LLMs for Code: Overview of the information retrieval in software engineering track at fire 2024

Soumen Paul[1], Srijoni Majumdar[2], Raj Shah[3], Susmita Das[4], Madhusudan Ghosh[6], Debasis Ganguly[4], Gul Calikli[4], Debarshi Sanyal[6], Partha Pratim Das[8], Paul D. Clough[5], Ayan Bandyopadhyay[7] and Samiran Chattopadhyay[7]

[1]*Indian Institute of Technology Kharagpur*
[2]*University of Leeds, UK*
[3]*Indian Institute of Technology Goa*
[4]*University of Glasgow, UK*
[5]*Sheffield University, Sheffield, UK*
[6]*Indian Association of Cultivation of Science, Kolkata*
[7]*Techno India University, India*
[8]*Ashoka University, India*

Abstract

The Software Engineering Information Retrieval (IRSE) track focuses on developing automated methods to evaluate code comments using a machine learning framework. This year, the track featured two key tasks: (i) *predicting the usefulness of code comments* and (ii) *estimating code quality*. The first task focuses on distinguishing code comments as either useful or not useful. The dataset comprises 9,048 pairs of code comments sourced from open-source C-based projects on GitHub, along with an additional dataset generated by teams utilizing large language models (LLMs). A total of 12 teams from various universities contributed to this effort, conducting experiments that were evaluated using both quantitative and qualitative metrics. Notably, while labels generated by large language models introduce bias into the prediction model, they also contribute to reducing overfitting, leading to more generalizable results. The sub-track pertaining to code quality estimation was introduced this year. Given a problem description, and a list of large language model (LLM) generated software code, the objective of the task is to automatically estimate the functional correctness of each generated code. 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

Evaluating the quality of comments is essential for optimizing codebases and improving code maintainability. Clear and well-organized comments can greatly enhance the readability and understanding of the code as long as they are consistent and informative.

Perceptions of comment quality, especially regarding their "usefulness," are context-dependent and can vary across different situations. Bosu et al. [1] sought to evaluate code review comments from a separate tool, focusing on their effectiveness in helping developers write better code. This evaluation was based on a comprehensive survey conducted at Microsoft. However, there is a need for a similar quality assessment model specifically designed to analyze source code comments that are crucial for routine maintenance tasks.

Majumdar et al. [2] developed a framework for evaluating comment quality, classifying comments as "useful," "partially useful," or "not useful" based on their ability to enhance the understandability of

Forum for Information Retrieval Evaluation, December 12-15, 2024, India
soumenpaul165@gmail.com (S. Paul); majumdar.srijoni@gmail.com (S. Majumdar); raj.shah.21031@iitgoa.ac.in (R. Shah); 2956827d@student.gla.ac.uk (S. Das); madhusuda.iacs@gmail.com (M. Ghosh); debasis.ganguly@glasgow.ac.uk (D. Ganguly); handangul.calikli@glasgow.ac.uk (G. Calikli); debarshisanyal@gmail.com (D. Sanyal); partha.das@ashoka.edu.in (P. P. Das); p.d.clough@sheffield.ac.uk (P. D. Clough); bandyopadhyay.ayan@gmail.com (A. Bandyopadhyay); samiran.chattopadhyay@jadavpuruniversity.in (S. Chattopadhyay)

nearby code snippets. The authors utilize a machine learning framework to assess comments, focusing on concepts that support code comprehension and identifying redundancies or inconsistencies in relation to the code. These concepts were informed by exploratory studies involving developers from seven companies and insights gathered from the broader community through crowd-sourcing.

In the first iteration of the IRSE track at FIRE 2022, the research in [2] was expanded to include an empirical investigation into comment quality, employing a broader range of machine learning techniques and features. In 2023, the IRSE track advanced further by introducing a challenge that focuses on evaluating the feasibility of incorporating silver standard quality labels generated by Large Language Models (LLMs). This year also, the task is to extend the dataset using LLM-generated data. The goal is to determine how this addition enhances the predictive capabilities of the classification model. Establishing a gold industry standard for assessing the usefulness of comments that facilitate understanding of code, particularly in legacy systems, is a challenging and time-consuming endeavor. Nonetheless, generating a larger dataset is essential for broadening the model's applicability across different programming languages, which is being pursued through the use of large language models.

The performance of these models, especially in understanding the relationships between code and comments, can provide valuable insights into the quality of the generated data and its potential for scaling the existing classification model. Additionally, this approach can be generalized to any classification model based on software metadata.

In addition to the core track of comment usefulness detection, this year - as a pilot track, we introduced a new sub-track pertaining to quality estimation of large language model (LLM)-generated code. More precisely speaking, given a problem description comprised of a natural language description of a programming task along with a partially written code, e.g., the function prototype, a common practice is to employ generative AI models (instruction tuned LLMs specifically fine-tuned on software code, e.g., CodeLlama, Codestral etc.) to automatically generate code to solve the given task. Given a pair comprised of a problem description along with an LLM-generated solution the objective is to estimate the likelihood that the code is functionally correct, i.e., it provides a correct solution to the problem. As analogy, this task is somewhat similar to the task of query performance prediction (QPP) [3, 4] in IR, where the objective is to estimate the quality of a retrieved list of documents in terms of relevance (which is substituted by the notion of 'functional correctness' in the context of our problem).

2. Related Work

Software metadata is essential for code maintenance and understanding. Numerous tools [5, 6, 7, 8, 9, 10] have been developed to facilitate the extraction of knowledge from software metadata, such as runtime traces and structural attributes of code.

In the context of mining code comments and assessing their quality, authors [11, 12, 13] analyze the similarity of words in code-comment pairs using Levenshtein distance and comment length to filter out trivial and non-informative comments. Rahman et al. [14] identify useful and non-useful code review comments in logged review portals based on attributes identified through a survey of Microsoft developers [1]. Majumdar et al. [2, 15] proposed a framework for evaluating comments based on concepts relevant to code comprehension. They developed textual and code correlation features utilizing a knowledge graph for semantic interpretation of the information in comments. These approaches leverage semantic and structural features to establish a prediction framework for classifying comments as "useful" or "not useful," which can then be integrated into efforts to declutter codebases.

With the rise of large language models [16], it is crucial to compare the quality assessment of code comments generated by standard models like GPT-3.5 or LLaMA with human interpretations. The IRSE track at FIRE 2023 builds on the approach proposed in [2] to investigate various vector space models [17] and features for the binary classification and evaluation of comments in the context of understanding code. This track also evaluates the performance of the prediction model with the inclusion of GPT-generated labels for the quality of code and comment snippets sourced from open-source software.

Table 1
Test data predictions of the submitted systems.

	Seed			Seed + LLM-augmented		
Affiliation	P	R	F1	P	R	F1
IIT KGP 1	0.8426	0.8576	0.8428	0.8462	0.8582	0.8573
IIT KGP 2	0.8100	0.8600	0.7923	0.8221	0.8241	0.8164
IIT KGP 3	0.7738	0.7233	0.7863	0.7900	0.7802	0.8046
IIT KGP 4	0.8100	0.8103	0.8212	0.8321	0.8121	0.8195
IIT KGP 5	0.7916	0.8446	0.8172	0.7886	0.8470	0.8167
IIT Goa 1	0.7901	0.8043	0.7942	0.7976	0.8017	0.7987
IIT Goa 2	0.8621	0.8750	0.8530	0.8900	0.8940	0.8920
IIT Goa 3	0.8350	0.8520	0.8340	0.8730	0.8710	0.8800
IIT Goa 4	0.7983	0.8040	0.7841	0.7922	0.8086	0.7985
IIT Goa 5	0.8283	0.8040	0.8141	0.8178	0.7906	0.8013
SRM Chennai 1	0.8120	0.7930	0.8000	0.8231	0.8500	0.8320
SRM Chennai 2	0.8143	0.8231	0.8000	0.8213	0.8423	0.8287

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

Comment Classification : The task involves binary classification of source code comments as either *Useful* or *Not Useful* given a comment and its associated code snippet as input. The output is based on whether the information contained in the comment is relevant, *and* would help comprehend the surrounding code, i.e., it is *useful*.

- *Useful* Comments have sufficient software development concept $\rightarrow$ Comment is Relevant, and these concepts are not primarily present in the surrounding code $\rightarrow$ Comment is not Redundant.
- *Not Useful* Comments have sufficient software development concept $\rightarrow$ Comment is Relevant, and these concepts are mostly present in the surrounding code $\rightarrow$ Comment is Redundant.

Dataset: For the IRSE track, we use a set of 9048 comments (from Github) with comment text, surrounding code snippets, and a label that specifies whether the comment is useful or not.

3.2. ST-2: Code Quality Estimation

Task and Evaluation Measures We scope the code quality estimation task to estimate the *functional correctness* of code snippets generated via LLMs in response to a prompt specifying a programming task. In particular, we make use of the HumanEval[1] dataset for this task, which constitutes of 161 programming problem descriptions.

Given a programming task description P, and a list of m solutions $\mathcal{S}^P = \{S_1^P, \ldots, S_m^P\}$ generated by an LLM, a predictor model θ should estimate a likelihood score of the functional correctness of each solution, i.e., $\theta : P, \mathcal{S} \mapsto \mathbb{R}^m$.

An effective model should estimate a high likelihood value for a functionally correct solution (the ground-truth being computed via a set of test-cases), which means that a standard evaluation metric for a ranking task may also be applied here - the only difference being the notion of 'relevance' replaced with that of 'functional correctness' (P being analogous to a query and $\mathcal{S}^P$ to that of a set of top-m retrieved documents). Motivated by this analogy, we report nDCG@m (in our setting, $m = 10$, i.e., 10 solutions are generated for each problem) as an evaluation measure.

[1] https://huggingface.co/datasets/openai/openai_humaneval

Table 2
Characterizations of the LLM Generated datasets

Team name	Total entry	Useful entry	Not useful entry
IIT KGP 1	431	412	19
IIT KGP 2	1228	730	497
IIT KGP 3	1510	24	1486
IIT KGP 4	202	185	17
IIT KGP 5	738	80	658
IIT Goa 1	236	93	143
IIT Goa 2	8598	4649	3949
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

Additionally, we also report a global ranking effectiveness measure to compare across the performance over all problem tasks. Specifically, we use the predicted likelihoods to rank all the P, S_i^P pairs for each $P \in \mathcal{P}$ (the set of all problem tasks in a benchmark), and compute the nDCG value of this set, i.e., nDCG@$(m|\mathcal{P}|)$.

To differentiate the two measures, we call the former local nDCG (**l-nDCG**) and the latter global nDCG (**g-nDCG**). More precisely speaking, to calculate **l-nDCG**, we rank P, S_i^P pairs for each problem P, calculate nDCG and then calculate the average of nDCG values for all $P \in \mathcal{P}$ (i.e., $\sum_{j=1}^{|\mathcal{P}|}(nDCG^j/|\mathcal{P}|)$), whereas to calculate **g-nDCG**, we rank all $m|\mathcal{P}|$ pairs of P, S_i^P for all $P \in \mathcal{P}$ and then calculate the nDCG value.

4. Participation and Evaluation

4.1. ST-1: Comment Usefulness Prediction

IRSE 2024 received a total of 12 experiments from 12 teams for the two tasks. As this track is related to software maintenance, we received participation from several research labs of educational institutes.

The various teams with the details of their submissions are characterized in Table 1. The dataset provided was balanced and had 4015 useful comments and 4033 not useful comments. The participants used various pre-trained embeddings such as one hot encoding, TF-IDF vectorizer, word2vec, or context-aware like ELM or BERT to generate vectors for the word sequence. Teams have used several machine learning models like support vector machine, logistic regression, and deep-learning based models such as BERT, Recurrent neural network, and so on.

Some participants were observed to achieve a slight increase in test accuracy when the model was trained with the addition of an LLM-generated dataset (Table 2). However, in many cases, the accuracy reduces (2%-4%). This behavior is due to the incorporation of silver standard data that reduces the over-fitting of the models.

4.2. ST-2: Code Quality Estimation

A team from IIT-KGP participated in this task. Similar to the methodology proposed in [18], they employed GPT-3.5 Turbo zero-shot inference on a problem description and a solution pair to estimate how likely is the solution to be functionally correct. They submitted three runs with three different temperature (τ) settings for the GPT decoder (specifically, $\tau = 0.7$, $\tau = 0.8$ and $\tau = 0.9$). The prompt used by the participating team is shown in Figure 1.

To set a reference point for comparison purposes, we employed a relatively simple heuristic baseline which given a problem description and a list of solutions measures the variance across the semantic

Given the problem {Problem} and the solution {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 by the participating team for the code quality estimation task via GPT-3.5 0-shot inference.

		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.

similarities between each solution pair. For measuring the semantic similarity between a pairs of code solutions, we use the CLS embeddings obtained from CodeBERT [19], a BERT model fine-tuned on large volumes of source code data.

The assumption of using the variance across generated code solutions as an estimate is that topical diversity of the solutions may indicate lack of consistency in the solutions being generated, which could be be associated with a risk of the solutions being incorrect. Making this assumption is appropriate since HumanEval dataset contains the method signature for each problem. Although there can be different ways to provide solution S (i.e., implement code) for a problem P (e.g., by using different data structures or implementing a recursive algorithm instead of an iterative one, the method signature limits the way one can provide a solution S for a given problem P.

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 focused on exploring various automated approaches for evaluating comment quality. The assessment criteria were based on whether a comment provided useful information that enhanced the comprehension of the surrounding code. A total of 12 teams participated, employing diverse machine learning models, embedding techniques, feature sets, and LLM-generated data. The highest F1-score achieved was 0.853, with a notable improvement to 0.892 when incorporating LLM-generated data. The inclusion of LLM-generated labels not only mitigated overfitting in classification models but also enhanced performance. Furthermore, when combining data from all participants with gold standard labels from industry practitioners, the overall F1-score improved, demonstrating the effectiveness of data augmentation through LLM-generated annotations.

The second sub-task of the IRSE track focused on assessing the effectiveness of predictive models in estimating the functional correctness of LLM-generated code. The evaluation revealed that LLM-based approaches for code quality estimation outperformed embedding-based baselines, demonstrating their superior ability to capture contextual and functional correctness aspects of generated code.

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] D. Ganguly, S. Datta, M. Mitra, D. Greene, An analysis of variations in the effectiveness of query performance prediction, in: ECIR (1), volume 13185 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 215–229.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] M. P. O'brien, Software comprehension–a review and research direction, Technical Report Technical Report, Department of Computer Science & Information Systems University of Limerick, Ireland, 2003.

[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] 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.

[13] 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.

[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] 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.

[17] 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.

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

[19] 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.