traces and code structure [4, 5, 6, 7, 8, 9, 10, 11, 12]. Within this domain, assessing comment quality has been a persistent challenge. Early approaches by Steidl et al. [13] used lexical similarity and comment length to filter out trivial comments. Later work, such as Rahman et al. [14], focused on predicting the utility of code review comments in external portals.

Majumdar et al. [2, 15] introduced a more nuanced framework by evaluating comments based on concepts central to code comprehension, employing both textual and code correlation features. These foundational efforts paved the way for building predictive models to declutter codebases by identifying and flagging unhelpful comments. Similarly, [16, 17] tackles this task with the help of LLMs.

The advent of large language models [18] has opened new avenues for this research. It is now imperative to benchmark automated assessments from models like GPT-3.5 against human judgment. The IRSE track at FIRE builds directly on these advancements, investigating modern vector space models [19] and the impact of including LLM-generated labels to enhance classification performance.

3. Task and Datasets

We now describe the task and the dataset details of the two sub-tracks (ST) for IRSE.

3.1. ST-1: Comment Usefulness Prediction

This task requires participants to build a binary classification model to determine if a source code comment is `Useful` or `Not Useful`. Given a comment and its related code snippet, the model must assess whether the comment's information would genuinely help a developer understand the code.

The core of the classification logic rests on avoiding redundancy. A `Useful` comment must be relevant and provide insights that are not immediately obvious from the code itself. In contrast, a `Not Useful` comment, while potentially relevant, is redundant because it simply restates what the code already clearly communicates. To support this task, we provide the IRSE track dataset, which contains 9,048 annotated code-comment pairs from GitHub.

Table 1
Comparative performance metrics of submitted systems.

Affiliation/System	Base Dataset (Seed)			Augmented Dataset (Seed + LLM)		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
IIT KGP 1	0.8430	0.8570	0.8440	0.8470	0.8580	0.8580
IIT KGP 2	0.8110	0.8610	0.7930	0.8230	0.8250	0.8170
IIT KGP 3	0.7740	0.7240	0.7870	0.7910	0.7810	0.8050
IIT KGP 4	0.8110	0.8110	0.8220	0.8330	0.8130	0.8200
IIT KGP 5	0.7920	0.8450	0.8180	0.7890	0.8480	0.8170
IIT Goa 1	0.7900	0.8040	0.7940	0.7980	0.8020	0.7990
IIT Goa 2	0.8630	0.8760	0.8540	0.8910	0.8950	0.8930
IIT Goa 3	0.8360	0.8530	0.8350	0.8740	0.8720	0.8810
IIT Goa 4	0.7990	0.8050	0.7850	0.7930	0.8090	0.7990
IIT Goa 5	0.8090	0.8050	0.8150	0.8180	0.7910	0.8020
SRM Chennai 1	0.8130	0.7940	0.8010	0.8240	0.8510	0.8330
SRM Chennai 2	0.8150	0.8240	0.8010	0.8220	0.8430	0.8290

3.2. ST-2: Code Quality Estimation

Task and Evaluation Measures The objective of the code quality estimation task is to predict the **functional correctness** of code snippets that have been generated by Large Language Models (LLMs) in response to a programming prompt. For this purpose, we utilize the HumanEval dataset, which contains 161 programming problems.

Formally, given a programming task description P and a list of m generated solutions $\mathcal{S}^P = \{S_1^P, \ldots, S_m^P\}$, a prediction model θ is expected to produce a vector of likelihood scores. Each score represents the estimated functional correctness for a corresponding solution, such that $\theta : (P, \mathcal{S}^P) \mapsto \mathbb{R}^m$.

To evaluate the model, we frame this as a ranking problem, drawing an analogy from Information Retrieval. The problem description P acts as a query, the solutions $\mathcal{S}^P$ are analogous to a set of retrieved documents, and the notion of 'functional correctness' replaces 'relevance'. This analogy allows us to use standard ranking metrics. We primarily report the Normalized Discounted Cumulative Gain at m (nDCG@m), where we use $m = 10$ solutions per problem.

We employ two distinct nDCG measures to capture different aspects of ranking performance:

1. **Local nDCG (l-nDCG):** This metric assesses the average per-problem ranking quality. For each problem P in the set of all problems $\mathcal{P}$, we rank its m solutions based on the predicted scores and compute nDCG@m. The final l-nDCG is the average of these scores over all problems, defined as:

$$\text{l-nDCG} = \frac{1}{|\mathcal{P}|} \sum_{j=1}^{|\mathcal{P}|} \text{nDCG}_j$$

2. **Global nDCG (g-nDCG):** This metric evaluates the model's overall ranking ability across the entire benchmark. We pool all $m \times |\mathcal{P}|$ solutions from all problems into a single list, rank them using the predicted scores, and calculate a single nDCG score for this global list, denoted as nDCG@$(m|\mathcal{P}|)$.

4. Participation and Evaluation

In this section, we detail the participation and methodologies for the two sub-tasks.

ST-1: Comment Usefulness Prediction

The IRSE 2024 track attracted significant interest from the academic community, receiving 12 submissions from various research labs, reflecting the importance of software maintenance.

Approaches and Methodologies Participants employed a diverse range of techniques to tackle the classification problem. The provided training dataset was well-balanced, containing 4015 `useful` and 4033 `not useful` comments. For feature representation, teams utilized everything from traditional methods like TF-IDF and word2vec to context-aware embeddings from models like ELMo and BERT. The classification models were similarly varied, including support vector machines, logistic regression, and deep-learning architectures such as RNNs and BERT-based classifiers.

Impact of LLM-Generated Data A key aspect of this track was evaluating the effect of data augmentation using LLM-generated "silver standard" labels. The results were nuanced: while some teams saw a slight improvement in test accuracy, many observed a minor decrease (2-4%). This behavior is interpreted as a positive outcome, suggesting that the synthetic data acts as a regularizer, reducing the model's tendency to overfit to the original training set and potentially leading to better generalization. Table 2 characterizes the LLM-generated datasets contributed by each team.

Table 2
Characterization of the LLM-Generated Datasets

Team Name	Total Entries	Useful Entries	Not Useful Entries
IIT KGP 1	431	412	19
IIT KGP 2	1,228	730	497
IIT KGP 3	1,510	24	1,486
IIT KGP 4	202	185	17
IIT KGP 5	738	80	658
IIT Goa 1	236	93	143
IIT Goa 2	8,598	4,649	3,949
IIT Goa 3	335	314	21
IIT Goa 4	334	309	25
IIT Goa 5	237	186	51
SRM Chennai 1	263	130	133
SRM Chennai 2	150	65	85

		Evaluation Metrics	
Participant	Method	l-nDCG	g-nDCG
	GPT-3.5 ($\tau = 0.7$)	0.6595	0.9108
IIT KGP	GPT-3.5 ($\tau = 0.8$)	**0.6616**	**0.9109**
	GPT-3.5 ($\tau = 0.9$)	0.6602	0.9107
CodeBERT-CLS	CodeBERT CLS	0.6401	0.9036

Table 3
Evaluation of the submitted runs for three different temperature settings and the in-house baseline of CodeBERT-based embedding similarities.

ST-2: Code Quality Estimation

For the pilot task on code quality estimation, we received a submission from a team at IIT-KGP.

Submitted System The team's approach was based on the methodology proposed by [20], employing zero-shot inference with GPT-3.5 Turbo. For a given problem-solution pair, the model was prompted to generate a likelihood score indicating the solution's functional correctness. They submitted three distinct runs by varying the GPT decoder's temperature setting ($\tau \in \{0.7, 0.8, 0.9\}$). The simple and effective prompt used is shown in Figure 1.

```
Given the problem and the solution, generate a likelihood score between 0 and 1
indicating how relevant the solution is to the problem. Only state the score.
```

Figure 1: Prompt used for the code quality estimation task via GPT-3.5 zero-shot inference.

Baseline for Comparison To provide a reference point, we developed a heuristic-based baseline. This baseline estimates quality by measuring the variance of semantic similarities across all solution pairs for a given problem. The core assumption is that for a well-defined problem with a fixed function signature (as in the HumanEval dataset), correct solutions should be semantically similar. A high variance, therefore, may indicate a lack of consensus in the generated solutions, correlating with a higher probability of incorrectness. Semantic similarity was calculated using CLS embeddings from CodeBERT [21].

Table 3 shows that the GPT-based 0-shot inference produced better results than the in-house heuristic-

based baseline of estimating code quality as a measure of the topical diversity between the LLM-generated solutions.

5. Conclusions

The first sub-task of the IRSE track centered on the automated evaluation of code comment quality. Across the 12 participating teams, a central finding was the significant benefit of data augmentation using Large Language Models. The top-performing system's F1-score increased from **0.853** to **0.892** when its training data was supplemented with LLM-generated labels. This improvement highlights that synthetic annotations can effectively mitigate model overfitting and enhance generalization. The value of this approach was further reinforced when a combined dataset, incorporating submissions from all teams and gold-standard labels, demonstrated even greater performance.

In the second sub-task, which focused on predicting the functional correctness of LLM-generated code, the results were equally decisive. Evaluation methods that themselves leveraged LLMs substantially outperformed embedding-based baseline models. This outcome underscores the advanced capabilities of LLMs to analyze the deep contextual and functional semantics of code, a task where simpler vector-space models are less effective.

Declaration on Generative AI

During the preparation of this work, the author(s) used ChatGPT in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publicationâsÂ content.

References

[1] 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.

[2] 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.

[3] S. Datta, D. Ganguly, D. Greene, M. Mitra, Deep-qpp: A pairwise interaction-based deep learning model for supervised query performance prediction, in: WSDM, ACM, 2022, pp. 201–209.

[4] 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.

[5] 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.

[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, 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 âinformation retrieval in software engineeringâ(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] 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.

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

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

[14] 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.

[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] 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.

[17] 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).

[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. 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.

[20] T. Y. Zhuo, Ice-score: Instructing large language models to evaluate code, 2024. arXiv:2304.14317.

[21] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, Codebert: A pre-trained model for programming and natural languages, CoRR abs/2002.08155 (2020). URL: https://arxiv.org/abs/2002.08155. arXiv:2002.08155.