Code Comment Classification with Data Augmentation and Transformer-Based Models

Mushfiqur Rahman
Bangladesh University of Engineering and Technology
Dhaka, Bangladesh
himel6087@gmail.com

Mohammed Latif Siddiq
University of Notre Dame
Notre Dame, IN, USA
msiddiq3@nd.edu

***Abstract*—Effective classification of code comment sentences into meaningful categories is critical for software comprehension and maintenance. In this work, we present a solution for the NLBSE'25 Code Comment Classification Tool Competition, achieving a 6.7% improvement in accuracy over the baseline STACC models. Our solution employs a multi-step methodology, beginning with translation-retranslation techniques to generate synthetic datasets. By translating the original dataset into multiple languages and back into English, we introduce linguistic diversity that enriches the training data and improves model generalization. We fine-tuned transformer-based architectures, including BERT, CodeBERT, RoBERTa, and DistilBERT, on this augmented dataset. After extensive evaluation, the best-performing model is selected for a robust multi-label classification framework tailored to Java, Python, and Pharo databases. The framework is designed to address the unique challenges of each programming language, ensuring high precision, recall, and F1 scores across all 19 categories. The source code is publicly available at https://github.com/Mushfiqur6087/NLBSE-25, and the trained model can be accessed at https://huggingface.co/MushfiqurRR.**

***Index Terms*—code comment, data augmentation, transformers, BERT**

I. Introduction

Code comments are a cornerstone of software documentation, offering critical insights into the design, behavior, and usage of software systems [1]. They serve as a key resource for developers seeking to understand and maintain software, especially during complex evolution and maintenance tasks. Despite their importance, the practices surrounding code comments vary significantly across programming languages and projects, with many software systems suffering from inadequate or inconsistent commenting [1]. This inconsistency poses challenges to program comprehension and hampers the ability to maintain high-quality documentation over time.

Prior studies have explored various aspects of code comments, focusing on their role in program comprehension, taxonomies, and co-evolution with source code. Rani et al. [1] proposed a taxonomy for class comments across Python, Java, and Smalltalk. Pascarella and Bacchelli [2] highlighted the varied purposes of comments in Java projects. Steidl et al. [3] analyzed comment quality, emphasizing documentation importance. At the same time, Fluri et al. [4] demonstrated that comments often fail to co-evolve with source code, leading to outdated documentation.

Current tools, such as Dopamin [5] and STACC [6] baseline, have demonstrated significant advancements in code comment classification; however, their performance is limited by the small size of the available dataset, which restricts the models' ability to generalize effectively and fully leverage the capabilities of transformer-based learning. To address these challenges, our work aims to enhance code comment classification accuracy by leveraging advanced transformer-based models and novel data augmentation techniques.

For that reason, we generated synthetic datasets using LLM-generated translation, where the original code comment dataset was translated into multiple languages and then back into English. Studies explore the use of large language models (LLMs) for synthetic data generation, curation, and augmentation, with applications ranging from depression prediction to addressing challenges in data augmentation and learning paradigms [7], [8], [9]. We utilized Helsinki NLP [10] language models to translate the dataset across 6 languages. Then, we employed sentence transformers [11] to identify similar data and filter out noise, ensuring higher-quality training data. Here, noise refers to the retranslated sentences that deviate significantly in meaning from the original comments during translation-retranslation. Then, we evaluate multiple models and select the best-performing architecture for a multi-label classification framework. This framework classifies comments and captures nuanced semantic information, leveraging deeper transformer layers for improved context comprehension. The model effectively accounts for language-specific comment structures and taxonomies by tailoring the solution to each programming language. The efficiency of our solution is further proved in experimental results, which achieves a higher F1 score in almost all categories. It achieves an F1-score of 0.70, surpassing the 0.63 F1-score of the existing STACC [6] baseline. The source code of this project is publicly available on GitHub[1], and the trained model can be accessed on Hugging Face[2].

II. Dataset Preparation

In this section, we describe step by step how to prepare the dataset for the tool competition [12].

[1]https://github.com/Mushfiqur6087/NLBSE-25
[2]https://huggingface.co/MushfiqurRR

A. Dataset Summary

The dataset from the NLBSE tool competition [12] consists of **1,733** manually labeled class comments containing **14,875 sentences** extracted from **20** open-source projects written in Java, Pharo, and Python. Each comment sentence belongs to one or more categories specific to each programming language, representing different types of information conveyed in the comments.

Dataset Structure: Each row in the dataset represents a sentence with the following attributes:

- **class:** The class name of the source code file where the sentence originates.
- **comment_sentence:** The actual text of the sentence; part of a multi-line class comment.
- **partition:** Indicates the dataset split; 0 for training instances and 1 for testing instances.
- **combo:** The class name appended to the sentence string; is used for baseline training.
- **labels:** A binary list indicating the ground-truth categories to which the sentence belongs. Each sentence can belong to one or more categories.

In Table I, **Initial train data** column shows the initial size of the dataset for each language.


```
[
    {
        "comment_sentence": "this impl delegates to the old filesystem",
        "french_translation": "Insérer des délégués dans l'ancien
    système de fichiers",
        "augmented_sentence": "Insert delegates into the old file system",
        "similarity_score": "0.5195"
    },
    {
        "comment_sentence": "available in the clustering",
        "spanish_translation": "están disponibles en el cluster",
        "augmented_sentence": "are available in the cluster",
        "similarity_score": "0.7593"
    }
]
```


Fig. 1. Similarity score between dataset and augmented dataset.

B. Dataset Augmentation

In this part, we implemented a process for translation and retranslation to augment the dataset. Using pre-trained Helsinki-NLP models [10], we set up translation pipelines optimized for memory efficiency and faster computation by leveraging half-precision and multi-GPU support. We translated text from English to six different languages—German, French, Chinese, Hindi, Spanish, and Russian—and then retranslated the text back to English. These six languages were chosen because they are among the most commonly used translation languages in Hugging Face and have undergone better training, ensuring higher translation quality. The retranslated text was added as a new column in the dataset for each target language, and this method was applied to all three programming languages (Java, Python, and Pharo).

We also experimented with translation-retranslation using 10 and 16 different languages and fine-tuned a RoBERTa base model with the resulting augmented datasets. However, these models performed worse because the larger number of languages introduced redundancy, leading to reduced dataset diversity and a decline in overall dataset quality. This highlights the importance of selecting languages that generate diverse and meaningful augmentations for optimal model performance.

C. Filtering Augmented Dataset

In this part, we initialize a pre-trained sentence embedding model (all-MiniLM-L6-v2[3]) to generate embeddings for the comments. Our goal was to compare retranslated languages' semantic similarity with the originals and retain only those translations that preserved the original meaning. We generated embeddings for both the original comments and their retranslated versions in various languages (e.g., German, French, Chinese). Using cosine similarity, We compared each retranslation with its corresponding original comment. In Figure 1, we can see the similarity between original and translated datasets. We set a similarity threshold of 0.7 to filter out translations that deviated significantly from the original meaning. Only the retranslated sentences exceeding this threshold were retained; others were discarded. To create an expanded dataset, We combined the filtered retranslated comments with the original ones.

TABLE I
DATASET STATISTICS

Language	Number of categories	Initial train data	Dataset after augmentation and filtering	Dataset after removing duplicates
Java	7	7614	47169	36199
Python	5	1885	10170	8175
Pharo	7	1298	7781	6457

As shown in Table I, the size of the dataset increases considerably after applying augmentation and filter.

D. Preprocess

Following Al-kaswan et al. [6], we concatenate the class name and the comment sentence of the new dataset to serve as the input to the model, using them as the separator between them. We also removed hyperlinks, special characters and blank spaces from the dataset.

III. Model and Hyperparameter Selection

For our multilabel classification task, we evaluated several state-of-the-art transformer-based models, including **BERT** [13], **RoBERTa** [14], **CodeBERT [15]**, and **DistilBERT** [16] using augmented dataset. These models were chosen for their demonstrated success in natural language processing (NLP) tasks, particularly in understanding complex linguistic patterns, making them well-suited for multi-label classification. BERT [13] captures bidirectional contextual information effectively but can be computationally intensive due to its large model size. RoBERTa [14] improves over BERT with optimized pretraining techniques, but its removal of the Next

[3]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Sentence Prediction task limits performance on certain tasks requiring inter-sentence relationships. CodeBERT [15] leverages cross-domain pretraining on programming and natural languages, enhancing domain-specific tasks, but may underperform on general NLP tasks without programming context. DistilBERT [16] achieves a performance comparable to that of BERT with a reduced computational cost, but its smaller size may limit its ability to handle very complex tasks. We conducted experiments on this model, fine-tuning with different hyperparameters. We selected the **Python** dataset for this experiment, as it offered a moderate size, with fewer rows than the Java dataset but more rows than the Pharo dataset, striking a balance between data volume and computational feasibility. We split the dataset into a training set (80%) and a validation set (20%). We used Optuna[4] to find the best value of **learning rate**, **weight decay**, and **batch size**. Table II shows the comparative results of the selected models after experimenting with different hyperparameters.

RoBERTa-large achieves the highest F1 score (0.6804), demonstrating strong performance despite being trained for only 8 epochs. CodeBERT-base follows closely with an F1 score of 0.6782, outperforming other base models like BERT-base (0.6291) and DistilBERT-base (0.6368), highlighting the advantage of CodeBERT's task-specific pretraining on code data. While base models like DistilBERT-base are more resource-efficient, RoBERTa-large and CodeBERT-base stand out, with **RoBERTa-large** selected for its top performance with fewer training epochs.

TABLE II
COMPARISON OF DIFFERENT MODELS

Model	Learning rate	Weight decay	Batch size	Average f1
BERT-base	9.5265E-05	0.03536	16	0.6291
BERT-large	3.0913E-05	0.06350	8	0.6459
CodeBERT-base	9.3896E-05	0.0869	16	0.6782
DistilBERT-base	5.6237E-05	0.0601	16	0.6368
RoBERTa-base	4.6431E-05	0.0337	16	0.6432
RoBERTa-large	1.9590E-05	0.0878	32	0.6804

IV. EXPERIMENT

A. Training hyperparameters

As identified in Section 3, the best-performing model for this task is **RoBERTa-large**. While we have already experimented with various hyper-parameters for Python code comment classification, Java and Pharo need to perform further fine-tuning to determine their optimal hyper-parameters. Since we aim to fine-tune three different versions of the same model for these languages, we trained RoBERTa-base using a similar approach for Java and Pharo code comment classification.

The evaluation also considers the runtime and FLOPs of the models, but these metrics were not prioritized during hyperparameter optimization as well as model selection, as the primary goal was to maximize the F1 score and identify the model that provides the best predictions. We trained the model using hyperparameters from Table II and Table III. Training epochs were 16 for Java and 24 for Pharo and Python.

TABLE III
ROBERTA-LARGE OPTIMAL HYPERPARAMETER FOR JAVA AND PHARO

Classification Language	Learning rate	Weight decay	Batch size	Average f1
Java	2.0191E-05	0.0913	32	0.7285
Pharo	1.3269E-05	0.01618	4	0.7342

B. Implementation

We conducted our model selection and hyper-parameter tuning using the Kaggle environment[5] with dual T4 GPUs with 15GB each. The final prediction and score calculation, as per instructions from the tool competition, were carried out on Google Colab[6] using a single T4 GPU. The implementation utilized Hugging Face Transformers[7] for pre-trained models, PyTorch[8] for training and inference, and Optuna for hyperparameter tuning. Additionally, essential Python packages such as NumPy[9], Pandas[10], and Scikit-learn[11] were used for data preprocessing and evaluation.

C. Metrics

For evaluation, we used metrics outlined by the competition [12].

The metrics are defined as follows:

$$P_c = \frac{TP_c}{TP_c + FP_c}, \quad R_c = \frac{TP_c}{TP_c + FN_c}, \quad F_c = 2 \cdot \frac{P_c \cdot R_c}{P_c + R_c} \quad (1)$$

where TP_c, FP_c, and FN_c represent the true positives, false positives, and false negatives for category c, respectively.

The final submission score is calculated as follows:

$$\begin{aligned} &\text{submission\_score(model)} = 0.60 \times \text{avg. } F_1 \\ &+ 0.2 \times \max\left(0, \frac{\text{max\_avg\_runtime} - \text{measured\_avg\_runtime}}{\text{max\_avg\_runtime}}\right) \\ &+ 0.2 \times \max\left(0, \frac{\text{max\_avg\_GFLOPS} - \text{measured\_avg\_GFLOPS}}{\text{max\_avg\_GFLOPS}}\right) \end{aligned} \quad (2)$$

V. RESULTS

From Table IV, the fine-tuned RoBERTa model demonstrates a noticeable improvement over the baseline, achieving an average F1-score increase of +0.067. Additionally, the total inference time and average Giga Floating Point Operations per Second (GFLOPS) were evaluated, resulting in a submission score of 0.41. For Java, the RoBERTa model consistently

[4]https://optuna.org/

[5]https://www.kaggle.com
[6]https://colab.research.google.com
[7]https://huggingface.co/transformers/
[8]https://pytorch.org/
[9]https://numpy.org/
[10]https://pandas.pydata.org/
[11]https://scikit-learn.org/

improved across most categories, with strong gains in Expand (+0.075) and Usage (+0.037) while maintaining perfect performance in the Ownership category.

In Pharo, the most dramatic improvements were observed in Classreferences (+0.314) and Intent (+0.129), reflecting the model's capacity to adapt to these specific categories despite smaller dataset sizes. Python exhibited steady performance improvements, with notable gains in Summary (+0.116) and Expand (+0.108), indicating RoBERTa's adaptability across diverse comment types and programming paradigms.

Upon further observation, we find that larger datasets generally lead to higher F1 scores for RoBERTa (e.g., Summary with 19,901 instances achieves 0.89), while smaller datasets often result in lower performance (e.g., Rational with 1,643 instances achieves 0.29). However, exceptions such as Ownership (980 instances, F1 = 1.00) and Intent (787 instances, F1 = 0.87) suggest that well-defined and less ambiguous patterns can yield high scores even with limited data. The dataset is imbalanced across categories, which may influence model performance, and further study is required to assess its impact. A more uniformly distributed dataset could potentially enhance RoBERTa's effectiveness.

While RoBERTa excels in languages like Java and Python, it shows weaker performance in some Pharo categories, such as Key Implementation Points (F1: 0.60, Dataset: 1,006), Collaborators (F1: 0.48, Dataset: 394), and Classreferences (F1: 0.60, Dataset: 212), likely due to smaller dataset sizes. Similarly, in Python, the lowest F1 score (Development Notes: 0.35) corresponds to the smallest dataset size (903 instances), emphasizing the impact of limited data on performance.

TABLE IV
PERFORMANCE COMPARISON OF BASELINE AND ROBERTA

Language	Category	Training Dataset Instances	Baseline			RoBERTa			ΔF1
			Prec.	Rec.	F1	Prec.	Rec.	F1	
Java	Summary	19901	0.87	0.83	0.85	0.90	0.88	0.89	**+0.042**
	Ownership	980	1.00	1.00	1.00	1.00	1.00	1.00	+0.000
	Expand	2476	0.32	0.44	0.37	0.44	0.46	0.45	**+0.075**
	Usage	8609	0.91	0.82	0.86	0.92	0.88	0.90	**+0.037**
	Pointer	3019	0.74	0.94	0.83	0.81	0.95	0.87	**+0.046**
	Deprecation	409	0.82	0.60	0.69	0.82	0.60	0.69	+0.000
	Rational	1643	0.16	0.30	0.21	0.27	0.32	0.29	**+0.084**
Pharo	Key Implementation Points	1006	0.64	0.65	0.64	0.73	0.51	0.60	-0.041
	Example	2490	0.87	0.90	0.89	0.92	0.89	0.91	**+0.018**
	Responsibilities	1355	0.60	0.60	0.60	0.59	0.79	0.67	**+0.076**
	Classreferences	212	0.20	0.50	0.29	0.50	0.75	0.60	**+0.314**
	Intent	787	0.72	0.77	0.74	0.84	0.90	0.87	**+0.129**
	Keymessages	1050	0.68	0.79	0.73	0.74	0.79	0.76	**+0.033**
	Collaborators	394	0.26	0.60	0.36	0.45	0.50	0.48	**+0.113**
Python	Usage	2247	0.70	0.74	0.72	0.79	0.79	0.79	**+0.076**
	Parameters	2507	0.79	0.81	0.80	0.85	0.81	0.83	**+0.029**
	Development Notes	903	0.24	0.49	0.33	0.43	0.29	0.35	**+0.023**
	Expand	1567	0.43	0.77	0.55	0.68	0.64	0.66	**+0.108**
	Summary	1685	0.65	0.59	0.62	0.69	0.78	0.73	**+0.116**
Average			0.61	0.69	0.64	0.70	0.71	0.70	**+0.067**

VI. CONCLUSION

The study demonstrates that the fine-tuned RoBERTa model consistently outperforms baseline models in multi-label code comment classification across Java, Pharo, and Python datasets. Significant improvements were observed in F1-scores, particularly in categories requiring nuanced contextual understanding, such as Summary, Usage, and Classreferences. The integration of data augmentation through translation-retranslation and transformer-based architectures underscores the effectiveness of synthetic data in enhancing classification accuracy. This research establishes RoBERTa as a robust framework for improving code comprehension and classification, while also highlighting the potential for further optimization in handling imbalanced datasets and specific linguistic contexts.

REFERENCES

[1] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, and O. Nierstrasz, "How to identify class comment types? a multi-language approach for class comment classification," *Journal of systems and software*, vol. 181, p. 111047, 2021.

[2] L. Pascarella and A. Bacchelli, "Classifying code comments in java open-source software systems," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017.

[3] D. Steidl, B. Hummel, and E. Jürgens, "Quality analysis of source code comments," *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 83–92, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:16657129

[4] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 2007, pp. 70–79.

[5] N. L. Hai and N. D. Q. Bui, "Dopamin: Transformer-based comment classifiers through domain post-training and multi-level layer aggregation," in *Proceedings of the Third ACM/IEEE International Workshop on NL-Based Software Engineering*, ser. NLBSE '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 61–64. [Online]. Available: https://doi.org/10.1145/3643787.3648044

[6] A. Al-Kaswan, M. Izadi, and A. Van Deursen, "Stacc: Code comment classification using sentencetransformers," in *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 2023, pp. 28–31.

[7] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang, "On llms-driven synthetic data generation, curation, and evaluation: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2406.15126

[8] A. Kang, J. Y. Chen, Z. Lee-Youngzie, and S. Fu, "Synthetic data generation with llm for improved depression prediction," 2024. [Online]. Available: https://arxiv.org/abs/2411.17672

[9] B. Ding, C. Qin, R. Zhao, T. Luo, X. Li, G. Chen, W. Xia, J. Hu, A. T. Luu, and S. Joty, "Data augmentation using large language models: Data perspectives, learning paradigms and challenges," 2024. [Online]. Available: https://arxiv.org/abs/2403.02990

[10] J. Tiedemann, M. Aulamo, D. Bakshandaeva, M. Boggia, S.-A. Grönroos, T. Nieminen, A. Raganato, Y. Scherrer, R. Vázquez, and S. Virpioja, "Democratizing neural machine translation with opus-mt," *Language Resources and Evaluation*, vol. 58, no. 2, pp. 713–755, Jun. 2024. [Online]. Available: https://doi.org/10.1007/s10579-023-09704-w

[11] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. [Online]. Available: https://arxiv.org/abs/1908.10084

[12] G. Colavito, A. Al-Kaswan, N. Stulova, and P. Rani, "The nlbse'25 tool competition," in *Proceedings of The 4th International Workshop on Natural Language-based Software Engineering (NLBSE'25)*, 2025.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186.

[14] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[15] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, and L. Shou, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020, pp. 1536–1547.

[16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.