*JKAU: Comp. IT. Sci.*, **Vol. 13** No. 2, pp: 67 – 76 (2024 A.D.)
DOI: 10.4197/Comp.13-2.5

# A Multi-Label Code Comment Classifier using BERT

Zarah Shibli, Emad Albassam
Department of Computer Science, King Abdulaziz University
Jeddah, Saudi Arabia
zshibli0002@stu.kau.edu.sa, ealbassam@kau.edu.sa

**Abstract— Code comments play an essential role in software development by providing documentation, explanations, and clarifications for program logic and functionality. It is crucial to effectively classify code comments to improve software maintainability and collaboration in the face of a growing amount of code. Developers can easily identify and comprehend different code sections' purpose, behavior, and requirements by accurately classifying code comments. Although there are prior research efforts in the area of code comment classification, they are restricted to binary or multi-class classification. With this regard, this paper advances the literature in the area of code comment classification by presenting a novel approach that incorporates multi-label classification to enhance code comment classification in three programming languages: Python, Pharo, and Java. We employ BERT, a widely used language model, and achieve an F1 score of 0.64 through experimentation. Our proposed approach facilitates the understanding and managing code comments, making software development more efficient and productive. Additionally, our approach can be extended to other programming languages and serve as a foundation for further research in code comment classification.**

***Keywords— Code Comment, Classification, Natural Language Processing, Deep Learning, Software Engineering***

## I. INTRODUCTION

**C**ode comments are essential components in software development because they give additional information about the code's purpose and functionality. The advantages of having a common category for code comments include increased code understanding, maintenance, and developer collaboration.

However, maintaining and classifying code comments may be difficult, particularly in large-scale software systems. Manually classifying code comments takes effort and is prone to human mistakes [1]. To address this problem, researchers have explored machine learning algorithms to automatically classify code comments in real-time. Prior works have introduced various taxonomies that categorize code comments based on their content and purpose [2]. Several studies have investigated the trends and patterns in code commenting behavior using machine learning algorithms with satisfactory results in effectively categorizing code comments [3] [4].

Recent advances in deep learning techniques have also been used to enhance code comment classification. Pre-trained models, such as BERT, have been used to classify code comments more accurately [5] [6]. These models can capture the semantic and syntactic structures of sentences, allowing them to understand the context and meaning of sentences more effectively.

Although prior studies have demonstrated the advantages of utilizing machine learning algorithms and Bert models to induct classifiers that categorize code comments into binary [4] [5] [6] [7] [8] or multi-class [3] [9] [10] categories, to the best of our knowledge prior research has yet to specifically address the development of a multi-label classifier for code comment classification. Therefore, this study aims to bridge this gap by presenting a novel method for classifying code comments using a multi-label classifier and the BERT transformer model.

The main objective that drives the motivation behind this study is to build a model to classify code comments and evaluate them. To pursue these objectives, two research questions (RQs) have been formulated:

**RQ1:** Can we build a multi-label classification model to classify code comments?

**RQ2:** How can we evaluate the accuracy of the classifier?

The remainder of this paper has been divided as follows: Section II involves a brief overview of classification types and an understanding of the foundation of machine learning and deep learning. The related research work is highlighted in Section III to find the significant gap. Section IV describes the methodology used in the research. The results and analysis of the study are reported in Section V. Finally, Section VI highlights the findings and provides potential future research areas.

## II. Background

### *Types of Classification*

Classification is a machine learning approach to assign labels or categories to given input data [11]. Classification is an important aspect in several domains, such as data mining, computer vision, and natural language processing. There are different types of classification techniques, such as binary, multi-class [12], and multi-label [13], each serving distinct purposes. Binary classification is the task of separating data instances into two classes or categories. It has various applications, including spam detection, fraud detection, and sentiment analysis. Multi-class classification extends the binary classification problem to instances with more than two classes. The goal is to assign a single label to each data instance from a predefined set of classes. Handwritten digit recognition, text categorization, and image classification are examples of multi-class classification applications. Furthermore, there is the concept of multi-label classification, which involves assigning multiple labels to each data instance. One example is tagging a blog post, where a post can have multiple labels such as ”technology,” ”art,” and ”travel.”.

### *Machine Learning and Deep Learning*

Machine learning is a field that involves methods for automatically detecting patterns in data and then utilizing those patterns to make predictions or decisions [13].

Deep learning is a subdivision area of machine learning that has gained much attention recently due to its ability to address complex tasks such as speech analysis, image recognition, and natural language processing. Deep learning models use artificial neural networks with multiple layers and millions of parameters to construct hierarchical representations from raw data. Recurrent and Convolutional Neural Networks have been successfully adopted in deep learning architectures for solving classification tasks [14].

In recent years, innovative architectures like BERT, which stands for Bidirectional Encoder Representations from Transformers, have further enhanced the capabilities of deep learning models. Based on the transformer architecture, BERT has achieved significant breakthroughs in natural language processing tasks. By undergoing pre-training on large amounts of unlabeled text data, BERT has become a powerful tool for various classification tasks, including sentiment analysis, question answering, and named entity recognition [15]

## III. Literature Review

Many researchers have investigated the code comments classification task, considering various aspects. These studies have contributed to understanding the different classifiers and their influence on the classification of code comments. The relevant literature can be categorized into two groups applicable to our research: (i) Utilize binary classifiers for categorizing code comments. (ii) Utilize multi-class classifiers for categorizing code comments.

### *A. Utilize binary classifiers for categorizing code comments*

Much of the literature on code comment classification tasks has concentrated on binary classifiers. The primary objective of these studies has been to create binary classifiers based on the NLBSE2023 dataset. The dataset comprises code comments for Java, Python, and Pharo programming languages. One study [7] proposed a strategy for classifying code comments using the NLBSE2023 dataset. They employed various machine learning classification algorithms, including two versions of Naive Bayes (Multinomial, Bernoulli), Linear Support Vector Classifier, Decision Tree, Random Forest, K-Nearest Neighbors, Logistic Regression, and Multi-Layer Perceptron for training and evaluation. Another study [5] and [6] utilized the BERT model to develop multiple binary classifiers for the same task. Liu et al. [5] utilized the CodeT5 pre-trained language model to construct a classifier for code comment classification. Al-Kaswan et al. [6] proposed the STACC model by selecting the all-mpnet-base-v2 model for training. Research by Beck et al. [4] aimed to create a classifier for categorizing students’ Python code into sufficient and insufficient categories. Multinomial Naive Bayes and Random Forest classifier models were used. Rani et al. [8] focused on programming languages like Python, Java, and Smalltalk. Their dataset included six Java, seven Python, and seven Smalltalk projects. They used three different machine learning models: Naive Bayes, J48, and Random Forest.

### *B. Utilize multi-class classifiers for categorizing code comments*

In the area of multi-class classifiers, Niazi et al. [3] used a dataset from 70 web development projects to develop an approach for classifying code comments from student or novice programmers. Three machine learning models were employed in their experiments: Support Vector Machine Classifier, Decision Tree, and Random Forest. The research objective of Pascarella and Bacchelli [9] was to classify Java program code comments using data from six open-source projects. The dataset consisted of boolean and numeric features, approximately 15,000 comment blocks, and definitions for 16 subcategories. The following ma- chine learning classifiers were assessed: probabilistic classifiers such as Naive Bayes Multinominal and decision tree techniques such as J48 and Random Forest. Zhang et al. [10] developed a Python code comment classification approach using seven open- source GitHub projects. There are 11 categories in the dataset: metadata, summary, usage, parameters, and others. They utilized machine learning models such as Decision Tree and Naive Bayes. Table I provides an overview of the research studies.

Previous research has shown the benefits of using machine learning and Bert models to create classifiers that categorize code comments into binary or multi-class classifications. However according to our knowledge, no previous study has focused on developing a multi-label classifier for code comment classification.

TABLE I Overview of the Research Studies

| Paper | Dataset | Classifier Type | Algorithm type |
|---|---|---|---|
| Amila Indika et al. [7] | NLBSE2023 | Binary | Machine Learning |
| Liu et al. [5] | NLBSE2023 | Binary | BERT |
| Al-Kaswan et al. [6] | NLBSE2023 | Binary | BERT |
| Beck et al. [4] | Student Codes | Binary | Machine Learning |
| Niazi et al. [3] | 70 web development projects | Multi-classes | Machine Learning |
| Rani et al. [8] | 6 Java, 7 Python and 7 Smalltalk projects | Binary | Machine Learning |
| Pascarella & Bacchelli [9] | 6 Java projects | Multi-classes | Machine Learning |
| Zhang et al. [10] | 7 Python projects | Multi-classes | Machine Learning |

This paper introduces a new approach to the code comment classification task using a multi-label classifier and the BERT transformer model.

## IV. METHODOLOGY

The research follows a specific methodology involving data selection, data preparation, data splitting, model selection, training the model, and evaluating the model's performance. Figure 1 depicts an outline of our methodology.

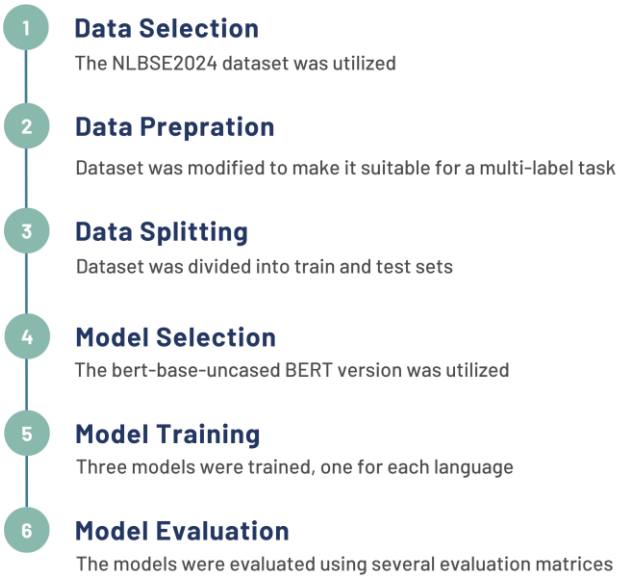


Fig I Methodology

### A. Dataset

Our approach is based on the NLBSE2024[1] dataset. It consists of comments written in Python, Java, and Pharo programming languages. The Python dataset is divided into five categories: Parameters, Expand, Usage, DevelopmentNotes, and Summary. The Pharo dataset comprises seven categories: Classreferences, Col- laborators, Example, Intent, Keyimplementationpoints, Keymes- sages, and Responsibilities. The Java dataset encompasses seven categories: Expand, Ownership, Pointer, Deprecation, Rational, Summary, and Usage.

The dataset is designed in CSV format. As shown in Table II, each row represents a sentence, and each sentence contains six columns as follows:

- **comment sentence id:** represents the unique ID of the sentence
- **class:** refers to the name of the source code file containing the sentence.
- **comment_ sentence:** is the actual string of the sentence, and it is part of a class comment that may have multiple lines.
- **partition:** determines the dataset split into training and testing, where 0 represents training instances and 1 represents testing instances.
- **instance_type:** indicates whether an instance belongs to a given category, with 0 signifying negative instances and 1 indicating positive instances.
- **category:** represents the category.

### B. Data Preparation

The NLBSE dataset initially intended for a binary classification task, underwent modifications to make it suitable for a multi-label task. Five rows originally represented each instance in the dataset, each indicating the presence or absence of the instance in a particular category. This concept is illustrated in Table III. We condensed each instance into a single row to adapt the Dataset for multi-label classification. Furthermore, we transformed each category into a column within the dataset. Therefore, for each instance, a value of 1 was assigned if it belonged to a specific category and 0 if it did not. Table IV showcases a sample from the modified dataset.

[1] https://github.com/nlbse2024/code-comment-classification/tree/main

TABLE II Sample from Original Python Dataset

| comment sentence id | class | comment_sentence | partition | instance_type | category |
|---|---|---|---|---|---|
| 2880 | FollowedBy | expression matches at the current position. | 0 | 0 | Parameters |

TABLE III Representation of One Instance from Original Python Dataset

| | comment sentence id | comment_sentence | partition | instance_type | category |
|---|---|---|---|---|---|
| **1** | 2880 | expression matches at the current position. | 0 | 0 | Parameters |
| **2** | 2880 | expression matches at the current position. | 0 | 0 | Expand |
| **3** | 2880 | expression matches at the current position. | 0 | 1 | Usage |
| **4** | 2880 | expression matches at the current position. | 0 | 1 | DevelopmentNotes |
| **5** | 2880 | expression matches at the current position. | 0 | 1 | Summary |

### *C. Data Splitting*

The original dataset was prepared for binary classification. Sometimes, a sentence could appear in the training and testing sets under different categories. Since we concentrated on multi-label tasks and were required to consider all categories for a single sentence, we did not employ "partition" for data splitting. Once the data preparation step was completed, the dataset was divided into two sets: 70\% of the data was allocated for training. The remaining 30\% was dedicated to testing. Table V shows the number of instances (N) and ratio (\%) of each category used for training and testing.

### *D. Model*

The main model utilized in this project was BERT, specifically the bert-base-uncased [2] version. Our approach involved fine-tuning BERT to classify comments. It is worth noting that this version is utilized as it does not differentiate between cases, considering "english" and "English" to be identical. The base BERT model consists of 12 transformer layers, 12 attention heads, and a hidden size of 768. Despite being a smaller version of BERT, it retains robust language comprehension capabilities, crucial for accurately interpreting code comments [15].

### *E. Proposed Model Architecture*

**RQ1:** Can we build a multi-label classification model to classify code comments?

We applied the BERT model to the NLBSE2024 dataset to answer this research question. To optimize BERT for classifying code comments, the main objective is to modify the pre-trained BERT model to accurately predict a code comment's category. The presented model consists of multiple steps as described below:

1. **Tokenization of comments:** The first step is to break down the code comments into smaller units called Word- Pieces using BERT's tokenization technique. Each Word-Piece rep- resents a token and is assigned an index from BERT's vocabulary. Unique tokens such as [CLS] (for the start of a sequence), [SEP] (to separate two sequences), and [PAD] (for padding) are added to handle comments of varying lengths.
2. **Bert Model:** To build a classifier, a multi-label classification layer, such as a fully connected layer, is added to the BERT model. This layer will produce probability scores for each label.
3. **Output:** The probability of a label surpassing a specific threshold is then examined. If it does, that label is assigned as the predicted label for the code comment.
4. **Conversion of the predicted label:** In this step, the predicted label is converted from a numeric representation to its corresponding actual category.

Figure 2 visually represents the proposed model architecture, illustrating the steps involved.

### *F. Implementation Details*

To train the BERT model, we utilized the Python development environment provided by Kaggle. For faster training, the runtime type is set to GPU. The maximum sequence length is defined as 200 tokens, while both the training batch size and validation batch size are set to 64. The desired number of training epochs is 10, the learning rate is 1.71e-05.

## V. RESULTS AND DISCUSSION

### *A. Evaluation Metrics*

**RQ2:** How can we evaluate the accuracy of the classifier?

[2] https://huggingface.co/bert-base-uncased

TABLE IV Sample from Modified Python Dataset

| comment id | comment_sentence | DevelopmentNotes | Expand | Parameters | Summary | Usage |
|---|---|---|---|---|---|---|
| 2880 | expression matches at the current position. | 1 | 0 | 0 | 1 | 1 |

TABLE V Number of Instances Used for Training and Evaluation

| Language | Category | Train | | Test | |
|---|---|---|---|---|---|
| | | Number of Instances (N) | Instances Ratio (%) | Number of Instances (N) | Instances Ratio (%) |
| Python | DevelopmentNotes | 220 | 12.30 | 92 | 11.99 |
| | Expand | 352 | 19.69 | 152 | 19.82 |
| | Parameters | 546 | 30.54 | 248 | 32.33 |
| | Summary | 318 | 17.79 | 136 | 17.73 |
| | Usage | 562 | 31.43 | 238 | 31.03 |
| | | 1788 | | 767 | |
| Pharo | Classreferences | 57 | 4.62 | 20 | 3.77 |
| | Collaborators | 87 | 7.04 | 40 | 7.55 |
| | Example | 510 | 41.30 | 238 | 44.91 |
| | Intent | 149 | 12.06 | 69 | 13.02 |
| | Keyimplementationpoints | 174 | 14.09 | 58 | 10.94 |
| | Keymessages | 220 | 17.81 | 85 | 16.04 |
| | Responsibilities | 250 | 20.24 | 86 | 16.23 |
| | | 1235 | | 530 | |
| Java | Expand | 563 | 7.62 | 265 | 8.37 |
| | Ownership | 386 | 5.22 | 164 | 5.18 |
| | Pointer | 964 | 13.05 | 406 | 12.82 |
| | Deprecation | 129 | 1.75 | 51 | 1.61 |
| | Rational | 380 | 5.14 | 148 | 4.67 |
| | Summary | 3167 | 42.87 | 1390 | 43.89 |
| | Usage | 2091 | 28.30 | 878 | 27.72 |
| | | 7388 | | 3167 | |

In order to provide an answer to this research question, various evaluation metrics such as Precision, Recall, and F1-score are used.

Precision is a performance measurement used in classification tasks to assess the accuracy of a model's positive predictions. It is calculated by dividing the total number of positive predictions (TP) by the number of actual positive predictions (TP and FP). Precision indicates the reliability of a model's positive predictions, with higher values implying a lower rate of false positives. Precision formula is defined as:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

Recall is another performance metric employed in classification tasks to evaluate a model's ability to identify all positive instances in a dataset correctly. It is derived by dividing the number of correct positive (TP) predictions by the total number of correct positives (TP) and false negatives (FN). Recall signifies the sensitivity of a model in identifying positive instances, with higher values suggesting a lower rate of false negatives. Recall formula is defined as:

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

The F1-score, a combined metric of precision and recall, evaluates a model's performance in a classification task. It takes into consideration the trade-off between recall and precision by computing the harmonic mean of both of them. The F1-score, which ranges from 0 to 1, with 1 indicating the highest performance, is typically employed when both recall and precision are important and must be assessed together. Equation 3 shows the F1-score formula.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (3)$$

### *B. Experiment Results*

We built three models, each one trained for a particular language. The threshold is a crucial component in a multi-label classification model. Our study investigates the impact of different threshold values on the resulting outcomes. Specifically, we experimented with three threshold values: 0.25, 0.5, and 0.75.

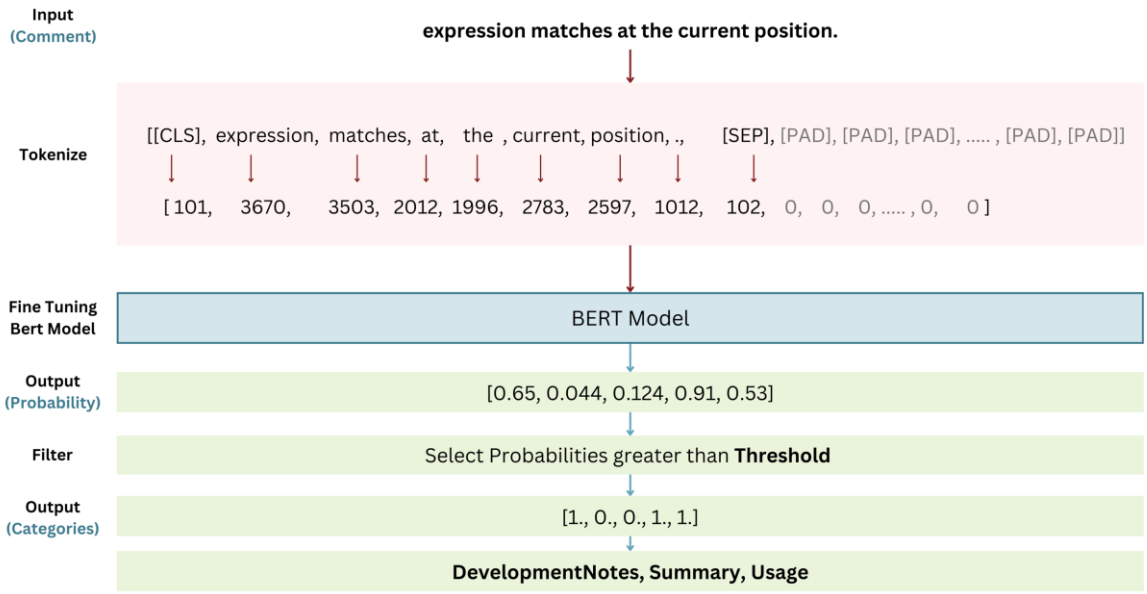


Fig II Proposed Model Architecture

The results presented in Table VI provide valuable insights into the outcomes of our model. The threshold value determines the minimum confidence required for a prediction to be classified as positive.

Figure 3 presents the model's accurate and erroneous classification examples across various programming languages, offering insights into its performance. In Python, the model effectively recognized comments summarizing code functionality, exemplified by Example 1. However, in Example 2, a comment instructing to 'Fit the model according to the given training data' was mislabeled as "DevelopmentNotes" and "Expand" instead of the expected "Summary" and "Usage." In Java, the model correctly categorized a comment displayed in Example 3, 'java.lang.Object#toString()' as "Pointer," aligning with expectations. Nevertheless, it misclassified a comment in Example 4 mentioning the '{@link java.text.Collator} class' as "Summary" rather than "Pointer," which fails to recognize its function in directing readers to specific class documentation, despite containing the @link keyword, which indicates the "Pointer" category. In Pharo, the model accurately classified a comment provided in Example 5 as "Example," but misidentified a comment in Example 6, "Add offset values to classPool," as both "Keymessages" and "Example" instead of the expected "Keyimplementationpoints," indicating a misinterpretation of its implementation instructions. These findings highlight the model's strengths in certain areas and reveal opportunities for improvement, particularly in accurately distinguishing between categories and interpreting instructional content.

### *C. Results Discussion*

#### Impact of Thresholds on Model Performance

When analyzing the results in Python, the category "Expand" achieved the same F1 score when the threshold was set to 0.25 and 0.5. On the other hand, the categories "Parameters" and "Usage" obtained a higher F1 score when the threshold was set to 0.5, showing an increase of 0.02 and 0.01, respectively, compared to the results obtained with a threshold of 0.25. The categories "DevelopmentNotes" and "Summary" achieved the best F1 score when the threshold was set to 0.25.

In Pharo, all categories achieved a high F1 score when the threshold was set to 0.25.

In Java, the categories "Ownership," "Pointer," and "Summary" obtained the same score regardless of the threshold value. The category "Deprecation" scored high with thresholds of 0.5 and 0.75. The category "Expand" achieved a high score with a threshold of 0.25. The categories "Rational" and "Usage" obtained a high score when the threshold was set to 0.5.

Overall, the F1 scores were generally higher across all languages when the threshold was set to 0.25 on average.

#### Impact of Programming Languages on Model Performance

It is important to note that the dataset utilized in our study was unbalanced. In the Python programming

TABLE VI Results

| | | Threshold = 0.25 | | | Threshold = 0.5 | | | Threshold = 0.75 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** | **Precision** | **Recall** | **F1-score** |
| **Python** | DevelopmentNotes | 0.27 | 0.25 | **0.26** | 0.37 | 0.08 | 0.13 | 0.00 | 0.00 | 0.00 |
| | Expand | 0.45 | 0.64 | **0.53** | 0.55 | 0.51 | **0.53** | 0.68 | 0.37 | 0.48 |
| | Parameters | 0.65 | 0.81 | 0.72 | 0.72 | 0.76 | **0.74** | 0.79 | 0.67 | 0.72 |
| | Summary | 0.57 | 0.57 | **0.57** | 0.64 | 0.44 | 0.52 | 0.78 | 0.39 | 0.52 |
| | Usage | 0.78 | 0.72 | 0.75 | 0.86 | 0.68 | **0.76** | 0.89 | 0.64 | 0.75 |
| | | 0.54 | 0.60 | **0.56** | 0.63 | 0.49 | 0.54 | 0.63 | 0.41 | 0.49 |
| **Pharo** | Classreferences | 0.35 | 0.30 | **0.32** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | Collaborators | 0.41 | 0.35 | **0.38** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| | Example | 0.84 | 0.87 | **0.85** | 0.91 | 0.79 | 0.84 | 0.96 | 0.73 | 0.83 |
| | Intent | 0.71 | 0.81 | **0.76** | 0.88 | 0.65 | 0.75 | 1.00 | 0.14 | 0.25 |
| | Keyimplementationpoints | 0.35 | 0.79 | **0.48** | 0.53 | 0.29 | 0.38 | 0.00 | 0.00 | 0.00 |
| | Keymessages | 0.46 | 0.85 | **0.60** | 0.60 | 0.58 | 0.59 | 1.00 | 0.25 | 0.40 |
| | Responsibilities | 0.49 | 0.73 | **0.59** | 0.77 | 0.47 | 0.58 | 0.91 | 0.12 | 0.21 |
| | | 0.52 | 0.67 | **0.57** | 0.53 | 0.40 | 0.45 | 0.55 | 0.18 | 0.24 |
| **Java** | Expand | 0.63 | 0.52 | **0.57** | 0.68 | 0.45 | 0.54 | 0.73 | 0.32 | 0.45 |
| | Ownership | 0.99 | 0.99 | **0.99** | 0.99 | 0.99 | **0.99** | 0.99 | 0.99 | **0.99** |
| | Pointer | 0.90 | 0.90 | **0.90** | 0.93 | 0.87 | **0.90** | 0.94 | 0.86 | **0.90** |
| | Deprecation | 0.82 | 0.78 | 0.80 | 0.85 | 0.78 | **0.82** | 0.97 | 0.71 | **0.82** |
| | Rational | 0.51 | 0.50 | 0.50 | 0.62 | 0.43 | **0.51** | 0.72 | 0.35 | 0.47 |
| | Summary | 0.90 | 0.95 | **0.93** | 0.91 | 0.94 | **0.93** | 0.92 | 0.93 | **0.93** |
| | Usage | 0.87 | 0.93 | 0.90 | 0.90 | 0.92 | **0.91** | 0.92 | 0.89 | 0.90 |
| | | 0.80 | 0.80 | **0.80** | 0.84 | 0.77 | **0.80** | 0.89 | 0.72 | 0.78 |
| **Average** | | 0.62 | 0.69 | **0.64** | 0.67 | 0.55 | 0.60 | 0.69 | 0.44 | 0.50 |

language, the DevelopmentNote category had fewer instances. Additionally, the Parameters category exhibited repeated the unique keyword "default."

In the Pharo programming language, the Classreferences, Collaborators, and Keyimplementationpoints categories had a relatively smaller number of instances compared to other categories. However, the Example category displayed a significant number of instances, some of which included the keyword "example"

In the case of Java, high f1-scores were achieved in the Ownership, Pointer, Summary, and Usage categories. The Ownership category exhibited several distinct keywords such as implementation, contributors, initial, api, and @author. The Pointer category frequently reiterated words like @link, @see, and @code in the comments. The Usage category encompassed the words @param and @return. Finally, the Deprecation category featured unique words like @since and @deprecated.

Overall, the findings of this study shed light on various aspects of the model's performance and provide valuable insights into the characteristics of different programming languages.

Considering all categories, our model attains an average F1 score of 0.64. The proposed model's performance is moderate, with specific categories performing better than others.

**Impact of Bias on Model Results**

The performance of BERT models varies when classifying code comments in Python, Pharo, and Java. The study indicated that the imbalance in the NLBSE dataset reported by \cite{indika2023performance} resulted in data bias issues that affected the model's performance. In order to address potential biases between programming languages, separate models were developed for each language. In Python, categories like "DevelopmentNotes" have low F1 scores due to insufficient data, while "Expand," "Parameters," "Summary," and "Usage" perform better but show drops at higher thresholds, indicating model uncertainty or overfitting. In Pharo, difficulties arise with "Classreferences" and "Collaborators," and performance declines at higher thresholds, pointing to data imbalance issues. In contrast, Java demonstrated consistently high performance, particularly in categories such as "Ownership" and "Pointer," indicating robust training data. To address biases and enhance performance, it is recommended to balance the datasets, adjust thresholds, and improve the model.

### *D. Comparison with Prior Research*

We compared our results with the results of prior approaches that utilize the Bert and traditional machine learning models on the same dataset used in this paper for a consistent and fair comparison. Al-Kaswan et al. [6] achieved a higher average F1 score of 0.74 using a binary classifier for

| Language | | Comment Text | Predicted Category | Expected Category | Reason |
|---|---|---|---|---|---|
| Python | 1 | This is an implementation that uses the result of the previous model to speed up computations along the set of solutions. | Summary | Summary | |
| | 2 | Fit the model according to the given training data. | DevelopmentNotes, and Expand | Summary, and Usage | This comment summarizes purpose or functionality of code. |
| Java | 3 | @see java.lang.Object#toString() | Pointer | Pointer | |
| | 4 | The {@link java.text.Collator} class | Summary | Pointer | This comment guides reader to specific class documentation. |
| Pharo | 5 | For example, let's consider a structure that models a fraction, i.e., | Example | Example | |
| | 6 | Add offset values to classPool. | Keymessages, Example | Keyimplementationpoints | This comment provides crucial implementation instructions. |

Fig III . Examples of Model Predictions and Expected Categories for Code Comments

code comment classification. Their model better distinguished between positive and negative comments. Furthermore, we also compared our results with another related work [5] that employed a binary classifier with the Bert model for code comment classification. They achieved an average F1 score of 0.66. In another study [7], the authors used a Linear Support Vector Machine and obtained an F1 score of 0.55. However, it is important to note that their models focused on binary classification, categorizing each comment into one of two classes. In contrast, our approach used multi-label classification, where a comment can simultaneously be associated with multiple labels. This different perspective may explain the slight decrease in the observed classification performance.

Table VII displays the performance of our proposed model compared to the related works. These comparisons indicate that the performance of the Bert model in code comment classification can vary depending on different factors, including the type of classifier that is selected, whether it is binary or multi-label, the number of instances within each category, and whether the programming language the code is written in follows to a particular structure. Additionally, Bert demonstrated competence in comparison to traditional machine learning approaches in the context of code comment classification.

TABLE VII Performance of the Proposed Model Compared to The Related Works

| | Average F1-Score | Is Multi-labels? |
|---|---|---|
| Liu et al. [5] | 0.66 | No |
| Al-Kaswan et al. [6] | 0.74 | No |
| Amila Indika et al [7] | 0.55 | No |
| Proposed model | 0.64 | Yes |

## VI. CONCLUSION

In this paper, we successfully implemented a code comment classification system using BERT. Unlike prior works which investigated binary and multi-class classifications, we consider that a single comment can be classified into multiple labels. Therefore, we implemented a multi-label comment classifier in which a BERT model is trained on a labeled dataset of code comments and achieved satisfactory results regarding multi-label classification. Our model accurately classified code comments into multiple categories, which would assist developers in understanding and organizing code comments more efficiently. Our findings show that the model's results are affected by the underlying programming language syntax for comments. In particular, languages such as Java in which distinct comment tags such as @author, @see, and @code are used have achieved higher results compared to other languages.

In terms of directions for future research, we can explore several aspects to improve our code comment classification model. Currently, we have focused on code comments in three programming languages. However, extending the model to work with code comments written in multiple languages would be beneficial.

BERT has released various versions with different architectures and pre-training methods. Experimenting with these different versions may lead to improvements in performance and accuracy. We can compare the performance of different versions to identify the most suitable one for our code comment classification task.

Overall, by exploring different languages and trying out various versions of BERT, we can continue to enhance the accuracy and applicability of our code comment classification system.

## REFERENCES


[1] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, et al., "Automating code review activities by large-scale pre-training," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022.

[2] M. Kosti´c, A. Srbljanovi´c, V. Batanovi´c, and B. Nikoli´c,, "Code comment classification taxonomies," 2022.

[3] T. Niazi, T. Das, G. Ahmed, S. M.Waqas, S. Khan, S. Khan, A. A. Abdelatif, and S. Wasi, "“Investigating novice developers’ code commenting trends using machine learning techniques," *Algorithms*, vol. 16, no. 1, p. 53, 2023.

[4] M. J. M.-A. Phyllis Beck and C. Archibald, "“An initial exploration of machine learning techniques to classify source code comments in real-time," in *2019 ASEE Annual Conference & Exposition*, 2019.

[5] Y. Li, H. Wang, H. Zhang, and S. H. Tan, "Classifying code comments via pre-trained programming language model," pp. 24-27, 2023.

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

[7] A. Indika, P. Washington, and A. Peruma, "Performance comparison of binary machine learning classifiers in identifying code comment types: An exploratory study," *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, p. 20–23, 2023.

[8] P. Rani, S. Panichella, M. Leuenberger, A. D. Sorbo, and O. Nierstrasz, "How to identify class comment types? A multi-language approach for class comment classification," *CoRR*, vol. abs/2107.04521, 2021.

[9] 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)*, 2017.

[10] J. Zhang, L. Xu, and Y. Li, "Classifying python code comments based on supervised learning," in *International Conference on Web Information Systems and Applications*, 2018.

[11] D. Dietrich, B. Heller, B.Yang, et al., Data science & big data analytics: discovering, analyzing, visualizing and presenting data, Wiley, 2015.

[12] P.-N. Tan, M. Steinbach, A. Karpatne, and V. Kumar, Introduction to Data Mining, 2nd ed., Pearson, 2018.

[13] K. P. Murphy, Machine learning: A probabilistic perspective, The MIT Press, 2012.

[14] Y. LeCun,Y. Bengio, and G. Hinton, "Deep learning," vol. 521, nature, 2015, p. 436–444.

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

# تصنيف التعليقات البرمجية متعددة التسميات باستخدام المحولات المدربة مسبقًا

**زرعه شبلي، عماد البسام**

قسم علوم الحاسبات، كلية الحاسبات وتقنية المعلومات، جامعة الملك عبد العزيز، جدة، المملكة العربية السعودية

zshibli0002@stu.kau.edu.sa, ealbassam@kau.edu.sa

***المستخلص***. تعد التعليقات البرمجية من الأسس في تطوير البرمجيات، ومع التزايد الكبير في عدد الأكواد البرمجية تتجلى أهمية تصنيف التعليقات البرمجية في تسهيل صيانة البرمجيات والتي تساعد المطورين من فهم الأكواد البرمجية بدقة وسهولة. تقدم هذه الدراسة منهجًا يستخدم التسميات المتعددة و نموذج المحولات المدربة مسبقًا لتصنيف التعليقات البرمجية في ثلاث لغات برمجة: بايثون، فارو، جافا. وقد أظهر النموذج المقترح نسبة تبلغ 0.64. كما يعمل النهج المقترح على تبسيط فهم وإدارة التعليقات البرمجية لتعزز كفاءة وإنتاجية تطوير البرمجيات. بالإضافة إلى ذلك، يمكن توسيع النهج المقترح ليشمل لغات البرمجة الأخرى ويشكل الأساس للأبحاث المستقبلية حول تصنيف التعليقات البرمجية.

***الكلمات المفتاحية***ـ التعليقات البرمجية، التصنيف، معالجة اللغات الطبيعية، التعلم العميق، هندسة البرمجيات