Overview of the IRSE track at FIRE 2025: Information Retrieval in Software Engineering

Adwita Deshpande[1,†], Aritra Maji[2,†], Diganta Mondal[2,†], Partha Pratim Das[2], Paul D. Clough[3] and Srijoni Majumdar[2,4]

[1]*Indian Institute of Technology Goa, Goa, India*
[2]*Indian Institute of Technology Kharagpur, Kharagpur, India*
[3]*University of Sheffield, Sheffield, UK; TPXimpact, London, UK*
[4]*University of Leeds, Leeds, UK*

Abstract

The Information Retrieval in Software Engineering (IRSE) track focuses on the automated assessment of software artifacts using machine learning techniques. The 2025 edition emphasizes two tasks: (i) binary classification of source code comments into useful and not useful categories, and (ii) a pilot task on estimating the functional correctness of code generated by Large Language Models (LLMs). The primary dataset comprises 9,048 comment–code pairs extracted from open-source C projects on GitHub. Fourteen teams from academia and industry submitted a total of 45 experimental runs. Results indicate that transformer-based models with software-specific embeddings outperform traditional classifiers, while LLM-based inference proves highly effective for code quality estimation.

Keywords

Information Retrieval in Software Engineering, Comment Usefulness Prediction, Code Quality Estimation, Large Language Models, Transformers

1. Introduction

Evaluating the quality of comments can help streamline codebases and enhance overall code maintainability. When consistent and informative, comments play a vital role in improving code readability and comprehension. However, the perceived usefulness of comments is subjective and varies depending on the context. Bosu et al. [1] conducted a detailed survey at Microsoft to assess the utility of code review comments (recorded in a separate tool) in aiding developers to write better code. While such models exist for review comments, an equivalent framework for assessing the quality of source code comments—particularly in the context of standard maintenance tasks—remains underdeveloped. Majumdar et al. [2] introduced a comment quality evaluation framework that categorizes comments as 'useful', 'partially useful', or 'not useful', depending on their contribution to the readability of nearby code segments. Building on this, the IRSE track of FIRE 2025 expands the work in [2] by empirically analyzing comment quality using a broader range of machine learning models and feature sets. In its inaugural edition, the IRSE track focuses on classifying comments into two categories—'useful' and 'not useful'—based on whether they convey information not apparent from the surrounding code design and thus enhance code comprehensibility. A total of 34 experiments were submitted by 11 teams (further details in [3, 4, 5, 6]).

2. Related Work

Steidl et al. [7] introduced a method for detecting comment quality by measuring the similarity between code and comment text using the Levenshtein distance, along with comment length, to eliminate

Forum for Information Retrieval Evaluation, December 17–20, 2025, India
[†]These authors contributed equally to this work.
adwita5b@gmail.com (A. Deshpande)
0009-0003-9442-3441 (A. Deshpande)

trivial or non-informative comments. Rahman et al. [8] identified useful and non-useful code review comments (logged in review portals) using features derived from a developer survey conducted at Microsoft [1]. They applied textual features and trained classifiers—specifically decision tree and naive Bayes models—on a dataset of 1,200 review comments for automated quality evaluation. In more recent work, Liu et al. [9] addressed the Declutter Challenge of DocGen2, where they identified 'not useful' comments using textual and structural features within a machine learning framework. Majumdar et al. [2] proposed a framework to assess comment quality by focusing on concepts that enhance code comprehension. Their approach utilized textual and code correlation features, leveraging a knowledge graph for semantic interpretation of the information embedded in comments, [10], [11] deploy similar ideas, and [12] follows the same idea as in this paper.

Existing methods largely focus on evaluating comment quality by identifying irrelevant or repetitive words and phrases in relation to nearby code constructs. However, the definition of quality is inherently context-dependent. For instance, Rahman et al. [8] restricted their assessment to code review comments, while Majumdar et al. [2] analyzed source code comments by extracting concepts that facilitate comprehension and aid in maintenance. The IRSE track extends the work of [2] by exploring diverse vector space models and feature representations to evaluate comments in the context of their contribution to code understanding.

3. Task and Dataset

The task is defined as a binary classification problem aimed at categorizing source code comments as either **Useful** or **Not Useful** based on a given comment and its corresponding code snippet.

A **Useful** comment contains meaningful software development concepts that are not explicitly reflected in the surrounding code, thereby enhancing its comprehensibility. Conversely, a **Not Useful** comment either contains very few relevant concepts or includes sufficient software development information that is already evident in the surrounding code, making it *redundant*.

```
/* Initializes the buffer only once to avoid redundant
allocations */
---- USEFUL
void init_buffer() {
    static int initialized = 0;
    if (!initialized) {
        buffer = malloc(BUFFER_SIZE);
        initialized = 1;
    }
}

/* Function to initialize buffer */
---- NOT USEFUL, REDUNDANT
void init_buffer() {
    static int initialized = 0;
    if (!initialized) {
        buffer = malloc(BUFFER_SIZE);
        initialized = 1;
    }
}
```

3.1. Dataset:

The IRSE track utilizes a robust dataset comprising **9,048** comment-code pairs extracted from open-source C-based projects on GitHub. Each data instance is a tuple consisting of the comment text, the associated surrounding code snippet (function or block), and a binary ground-truth label indicating

whether the comment is *useful* or *not useful*. The rigorous data collection process ensured that the comments evaluated were pertinent to real-world software maintenance scenarios.

3.2. Code Quality Estimation

The pilot task focuses on predicting functional correctness of LLM-generated solutions using the HumanEval benchmark. Given a problem description and multiple candidate solutions, systems rank solutions by estimated correctness using IR-style ranking metrics such as nDCG.

4. Participation and Evaluation

The 2025 edition of the IRSE track witnessed active engagement from the software engineering research community, receiving experimental submissions from 14 diverse teams. The participation featured a strong mix of academia and industry, reflecting the universal relevance of the problem statement. The cohort comprised 7 teams from the Indian Institute of Technology (IIT) Kharagpur, 4 from IIT Goa, 2 from SSRN College, and 1 from Salesforce. The track focuses on software maintenance, a critical phase in the software lifecycle, and participant details are outlined in Table 1.

The provided dataset was meticulously balanced to prevent class bias during training, containing 4,208 instances labeled as "useful" and 4,235 instances labeled as "not useful." This balance allowed for a fairer evaluation of model performance using metrics like the F1-score. In terms of feature extraction, participants primarily employed feature engineering techniques centered on text mining to isolate significant keywords and phrases. Additionally, almost all teams utilized string matching and overlap coefficients to quantify the redundancy between the comment text and the source code tokens.

4.1. Machine Learning Architectures:

A significant number of teams utilized state-of-the-art transformer-based models such as BERT and GPT to learn the comment quality labels. These models were typically fine-tuned over 50 to 75 epochs, utilizing binary cross-entropy loss functions and the Adam optimizer to minimize classification error. While the GPT architecture achieved the highest F1 score—demonstrating superior capability in understanding context—its deployment requires substantial computational resources, which are typically more accessible to large software companies like Salesforce.

In contrast, other teams employed more traditional architectures including Recurrent Neural Networks (RNNs), Support Vector Machines (SVMs), Random Forests, and Logistic Regression. These models often relied on a hybrid of textual features (e.g., length, keyword frequency) and code-correlation features. Interestingly, the F1 scores obtained from RNNs and SVMs were comparable to those obtained from the more complex BERT variants. This competitiveness is attributed to the balanced nature of the dataset and the high discriminative power of explicit textual features when combined with numerical vectors, suggesting that lighter models can still be effective for this specific task.

4.2. Pre-Trained Embeddings:

The experiments involved a comprehensive exploration of embedding spaces, utilizing both context-aware and context-independent embeddings. These were either trained from scratch on specific corpora or fine-tuned with software development concepts. The best results were consistently achieved using the recently released contextualized CodeBERT embeddings [13], which are pre-trained on bimodal data (natural language and programming language), allowing them to capture the semantic relationship between code and comments effectively.

Comparable results were also obtained by using CodeELMo [2], which trains ELMo embeddings from scratch using large-scale software development corpora from books, journals, and code repositories. Conversely, while TF-IDF vectorization was frequently used by several teams to generate word vectors, it generally did not generate high scores. This underperformance highlights the limitation of statistical

Table 1
Ranking F1-Scores

Team Name	Model #Runs	Macro F1-Score
SSN College Team 1	2 CodeBERT	0.9193
Salesforce	4 Transformer (GPT-2)	0.9102
IIT KGP Team 1	3 CodeBERT	0.9085
IIT KGP Team 2	2 Transformer (BERT)	0.9010
IIT Goa Team 1	4 SVM-Radial Basis Function	0.8845
IIT KGP Team 3	5 Transformer (ALBERT)	0.8750
IIT KGP Team 4	3 Random Forest	0.8620
SSN College Team 2	2 Graph Neural Network	0.8250
IIT Goa Team 2	3 Naive Bayes	0.8250
IIT KGP Team 5	4 SVM	0.8015
IIT Goa Team 3	4 Bi-LSTM	0.7890
IIT KGP Team 6	3 Logistic Regression	0.7540
IIT Goa Team 4	3 Random Forest	0.6550
IIT KGP Team 7	3 SVM	0.5820

Table 2
Feature Characterisation and Machine Learning Frameworks - Comment Quality Evaluation

Team Name	Textual	Code Correlation	Vector Space	Models used
SSN College Team 1	PRE	None	CodeBERT(C+C)	Transformer
Salesforce	PRE	StrMatch	CodeBERT (C+C)	Transformer
IIT KGP Team 1	PRE	StrMatch	CodeBERT (C+C)	Transformer
IIT KGP Team 2	PRE	StrMatch	BERT (uncased)	Transformer
IIT Goa Team 1	PRE; MINE	None	ELMo (Code)	SVM
IIT KGP Team 3	PRE	StrMatch	ALBERT	Transformer
IIT KGP Team 4	PRE; MINE	StrMatch	TF-IDF based	Random Forest
SSN College Team 2	PRE	StrMatch	Node embeddings	Graph Neural Network
IIT Goa Team 2	PRE	None	TF-IDF based	Naive Bayes
IIT KGP Team 5	PRE; MINE	StrMatch	TF-IDF based	SVM
IIT Goa Team 3	PRE; DEP	LemMatch	TF-IDF based	Bi-LSTM
IIT KGP Team 6	PRE	None	CBOW	Logistic Regression
IIT Goa Team 4	PRE; MINE	None	CBOW	Random Forest
IIT KGP Team 7	PRE; MINE	None	TF-IDF based	SVM

counting methods in capturing the deep semantic context required to distinguish between a useful explanation and a redundant restatement of code.

5. Conclusions

The Information Retrieval in Software Engineering (IRSE) track represents a significant step forward in empirically investigating a diverse range of approaches within a machine-learning framework to automate the evaluation of comment quality. Central to this evaluation is the definition of quality, which is rigorously assessed based on whether a comment provides semantic information that aids in the comprehension of the surrounding code, rather than mere syntactic correctness.

This iteration of the track saw robust engagement from the research community, with 14 teams participating and submitting a total of 45 distinct experiments. This diverse participation facilitated a comprehensive comparative analysis of various machine learning models, embedding spaces, and feature engineering strategies, ranging from traditional statistical classifiers to advanced neural networks.

The results highlight a clear trend toward the efficacy of large language models in software maintenance tasks. The highest F1-Score of 0.9102 was achieved by the Salesforce team using the GPT-2

architecture, combined with hybrid textual and numerical features derived from CodeBERT vector space embeddings. This superior performance in identifying 'useful' versus 'not useful' comments underscores the potential of combining generative pre-trained transformers with domain-specific embeddings to build the next generation of intelligent automated software maintenance tools.

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] A. Bosu, M. Greiler, C. Bird, Characteristics of useful code reviews: An empirical study at microsoft, in: Proceedings of the 12th Working Conference on Mining Software Repositories (MSR), 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] 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.

[4] 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.

[5] 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).

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

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

[8] M. M. Rahman, C. K. Roy, R. G. Kula, Predicting usefulness of code review comments using textual features and developer experience, in: Proceedings of the International Conference on Mining Software Repositories (MSR), IEEE, 2017, pp. 215–226.

[9] M. Liu, Y. Yang, X. Peng, C. Wang, C. Zhao, X. Wang, S. Xing, Learning based and context aware non-informative comment detection, in: Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2020, pp. 866–867.

[10] 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).

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

[12] 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.

[13] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).