IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 28 June 2024, accepted 14 July 2024, date of publication 18 July 2024, date of current version 29 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3430558

RESEARCH ARTICLE

Multi-Label Code Error Classification Using CodeT5 and ML-KNN

MD. FAIZUL IBNE AMIN[1], (Graduate Student Member, IEEE), ATSUSHI SHIRAFUJI[1], (Member, IEEE), MD. MOSTAFIZER RAHMAN[2], AND YUTAKA WATANOBE[1], (Member, IEEE)

[1]Graduate Department of Computer and Information Systems, The University of Aizu, Aizuwakamatsu, Fukushima 965-8580, Japan
[2]Information and Communication Technology Cell, Dhaka University of Engineering and Technology, Gazipur 1707, Bangladesh

Corresponding authors: Md. Faizul Ibne Amin (aminfaizul007@gmail.com) and Md. Mostafizer Rahman (mostafiz26@gmail.com)

This work was supported by Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 23H03508.

ABSTRACT Programming is an essential skill in computer science and in a wide range of engineering-related disciplines. However, occurring errors, often referred to as ''bugs'' in code, can indeed be challenging to identify and rectify, both for students who are learning to program and for experienced professionals. These errors can lead to unexpected behaviors in programming. Understanding, finding, and effectively dealing with errors is an integral part of programming learning as well as software development. To classify the errors, we propose a multi-label error classification of source code for dealing with programming data by using the ML-KNN classifier with CodeT5 embeddings. In addition, several deep neural network (DNN) models, including GRU, LSTM, BiLSTM, and BiLSTM-A (attention mechanism) are also employed as baseline models to classify the errors. We trained all the models by using a large-scale dataset (original error labels) as well as modified datasets (summarized error labels) of the source code. The average classification accuracy of the proposed model is 95.91% and 84.77% for the original and summarized error-labeled datasets, respectively. The exact match accuracy is 22.57% and 27.22% respectively for the original and summarized error-labeled datasets. The comprehensive experimental results of the proposed approach are promising for multi-label error classification over the baseline models. Moreover, the findings derived from the proposed approach and data-driven analytical results hold significant promise for error classification, programming education, and related research endeavors.

INDEX TERMS CodeT5, data analysis, educational big data, error classification, learning analytics, ML-KNN, multi-label classification, programming learning, software engineering.

I. INTRODUCTION

Practical-based learning plays a significant role in enriching logical and innovative thinking [1]. Computer programming stands as a prime example of practical-based learning, particularly in the field of computer science. Programming education serves as the foundation for artificial intelligence (AI), modern applications, numerical analysis, and data analysis. To motivate and inspire students and programmers to engage in programming learning, many online learning platforms and e-learning systems are being employed. Students can utilize these platforms to enrich their learning experiences and study more effectively, ultimately helping them achieve their desired goals. In this context, Online Judge (OJ) systems [2], [3], play an important role in enhancing programming learning and practice opportunities alongside traditional classroom-based instruction.

OJ systems function as supplementary resources that complement traditional classroom-based programming education [4]. In contemporary education, OJs have become integral tools for numerous educational institutions that offer courses in computing, programming, and software engineering [5]. Several universities have also developed Automated Program Assessment (APA) systems for programming courses, aiming to enhance the speed and effectiveness of students' learning [6], [7]. A few examples

The associate editor coordinating the review of this manuscript and approving it for publication was Loris Belcastro.

of OJ systems, such as Aizu Online Judge (AOJ) [8], UVa [9], URI [10], Codeforces [11], and Jutge.org [12], are widely employed as academic tools. OJ systems are not only beneficial for programming learning but also have broad applications in various domains [13]. As a result, OJ systems have generated extensive archives of problem-solving data, including solution codes, scores, and logs which serve as valuable resources in programming education research.

Programmers, especially novices, often encounter challenges in understanding errors in their written codes. Identifying errors and their types in the solution code poses a challenging task for programmers, teachers, and instructors. Programmers often invest considerable time and effort in error identification within solution codes, a task that can be both time-consuming and labor-intensive. Analyzing the common and frequent errors made by programmers [14], is a valuable endeavor, contributing to a better understanding of the challenges they face and providing insights for improving programming education. Source code typically contains various error types, including syntax, semantics, communication, calculation, and logic errors. Conventional compilers, while useful, fall short of comprehensively assessing all source code errors [15]. Even with the assistance of traditional compilers, students and professional programmers often find it challenging to identify logic errors in the source code.

For instance, consider the scenario involving a logic error: although the code compiles and runs, it yields incorrect output because it contains errors. During compilation and execution, a program might terminate due to a compilation or runtime error. However, codes that contain logical errors can still be compiled and run successfully, yet produce unexpected results. Visually, a code with logical errors may appear correct, but from the perspective of logical reasoning, it is indeed erroneous.

Figure 1 presents a motivational example of error classification. The leftmost block contains the source code, which exhibits errors such as the incorrect arithmetic operator (''−'' instead of ''+'') and the incorrect variable usage (''c'' instead of ''b'') in the third line. Based on the error classification, the rightmost block displays the output section with the corresponding error labels of the code.

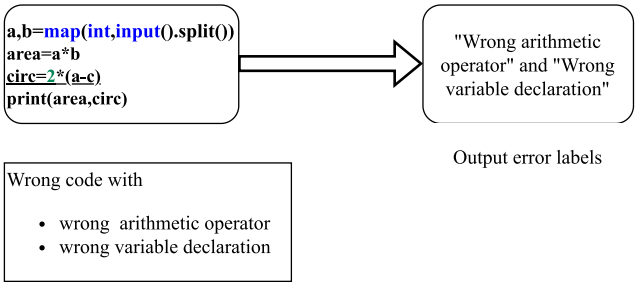


FIGURE 1. Motivational example of the error classification.

In recent times, multi-label classification has garnered significant attention and holds promise for addressing a variety of problems. In machine learning, multi-label classification presents a variation in the classification problem, where each instance can be assigned multiple nonexclusive labels. This contrasts with multi-class classification, where instances are assigned precisely one class from a set of two or more classes. Multi-label classification allows instances to be assigned to multiple classes without restrictions on the number of labels. The objective of multi-label classification is to find a model that maps inputs, represented as x, to binary vectors y, where each element (label) in the vector y is assigned a value of 0 or 1. Unlike binary or multi-class classification, where the goal is to predict only one output label, multi-label classification aims to predict data with as many applicable labels as possible based on the input data. The output in this scenario can range from one label to the maximum number of available labels.

The labels predicted by the model often exhibit interconnected relationships [16]. In NLP, multi-label text classification [17], stands as one of the fundamental tasks including diverse applications [18]. However, the simultaneous involvement of numerous classes in multi-label data introduces a level of ambiguity because of its association with multiple classes simultaneously. Consequently, multi-label data are inherently complex and, require efficient handling to navigate its inherent ambiguity. Diverse approaches have been employed to address multi-label data, including the transformation of data [19], problem adaption [20], and ensemble techniques [21]. Dealing with this type of data presents numerous complexities for classification, such as complex decision spaces and correlated classes. Researchers address these challenges from diverse perspectives to mitigate associated shortcomings.

Assisting programmers in effectively assessing, categorizing, and classifying source code errors has emerged as a pivotal research focus in software engineering and programming education. We have been inspired by the need to identify common errors made by programmers, aiming to contribute to programming learning. The primary objective of this study is to classify these prevalent errors. For the error classification task, this study utilizes a multi-label classification method. Deep Learning (DL) models have been utilized to classify multi-label data derived from real-world programming problem-solving datasets. Following data preprocessing, we employed the CodeT5 transformer for dataset embedding, enabling the ML-KNN model to perform the classification task. Additionally, we trained GRU, LSTM, BiLSTM, and BiLSTM-A on the processed data for multi-label classification purposes. Our primary focus was on classifying errors, to provide valuable insights for programmers and enhance programming learning. The key contributions and highlights of this study are summarized as follows:

- We propose a multi-label classification model to classify errors in program code. The ML-KNN classifier is used for our proposed approach. In this case, the transformer model CodeT5 is leveraged for the embedding. The average accuracy for error classification achieved by the proposed model is 95.91% and 84.77% for the original

and summarized error-labeled datasets, respectively. The exact match accuracy stands at 22.57% and 27.22%.
- Our proposed model outperformed the baseline models in terms of average accuracy, exact match accuracy, and other evaluation metrics including precision, recall, and F1 score.
- To highlight the effectiveness of the proposed approach, experiments were conducted using a real-world problem-solving dataset. The results indicate that error classification has significant potential for enhancing the skills of programmers and programming learning.

The rest of this paper follows a structured outline as follows. Section II presents some recent related studies and theoretical information on the multi-label classification related to this study is presented in Section III as baseline models. Section IV proposes the approach for the multi-label error classification. Section V presents the experimental settings, especially the dataset preparation and evaluation metrics. Section VI details the experimental results including implementation and evaluation. Section VII presents the discussion and result analysis. Section VIII encapsulates the conclusion and outlines future avenues of work.

II. RELATED WORK

This section highlights recent research relevant to multi-label classification, NLP, program code classification, and programming-related data analysis.

A. PROGRAMMING DATA ANALYSIS AND CODE CLASSIFICATION

The authors [4], primarily focused on programming education, employing the educational data mining approach. They utilize machine learning techniques to extract valuable insights from actual problem-solving data. In another research [1], the authors addressed the core objective which lies in investigating the influence of practical skills on academic performance. As a programming problem can be approached using various languages, identifying the problem solely from the source code becomes challenging. Therefore, there is a need for a classification model to assist programmers in identifying problems developed in Multi-Programming Languages (MPLs). To address this gap, the authors [22], proposed a stacked BiLSTM model designed to classify the source codes developed in MPLs. However, in this study, there are not so explicit statements to address long source code.

In research [23], the authors emphasized the importance of algorithmic thinking and the proper selection of algorithms to enhance software engineering, computational performance, and programming education. They introduced a program code classification model based on a CNN designed to classify code based on algorithms. However, The study does not sufficiently address the model's generalizability across different programming languages or coding styles and it also lacks comparisons with the latest methods in the rapidly evolving field of machine learning. Moreover, concerns about overfitting and the interpretability of the model's decisions are not thoroughly addressed.

The authors [14], delved into an intriguing topic concerning error classification to analyze the differences in frequent errors. In their research, they addressed the challenge of finding and rectifying errors that can be time-consuming for both novice and expert programmers. They have been motivated by this issue and employed a rule-based error classification tool to categorize errors in code pairs, comprising both incorrect and correct programs. The categorized errors are then utilized to scrutinize the distinctions in frequent errors between novice and expert programmers. While rule-based systems are straightforward and interpretable, they may lack the flexibility and adaptability of machine learning models. A programming problem can be solved by using various programming languages, leading to a huge multilingual solution code. Code identification from this vast archive poses a difficult task [24]. The authors have been motivated by this issue and presented a novel classification model based on problem names and algorithms. However, the dependency on intensive hyperparameter optimization could hinder adaptability and quick deployment, and the model's deep learning architecture inherently lacks interpretability, which could be a significant drawback in scenarios where understanding decision-making processes is crucial.

Moreover, several studies have introduced data analysis and machine learning-based models aimed at bolstering programming education [15], [25], [26], [27], [28], [29], [30]. These models are crafted to elevate the learning experience and improve educational outcomes in programming learning.

B. MULTI-LABEL CLASSIFICATION

The article [31], introduces a binary tree of classifiers for preserving label dependencies and addressing the class imbalance. The overarching objective of the proposed work is to construct a decision tree-based model that leverages the inherent label correlations in the data, enabling efficient machine learning classification while mitigating issues related to class imbalance. In the study [16], the authors introduced a multi-label guided network designed to guide document representation with multi-label semantic information. Additionally, to enhance the original label predictions in downstream tasks, they leveraged correlation knowledge.

Searching for hardware configurations through online embedded code can be a time-consuming process and may not ensure an optimal solution. To tackle this problem, the research outlined in [32], introduced an embedded code classifier aimed at aiding programmers in locating the most effective code snippets with accurate tags. The study involves the development of a tag-correlated multi-label machine learning model tailored for the embedded code dataset. Students' performance prediction in higher education presents notable challenges in crafting accurate and robust diagnostic models. With this aim, the study [33], focused on

two key aspects: first, they created a hybrid regression model to enhance the accuracy of predicting student performance. Subsequently, they introduced an optimized multi-label classifier designed to forecast qualitative values representing the influence of different factors linked to student performance.

In multi-label classification, where labels are often interdependent, harnessing label dependencies can markedly enhance performance. In research [34], the authors introduced a novel algorithm, Ml-Forest, which learns an ensemble of hierarchical multi-label classifier trees to unveil intrinsic label dependencies. In the study [18], the authors delved into the realm of multi-label classification algorithms based on semi-supervised learning. The study begins with a comprehensive review of supervised learning classification algorithms, taking into account both label non-correlation and label correlation. Subsequently, the authors explored semi-supervised learning classification algorithms and categorized them into two main groups: inductive and transductive methods.

In large-scale multi-label classification, the application of SVM is severely restricted by excessive time complexity. Consequently, the study [35], introduced a multi-label SVM classification algorithm, termed AEDC-MLSVM. This hybrid approach combines the approximate extreme points method with the divide-and-conquer strategy. In the study [36], the authors proposed a novel approach for learning from multi-label data, leveraging label-specific features to enhance the performance of classification models. Specifically, they introduced a wrapped learning approach that integrates label-specific feature generation with classification model induction in a simultaneous manner. Some more studies related to multi-label classification are [37], [38], [39], [40], [41], [42], where various methods have been discussed.

C. TEXT CLASSIFICATION

Conventional approaches to multi-label text classification, particularly those employing deep learning techniques, have demonstrated impressive outcomes. However, the conventional approach treats labels as independent entities, overlooking the relationships between them. The study [43], presented a new hierarchical graph transformer-based model for tackling large-scale multi-label text classification, aiming to overcome this challenge. In the study [44], the authors addressed a multi-label text classification task by highlighting the significance of explicitly incorporating label semantics. They introduced a hybrid neural network model that aims to simultaneously harness both label semantics and detailed text information. To accomplish this, they employed the pre-trained BERT model to generate context-aware representations of documents, enabling a more nuanced comprehension of the text content.

For multi-label text classification, many existing deep learning models focus on either non-consecutive or long-distance semantics. However, the coherent integration of these aspects remains under-explored. In research [45], the authors introduced a novel hierarchical taxonomy-aware and attentional graph capsule recurrent CNNs for large-scale multi-label text classification. In research [46], the authors' objective was to employ NLP techniques on articles sourced from peer-reviewed journals. The article compares the performance of multi-label text classification models using datasets with different characteristics.

D. LANGUAGE MODELS FOR CODE

In research [47], the authors emphasized the significance of programming skills in computer science and engineering-related disciplines. However, it acknowledges the inherent difficulty in crafting source code, particularly in identifying logical errors. To tackle this challenge, the authors introduced a language model for evaluating source codes, leveraging a BiLSTM neural network. Moreover, different programming environments and more complex code structures have not been thoroughly demonstrated, which may restrict its practical utility in real-world applications. In research [48], the authors introduced CodeT5, a unified pre-trained encoder-decoder transformer model designed to enhance code semantics. They propose a novel identifier-aware pre-training task, allowing the model to distinguish code tokens as identifiers and reconstruct them when masked. In research [49], the authors suggested an approach for recommending the correct program with minimal repair edits by employing CodeT5. They employed a fine-tuned pre-trained CodeT5 model by utilizing correct and incorrect code pairs.

In research [50], the authors introduced a novel type inference method that views type prediction as a code-infilling task, leveraging CodeT5. Their method employed static analysis to generate dynamic contexts for each code element. Additionally, they proposed an iterative decoding scheme that integrates previous types of predictions into the model's input context. In research [51], the authors introduced CodeBERT (a transformer-based neural architecture), designed for processing both programming language (PL) and natural language (NL). Moreover, other related research on the language models for code is presented here [52], [53], [54].

E. ML-KNN BASED STUDY

In multi-label learning, the training set consists of instances that are, linked with a set of labels. By analyzing the training instances with known label sets, the objective is to predict the label sets of unseen instances. In research [55], the authors introduced a multi-label lazy learning approach named ML-KNN, derived from the traditional KNN algorithm. Multi-label classification poses a complex and critical challenge in the domains of NLP and text mining. The research [56], aims to address these issues and deliver a standardized solution for Bengali text. The study utilizes a substantial dataset with unique labels, employing a supervised model approach that

incorporates the ML-KNN algorithm and neural network. Additionally, the Count vectorizer is employed for word embedding features. Moreover, some other research related to ML-KNN is presented here [57], [58], [59].

In our reviewed code classification-related research, we found some significant drawbacks such as (*i*) most of the study performed the classification task in either conventional ML or RNN models, (*ii*) there is a lack of discussion on how to address longer source code, (*iii*) there is no sufficient information on how to address the model's generalizability and interoperability, (*iv*) and there is a lack of explanation on how to tackle complex code structures and overfitting problems during training. Additionally, ML and RNN models have some shortcomings due to their algorithmic and structural limitations [24]. Individually, the performance of ML or RNNs is not efficient for multi-label classification tasks.

In our study, we conducted multi-label error classification to classify errors made by programmers by using ML-KNN with CodeT5. To the best of our knowledge, employing multi-label error classification for program code by this approach is distinct.

Our proposed approach integrates the transformer-based CodeT5 and ML-KNN models, effectively addressing the limitations of conventional methods. CodeT5 is a pre-trained large language model with parameters ranging from 220 million (base) to 770 million (large) [48], enabling it to handle long sequences more effectively than traditional models and thus improving classification performance. Additionally, CodeT5 is specifically optimized for code-related tasks, making it more scalable and suitable for our purposes. This hybrid approach not only remedies the shortcomings of previous models but also excels in multi-label classification tasks, surpassing conventional methods. The structural innovations in our method further set it apart from other approaches, offering a unique solution to complex multi-label classification challenges.

Another factor contributing to the uniqueness of our proposed model is the datasets used in our experiments, which are new, distinctive, and sourced from real-world programming submissions. Additionally, our approach includes an innovative feature: labeling errors in the erroneous code based on various contexts. This capability allows for precise label predictions, aiding programmers in understanding the context and nature of the errors more effectively. However, since our dataset is new and has not yet been used in other studies, presenting a direct comparison is challenging. Instead, we conducted experiments using several RNN models as baseline comparisons.

III. BASELINE MODELS

This section offers a brief introduction to baseline models that have been used in this study. Consequently, the mathematical representations of the GRU, LSTM, BiLSTM, and BiLSTM-A for sequential language modeling tasks are presented. The overview of the models is described below:

A. GRU

GRU is a type of recurrent neural network (RNN) architecture designed to address some of the limitations of traditional RNNs, such as difficulties in learning long-term dependencies and it was introduced by Cho et al. [60]. The GRU is defined by the following equations:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{1}$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2}$$
$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t]) \tag{3}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4}$$

where, z_t and r_t denote the update gate and reset gate, $\tilde{h}_t$ is the current memory content, and h_t represents the final hidden state. x_t represents the input at time step t. W_z, W_r, and W_h are the weight matrices that are learned during training. σ is the sigmoid activation function. *tanh* is the hyperbolic tangent activation function. $\odot$ is for element-wise multiplication. The final hidden state is computed by blending the previous hidden state with the new memory content based on the update gate.

B. LSTM

LSTM [61], networks are a type of RNN designed to overcome the vanishing gradient problem. The equations for an LSTM cell at a single time step are as follows. Let x_t be the input at time step t, h_{t-1} and c_{t-1} indicate the hidden state and cell state at time step $(t-1)$. The LSTM cell involves several gates and operations as follows:

$$f_t = \sigma(W_{if} x_t + W_{hf} h_{t-1} + b_f) \tag{5}$$
$$i_t = \sigma(W_{ii} x_t + W_{hi} h_{t-1} + b_i) \tag{6}$$
$$\tilde{c_t} = tanh(W_{ig} x_t + W_{hg} h_{t-1} + b_g) \tag{7}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{8}$$
$$o_t = \sigma(W_{io} x_t + W_{ho} h_{t-1} + b_o) \tag{9}$$
$$h_t = o_t \odot \tanh(c_t) \tag{10}$$

where, f_t, i_t, $\tilde{c}_t$, c_t, o_t, h_t are the forget gate, input gate, cell state update, cell state, output gate, and hidden state, respectively. In these equations, W_{if}, W_{hf}, W_{ii}, W_{hi}, W_{ig}, W_{hg}, W_{io}, and W_{ho} are weight matrices, and b_f, b_i, b_g, and b_o are the bias vectors. The σ and *tanh* represent the sigmoid and hyperbolic tangent activation functions, respectively. $\odot$ denotes element-wise multiplication.

C. BILSTM

BiLSTM is a type of RNN architecture that processes input sequences in both forward and backward directions, allowing it to capture contextual information from both preceding and following elements in the sequence. Let's consider a single-time step in a BiLSTM. The forward LSTM equations can be represented as follows:

$$i_t = \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}) \tag{11}$$
$$f_t = \sigma(W_{if} x_t + b_{if} + W_{hf} h_{t-1} + b_{hf}) \tag{12}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \quad (13)$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \quad (14)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (15)$$
$$h_t = o_t \odot \tanh(c_t) \quad (16)$$

The backward LSTM equations can be represented as follows:

$$i'_t = \sigma(W'_{ii}x_t + b'_{ii} + W'_{hi}h_{t+1} + b'_{hi}) \quad (17)$$
$$f'_t = \sigma(W'_{if}x_t + b'_{if} + W'_{hf}h_{t+1} + b'_{hf}) \quad (18)$$
$$g'_t = \tanh(W'_{ig}x_t + b'_{ig} + W'_{hg}h_{t+1} + b'_{hg}) \quad (19)$$
$$o'_t = \sigma(W'_{io}x_t + b'_{io} + W'_{ho}h_{t+1} + b'_{ho}) \quad (20)$$
$$c'_t = f'_t \odot c'_{t+1} + i'_t \odot g'_t \quad (21)$$
$$h'_t = o'_t \odot \tanh(c'_t) \quad (22)$$

where x_t, h_t, and c_t represent the input, hidden state, and cell state at time step t. σ is the sigmoid activation function, and $\odot$ represents element-wise multiplication. Finally, the network combines the output of both forward and backward LSTMs at each time step and the equation is as follows:

Assuming h_t and h'_t represent the hidden state at time t in the forward and backward LSTM, the hidden state in the BiLSTM is given by the concatenation of those two hidden states:

$$h_t^{\text{bi}} = [h_t; h'_t] \quad (23)$$

where $[h_t; h'_t]$ denotes the concatenation of the forward and backward hidden states. In a BiLSTM, the final output at each time step is often obtained by applying a fully connected layer or other relevant operations on the concatenated hidden state h_t^{bi}.

D. BILSTM-A

BiLSTM-A is a model that combines the capabilities of bidirectional LSTMs for sequence processing with an attention mechanism to focus on specific parts of the input sequence when making predictions.

Here's a conceptual overview of the BiLSTM-A: Assuming we have a sequence of input vectors $x_1, x_2, \ldots, x_t$ and the hidden states from both the forward and backward LSTMs h_t and h'_t respectively at time t, and α_t represents the attention weights, the hidden state with attention (h_t^{att}) is computed as follows:

$$e_t = v_a^T \tanh(W_a[h_t; h'_t] + b_a) \quad (24)$$
$$\alpha_t = \text{softmax}(e_t) \quad (25)$$
$$c_t = \sum_{i=1}^{t} \alpha_i [h_i; h'_i] \quad (26)$$
$$h_t^{\text{att}} = o_t \odot \tanh(c_t) \quad (27)$$

where, e_t, stands for attention scores, α_t is for attention weights, c_t is for context vector, and h_t^{att} is for final hidden state with attention. W_a, v_a, b_a are learnable parameters for the attention mechanism. *tanh* is the hyperbolic tangent function. *Softmax* is the softmax function applied element-wise. $\odot$ represents element-wise multiplication. o_t is the output gate activation from the BiLSTM.

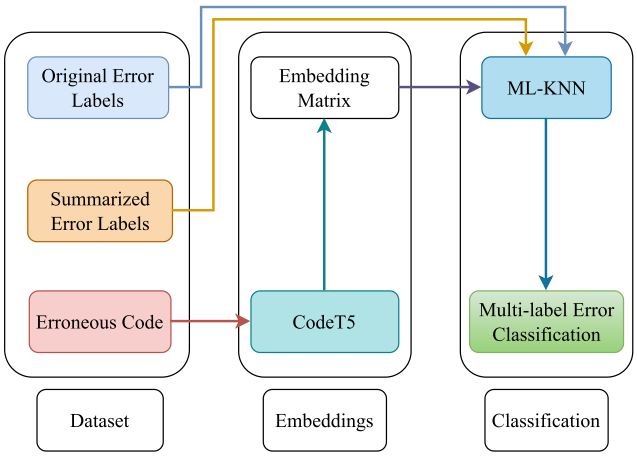


FIGURE 2. Proposed approach for multi-label error code classification with CodeT5 and ML-KNN.

IV. MULTI-LABEL ERROR CLASSIFICATION

In multi-label classification, the approach is better suited for complex tasks where instances can possess multiple attributes or labels. The multi-label classification provides a more representative depiction of real-world scenarios, acknowledging that objects or documents can exhibit diverse characteristics. Consequently, it facilitates a more nuanced understanding of the intricate relationships between instances and labels. However, it's important to note that this task is inherently more complex than multi-class classification, posing challenges in terms of both model training and interpretation. Imbalances may arise in datasets, especially when some labels are rare, potentially introducing challenges during the training process.

A. PROPOSED APPROACH

The proposed multi-label program codes' error classification approach comprises two main phases: data preprocessing and classification, employing the ML-KNN with the CodeT5 model. Below, we provide an overview of the proposed method and the architecture of the ML-KNN with the CodeT5 model.

1) ARCHITECTURE OF THE ML-KNN WITH CODET5

CodeT5 is a variant of the Text-To-Text Transfer Transformer (T5) model designed for code generation tasks [48]. It is a Transformer-based model trained on a mixture of code and natural language data to perform various tasks, including code summarization, translation, and completion. Combining CodeT5 with ML-KNN involves two distinct steps: using CodeT5 for code generation or related tasks and then using ML-KNN for multi-label classification based on the generated content. Figure 2 represents the graphical illustration of the proposed approach.

TABLE 1. Basic Statistics of the Error-labeled Dataset.

Name	Value
Source	AOJ [8] ITP1
Language	Python 3
# of Problems	44
# of Pairs	95,631
# of Users	10,361
Avg. # of Errors	3.47 (± 2.69)
Avg. Char-based Edit Distance	13.54 (± 15.23)
Avg. Char-based Similarity	90.94% (± 11.13%)
Avg. Token-based Edit Distance	5.49 (± 5.85)
Avg. Token-based Similarity	89.44% (± 12.51%)

The conceptual overview of the workflow of the model is below:

2) STEP1: USE CODET5 FOR CODE EMBEDDING

$$\text{embedding} = \text{CodeT5 (source\_code)} \tag{28}$$

3) STEP2: USE ML-KNN FOR MULTI-LABEL CLASSIFICATION:

$$\text{labels} = \text{ML\_KNN (embedding)} \tag{29}$$

Equations (28) and (29) represent the pseudocode for combining CodeT5 with ML-KNN, as they involve distinct models and processes. The outputs of CodeT5 (generated code embedding) will be used as inputs for ML-KNN.

The leftmost part of Figure 2 provides information regarding the dataset. After data preprocessing, we obtained two datasets, named the original error label (OEL) and summarized error label (SEL), along with the erroneous source code. The erroneous code is then passed to CodeT5 for embedding, as shown in the middle part. The output from CodeT5 yields the embedding matrix. In the rightmost part, the embedding matrix, along with the OEL and SEL, is passed to the ML-KNN for our classification task. Finally, multi-label error classification is performed, yielding the output from the ML-KNN.

V. EXPERIMENTS

A. DATASET

In this work, we utilized the error-labeled dataset proposed by Shirafuji et al. [14]. The dataset contains 95,631 pairs of erroneous and accepted codes, collected from 44 introductory programming problems from the course Introduction to Programming 1 (ITP1) on an online judge system. The average number of errors for each pair is 3.47 (± 2.69), and at least one error is contained. Since the erroneous code is collected from the AOJ [2], [8], it is confirmed that the code is evaluated using test input/output cases, and error verdicts are obtained. The basic statistics of the error-labeled datasets are listed in Table 1.

The overview of the data pre-processing is illustrated in Figure 3. For the multi-label error classification, we used only the erroneous code and its error labels of the code pairs. To improve the classification performance while keeping consistency, we merged some similar labels into one, as described in Section V-A1c.

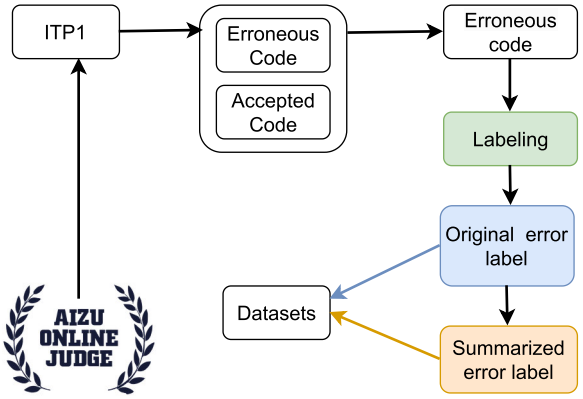


FIGURE 3. Overview of the data preprocessing.

Code	$label_1$	$label_2$	$label_3$	⋯	$label_n$
$code_1$	0	1	1	⋯	0
$code_2$	1	1	1	⋯	0
$code_3$	0	0	0	⋯	1
⋯	⋯	⋯	⋯	⋯	⋯
$code_m$	1	1	1	⋯	1

FIGURE 4. Illustration of the one-hot encoding of the error-labeled dataset. For each $code_j$, it indicates that it contains the error $label_i$ if ($label_i$, $code_j$) is 1, but not otherwise.

1) PREPROCESSING

We apply some preprocessing to utilize the dataset for multi-label classification tasks. The preprocessing has three phases: (1) filtering, (2) transformation, and (3) summarization.

a: FILTERING

To avoid cheating on the models, we filtered out duplicate pairs. We consider the data to be duplicated if both the wrong and correct programs are exactly the same. In addition, we removed the samples that contained no errors. As an integral part of the preprocessing, we also shuffled the data to mitigate the over-fitting problem and to improve the model learning.

b: TRANSFORMATION

In the data transformation phase, we applied one-hot encoding [62] to convert categorical labels to numerical values. We created a matrix, where each column represents an error, and each row represents a code pair. The one-hot encoding process is illustrated in Figure 4.

c: SUMMARIZATION

The original error label (OEL) dataset comprises 55 labels, while the summarized error label (SEL) dataset reduces this number to 11. Table 2 provides a list of both summarized and original error labels. As shown in Figure 5, some labels have very low frequency, potentially causing dataset imbalance and impacting classification results. To address this issue,

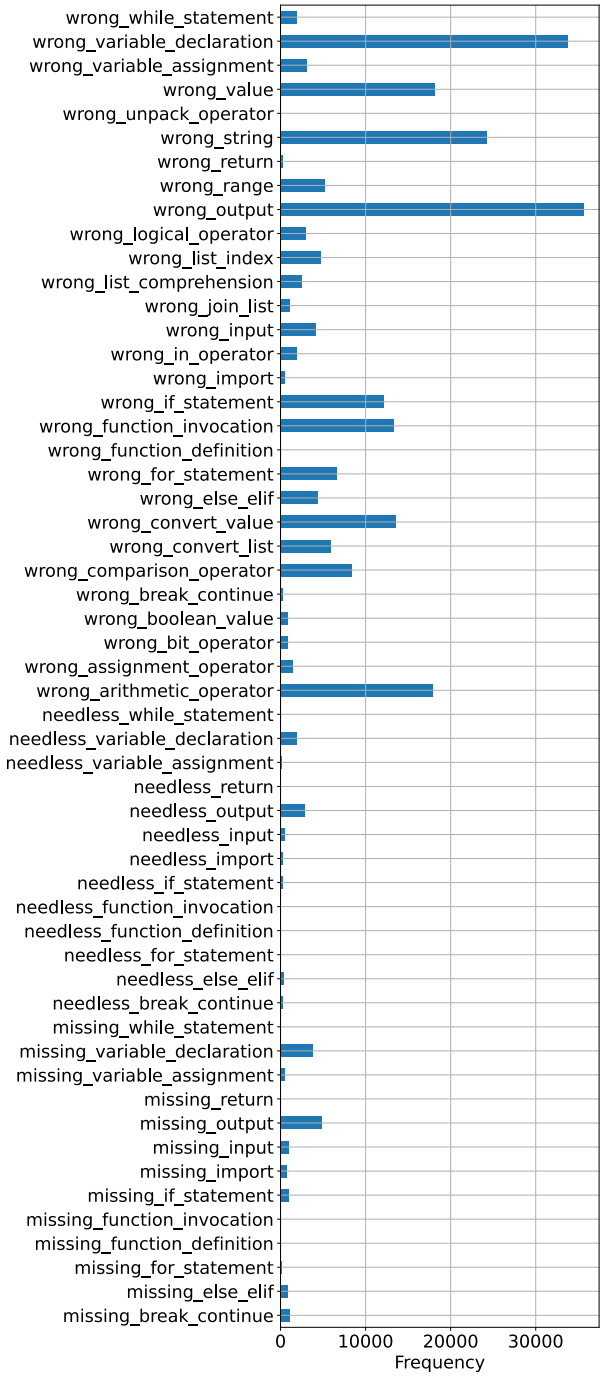


FIGURE 5. Frequency of each label in the OEL dataset.

we have summarized the dataset, as illustrated in Figure 6, and achieved a more balanced distribution of labels compared to the OEL dataset.

B. EVALUATION METRICS

In the classification task, the performance of a classifier is typically assessed using a confusion matrix. This matrix, in turn, serves as the basis for calculating key metrics, such as accuracy, precision, recall, and F1-score. We adhere to

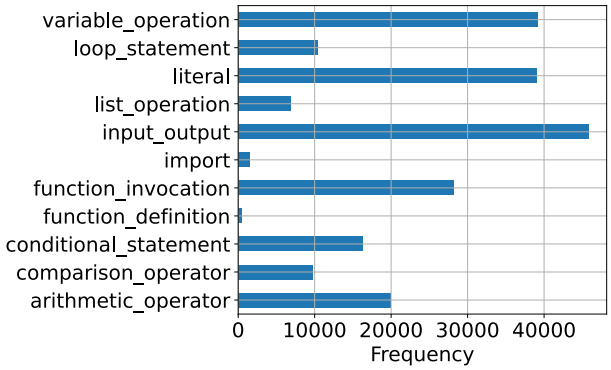


FIGURE 6. Frequency of each label in the SEL dataset.

the standard evaluation approach [24], [31], [63], [64] for our multi-label error classification task. In our classification task, we adopted the precision, recall, and F1 score including macro and weighted settings. In addition, the average accuracy and exact match accuracy are also calculated for the evaluation. The description of the evaluation matrices is presented below:

1) AVERAGE ACCURACY

The average accuracy is defined as the average number of correct predictions compared to the total number of predictions. For example, a sample (data) contains n labels (actual/true-label), now if the the output (number of predicted labels) matches the actual label then the average accuracy of the sample is:

$$Average\_accuracy = \frac{Predicted\ labels}{True\ labels} \tag{30}$$

The average accuracy of all samples is calculated as below:

$$Avg_{acc} = \frac{\sum_i^N \frac{PL_i}{TL_i}}{N} \tag{31}$$

where Avg_{acc} is the average accuracy, PL_i is the predicted labels, TL_i is the true labels, and N denotes # of all samples.

2) EXACT MATCH ACCURACY

Then we focused on calculating the exact match accuracy. In case of the exact match accuracy, if all the actual and predicted labels match then it will be the exact match. A sample contains n labels, and if the actual labels (e.g., 0, 1, 1, . . . , 0) exactly match all the predicted labels (e.g., 0, 1, 1, . . . , 0) then it is an exact match and it is calculated by the following equation:

$$EM = \frac{\sum_i^N S_i}{N} \tag{32}$$

where EM presents average exact match accuracy, S_i represents exactly matched # of samples, and N denotes # of all samples.

3) MACRO PRECISION, RECALL, AND F1 SCORE

Program code datasets often exhibit various data imbalances and to make a neutral evaluation of these data, advanced

TABLE 2. List of Summarized and Its Original Labels.

Summarized Labels	Original Labels
arithmetic_operator	wrong_arithmetic_operator
	wrong_bit_operator
	wrong_logical_operator
comparison_operator	wrong_comparison_operator
	wrong_in_operator
conditional_statement	missing_else_elif
	missing_if_statement
	needless_else_elif
	needless_if_statement
	wrong_else_elif
	wrong_if_statement
function_defination	missing_function_defination
	missing_return
	needless_function_defination
	needless_return
	wrong_function_defination
	wrong_return
function_invocation	missing_function_invocation
	needless_function_invocation
	wrong_convert_list
	wrong_convert_value
	wrong_function_invocation
	wrong_join_list
import	missing_import
	needless_import
	wrong_import
literal	wrong_boolean_value
	wrong_string
	wrong_value
input_output	missing_input
	missing_output
	needless_input
	needless_output
	wrong_input
	wrong_output
list_operation	wrong_list_comprehension
	wrong_list_index
	wrong_unpack_operator
loop_statement	missing_break_continue
	missing_for_statement
	missing_while_statement
	needless_break_continue
	needless_for_statement
	needless_while_statement
	wrong_break_continue
	wrong_for_statement
	wrong_range
	wrong_while_statement
variable_operation	missing_variable_assignment
	missing_variable_declaration
	needless_variable_assignment
	needless_variable_declaration
	wrong_assignment_operator
	wrong_variable_assignment
	wrong_variable_declaration

evaluation metrics (e.g., precision, recall, and F1-score) are adopted. Although a higher micro F1 score suggests superior overall performance, it lacks sensitivity to individual classes because of imbalanced class data distribution. To mitigate the issue, the macro-average is considered because a higher macro F1 score signifies better model performance for individual classes. For our classification task, macro F1 scores have been taken into account. The macro precision (P), recall (R), and F1 score ($F1$) are calculated as follows:

$$P = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FP_i} \tag{33}$$

$$R = \frac{1}{C}\sum_{i=1}^{C}\frac{TP_i}{TP_i + FN_i} \tag{34}$$

$$F1 = \frac{2}{C}\sum_{i=1}^{C}\frac{P_i \times R_i}{P_i + R_i} \tag{35}$$

where P_i is the precision for class i and R_i is the recall for class i.

In these equations, TP, FP, and FN refer to the corresponding counts for each class, and C is the total number of classes.

4) WEIGHTED PRECISION, RECALL, AND F1 SCORE

The weighted average F1 score is calculated by averaging all F1 scores for individual classes, taking into account the support of each class. The weighted precision, recall, and F1 score are commonly utilized metrics in evaluating classification models for imbalanced datasets. In these metrics, each class contributes proportionally to the final score based on its support (number of instances). For a multi-label classification problem, the weighted precision ($P_{weighted}$), recall ($R_{weighted}$), and F1 score ($F1_{weighted}$) are calculated as follows:

$$P_{\text{weighted}} = \frac{\sum_{i=1}^{C}\text{Support}_i \times P_i}{\sum_{i=1}^{C}\text{Support}_i} \tag{36}$$

$$R_{\text{weighted}} = \frac{\sum_{i=1}^{C}\text{Support}_i \times R_i}{\sum_{i=1}^{C}\text{Support}_i} \tag{37}$$

$$F1_{\text{weighted}} = \frac{\sum_{i=1}^{C}\text{Support}_i \times F1_i}{\sum_{i=1}^{C}\text{Support}_i} \tag{38}$$

where $F1_i$ is the F1 score for class i.

VI. EXPERIMENTAL RESULTS

A. IMPLEMENTATION DETAILS

A program code is typically a compilation of intricate instructions, encompassing mathematical operations, functions, keywords, variables, and tokens. These elements are interrelated within a program code. The selection of hyperparameters is of great importance in comprehending

TABLE 3. Hyperparameter Settings.

Hyperparameters	Value
Maximum sequence length	100
Vocabulary Size	10000
Embedding Dimension	200
Padding and Truncating	post
Optimizer	adam
Loss function	binary_crossentropy
Activation function of dense layer	ReLU
Activation function of output layer	sigmoid
Epochs	100
Batch size	256
Weight decay	1e-3
Learning rate	0.001

the complex interrelationships within the code. We employ different models for our multi-label error classification task. The details of the hyperparameters are presented in Table 3. In addition, 80%, 10%, and 10% of the data are used for the training, validation, and evaluation, respectively.

B. RESULTS

Table 4 presents the quantitative classification results for P, R, $F1$ as well as $P_{weighted}$, $R_{weighted}$, and $F1_{weighted}$ for the OEL and SEL datasets for all the five models. In addition, the Avg_{acc} and EM accuracy with the number of exactly matched error labels are also presented in Tables 5 – 6.

It is observed that the average P, R, and $F1$ scores of the GRU model for the OEL dataset remain 0.00. In this case, the GRU model failed to provide results for the dataset with a large number of class labels. For the SEL dataset, P, R, and $F1$ scores are 0.05, 0.09, and 0.06, respectively, while the $P_{weighted}$, $R_{weighted}$, and $F1_{weighted}$ scores are 0.12, 0.21, and 0.15, respectively. For the SEL dataset, which contains fewer classes, the GRU model can perform the classification task and provide results. However, similar to the GRU model, the LSTM model also failed to perform the classification task for datasets with a large number of classes. The results obtained from the LSTM model are identical to those of the GRU model, indicating that the LSTM model also fails to provide the classification results.

Unlike LSTM and GRU, the BiLSTM model can perform the classification task for both OEL and SEL datasets. The average $F1$ and $F1_{weighted}$ scores for the OEL are 0.12 and 0.36, respectively. For the SEL, the $F1$ and $F1_{weighted}$ scores are 0.45 and 0.56, respectively. The BiLSTM model demonstrates better performance in the SEL dataset compared to the OEL dataset. It also provides good results for P, $P_{weighted}$, R, and $R_{weighted}$ for the SEL dataset compared to the OEL dataset. It is observed that the BiLSTM model's performance depends on the number of class labels, showing better results for datasets with fewer classes than those with many classes. Moreover, compared to GRU and LSTM, the BiLSTM model performs better in the classification task and provides good results for both the OEL and SEL datasets.

The BiLSTM-A model demonstrates promising results over BiLSTM, LSTM, and GRU for both OEL and SEL datasets. The P and R for both the OEL and SEL are higher compared to BiLSTM, LSTM, and GRU. Although the $P_{weighted}$ for the OEL and SEL remains the same for BiLSTM-A, it is higher than all the previous models. Similarly, the $R_{weighted}$ is also higher than the previous models. The average $F1$ and $F1_{weighted}$ scores for the BiLSTM-A mechanism for the OEL dataset are 0.21 and 0.45, respectively. For the SEL dataset, the scores are 0.51 and 0.60, respectively. BiLSTM-A can perform the classification task for the both OEL and SEL datasets, but it also performs better in the SEL dataset than the OEL dataset, similar to the BiLSTM model. It is observed that BiLSTM-A exhibits better performance in the multi-label error classification task compared to BiLSTM, LSTM, and GRU.

The average $F1$ and $F1_{weighted}$ scores by the proposed model for the OEL dataset are 0.26 and 0.53, respectively. For the SEL dataset, the $F1$ and $F1_{weighted}$ scores are 0.52 and 0.65, respectively. It can be seen that the average F1 score for the SEL dataset is higher than that for the OEL dataset. The average P, $P_{weighted}$, R, and $R_{weighted}$ for the SEL dataset are also higher than the OEL dataset. In the multi-label error classification task, our proposed ML-KNN with CodeT5 demonstrates notable performance for the SEL dataset compared to the OEL dataset.

The average $F1$ and $F1_{weighted}$ scores as well as the recall for both OEL and SEL datasets demonstrate that our proposed model outperformed the BiLSTM-A, BiLSTM, LSTM, and GRU. Although the P of the BiLSTM-A is slightly higher than that of the proposed model, in comparison to other models, the proposed models' performance is better. Additionally, the $P_{weighted}$ of the proposed model is higher than all models for both OEL and SEL datasets, except for the BiLSTM-A, which is only the case for the OEL dataset. It is observed that our proposed model outperformed the BiLSTM-A, BiLSTM, LSTM, and GRU models for the overall multi-label error classification task for both the OEL and SEL datasets.

A comparison of the models' performance for the classification task based on precision is illustrated in Figure 7, and in Figure 8 the comparison is presented based on recall.

A comparison of the models' performance for the classification task based on the F1 score is illustrated in Figure 9

In Table 5, the Avg_{acc} of all the models is presented. For the OEL, both GRU and LSTM provide the same Avg_{acc} results, and the same applies to the SEL dataset. The BiLSTM model yields better results than LSTM and GRU for both the OEL and SEL datasets in terms of Avg_{acc}. The BiLSTM-A outperforms the BiLSTM, LSTM, and GRU models for both OEL and SEL datasets. Finally, our proposed ML-KNN with CodeT5 exhibits superior performance compared to all the baseline models for both the OEL and SEL datasets. It is observed that ML-KNN with CodeT5 outperforms all the

TABLE 4. Classification Results of All the Models for the OEL and SEL Datasets.

Metrics		GRU		LSTM		BiLSTM		BiLSTM-A		CodeT5 + ML-KNN	
		OEL	SEL	OEL	SEL	OEL	SEL	OEL	SEL	OEL	SEL
Precision	Macro	0.00	0.05	0.00	0.05	0.39	0.63	0.52	0.65	0.44	0.64
	Weighted	0.00	0.12	0.00	0.12	0.65	0.67	0.69	0.69	0.66	**0.70**
Recall	Macro	0.00	0.09	0.00	0.09	0.09	0.38	0.16	0.44	**0.20**	**0.46**
	Weighted	0.00	0.21	0.00	0.21	0.29	0.51	0.36	0.55	**0.46**	**0.62**
F1 Score	Macro	0.00	0.06	0.00	0.06	0.12	0.45	0.21	0.51	**0.26**	**0.52**
	Weighted	0.00	0.15	0.00	0.15	0.36	0.56	0.45	0.60	**0.53**	**0.65**

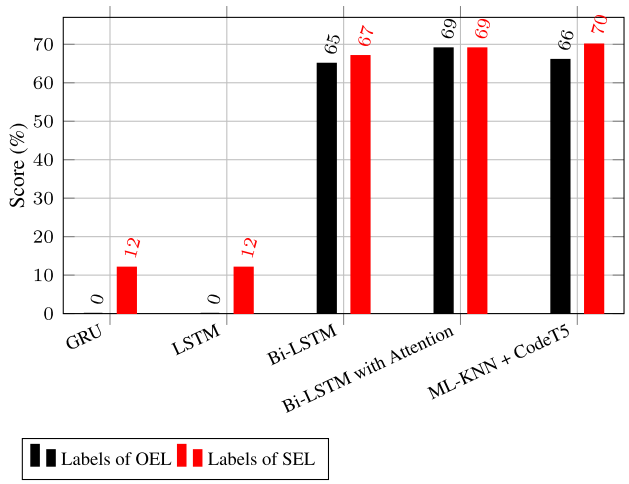


FIGURE 7. Comparisons of models' performance in code label classification based on $P_{weighted}$ scores.

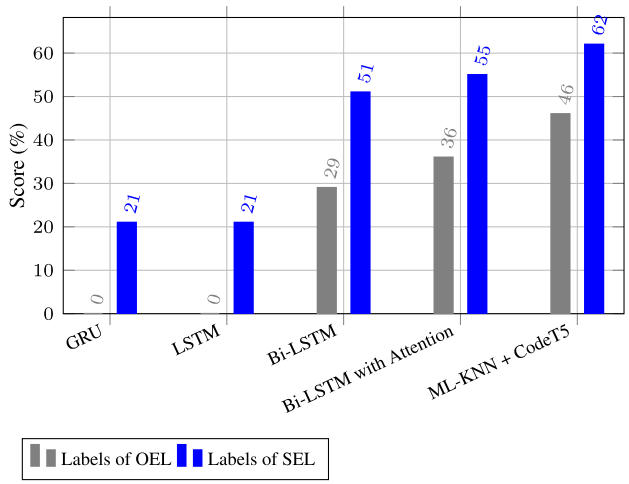


FIGURE 8. Comparisons of models' performance in code label classification based on $R_{weighted}$ scores.

other models and performs the classification task better in terms of Avg_{acc}.

In Figure 10 the comparison of the Avg_{acc} for the OEL and SEL datasets is illustrated.

The *EM* accuracy of all the models for the OEL and SEL datasets is presented in Table 6. For the OEL dataset, the GRU and LSTM models fail to provide the *EM* accuracy while they provide results for the SEL dataset and the results are the same

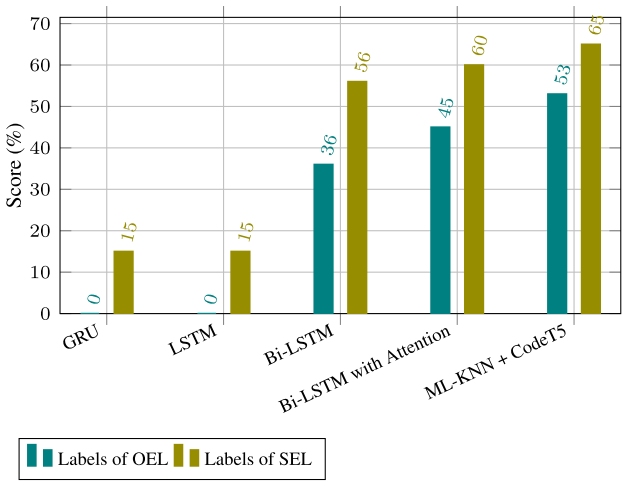


FIGURE 9. Comparisons of models' performance in code label classification based on $F1_{weighted}$ scores.

TABLE 5. Avg_{acc} of All the Models for the OEL and SEL Datasets.

Model	Average Accuracy for Data Labels	
	OELs	SELs
GRU	94.56%	77.18%
LSTM	94.56%	77.18%
BiLSTM	95.44%	82.81%
BiLSTM-A	95.67%	83.79%
ML-KNN with CodeT5	**95.91%**	**84.77%**

for both models. The *EM* accuracy by the BiLSTM model for the OEL and SEL datasets is 12.80% and 19.26%, with the number of exactly matched labels being 1061 and 1594, respectively. The *EM* accuracy for the SEL dataset is better than the OEL dataset for the BiLSTM model.

The BiLSTM-A provides the *EM* accuracy for the OEL dataset with the result being 16.84% and the number of exact match labels being 1396. For the SEL dataset, the result is 22.12%, with the number of exact match labels being 1831. The *EM* accuracy for the SEL dataset of the BiLSTM-A is better than for the OEL dataset. Additionally, the *EM* accuracy of the BiLSTM-A is also superior to BiLSTM, LSTM, and GRU models for both OEL and SEL datasets.

The *EM* accuracy of our proposed ML-KNN with CodeT5 for the OEL dataset is 22.57% and the number of exactly

TABLE 6. *EM* Accuracy of All the Models for the OEL and SEL Datasets.

Model	Exact Match Accuracy for Data Labels			
	OEL	# Of Exact Match for OEL	SEL	# Of Exact Match for SEL
GRU	0.00%	0	8.03%	665
LSTM	0.00%	0	8.03%	665
BiLSTM	12.80%	1061	19.26%	1594
BiLSTM-A	16.84%	1396	22.12%	1831
ML-KNN with CodeT5	**22.57%**	**1871**	**27.22%**	**2253**

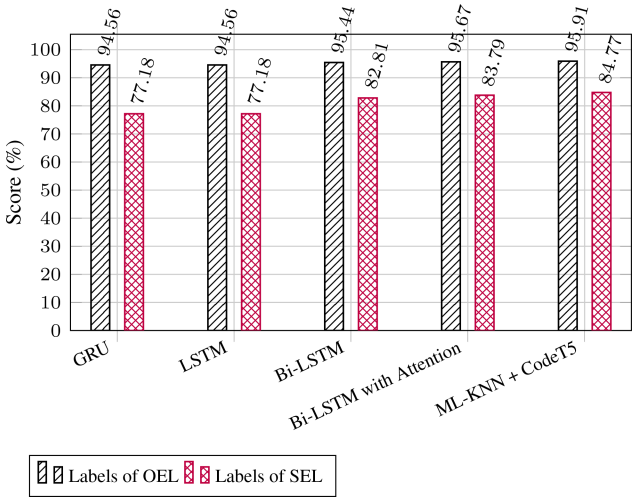


FIGURE 10. Comparisons of models' performance in code label classification based on Avg_{acc}.

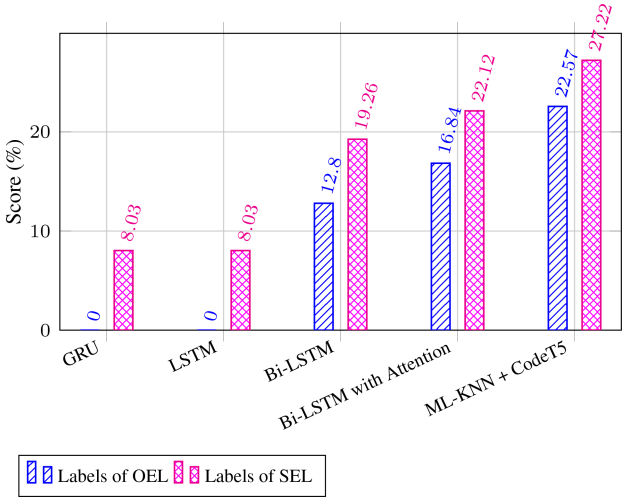


FIGURE 11. Comparisons of models' performance in code label classification based on *EM* accuracy.

match labels is 1871. In terms of the SEL dataset, the result is 27.22% and the number of matched labels is 2253. The proposed model also demonstrates a better *EM* accuracy for the SEL dataset than the OEL dataset. Furthermore, compared to the other models, our proposed model outperformed the *EM* accuracy for both the OEL and SEL datasets.

In Figure 11 the comparison of the *EM* accuracy for the OEL and SEL datasets is illustrated. The number of exactly matched labels for both datasets is presented in Figure 12

The comprehensive comparison of results, encompassing $F1_{weighted}$, Avg_{acc}, and *EM*, for all baseline models along with our proposed model, across both the OEL and SEL datasets, is illustrated in Figure 13.

VII. DISCUSSION

A. PERFORMANCE ANALYSIS

In this study, we approached the CodeT5 model with ML-KNN for the multi-label error classification task in program codes, aiming to address the inherent complexity and diversity present in program codes. We assess the model's performance through training, validation, and testing with real-world program codes. The quantitative classification results (Table 4) demonstrate that the proposed model outperforms other baseline models such as GRU, LSTM, BiLSTM, and BiLSTM-A. The comparison of the models' performance based on $P_{weighted}$, $R_{weighted}$, and $F1_{weighted}$ score is visualized in Figures 7, 8, and 9, respectively. Notably, the performance of the models seems to be influenced by the number of classes, with the SELs showing better performance compared to the OELs across all models. The accuracy, including Avg_{acc} and *EM* accuracy is presented in Tables 5 and 6, and their comparison is illustrated in Figures 10 and 11. We observed that the Avg_{acc} is generally higher for the OELs compared to the SELs, which could be attributed to the variations in the test data. As the number of classes differs between the OEL and SELs, the test data for each also varies. The OELs encompass a larger number of labels compared to the SELs, resulting in a higher likelihood of predicted labels in the test data.

In terms of *EM* accuracy, our proposed CodeT5 with ML-KNN consistently outperformed the other baseline models for both OELs and SELs, as shown in Table 6 and Figure 12. Particularly, our model achieves higher accuracy in matching exactly labeled errors compared to other models, indicating its superior performance in the multi-label error classification task.

Overall, our study demonstrates that the proposed CodeT5 with the ML-KNN model is highly effective in classifying errors in program codes, outperforming the other baseline models across various evaluation metrics.

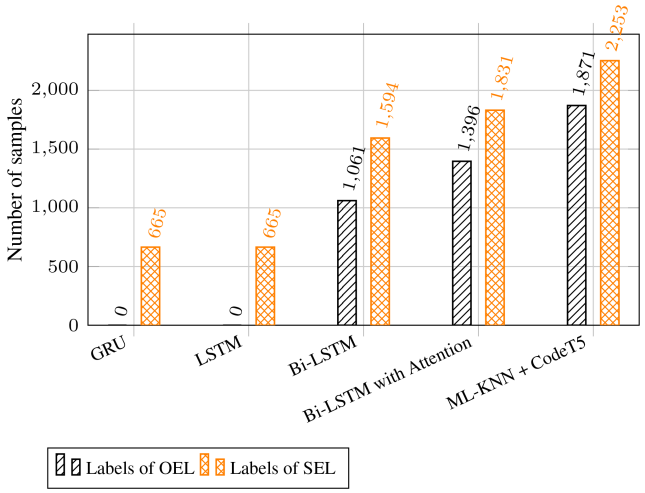


FIGURE 12. Comparisons of models' performance in code label classification based on number of exactly matched labels.

B. SCALABILITY

In this study, multi-label error classification model is employed to classify errors that exist in the program code. The model classifies code errors and presents the output as a label. Our proposed approach demonstrated superior performance compared to the baseline models in the error classification task, particularly those in Python, which is considered a procedural language. Therefore, the proposed model has the potential to classify program codes in other procedural languages, such as C, C++, and Java. Additionally, the proposed model is scalable and can handle large industrial program codes, which are often lengthy and contain numerous functions and classes. Given that industrial codes can be extensive and diverse, with various functions and classes, the proposed model proves beneficial for their classification. Moreover, this study can be extended using different approaches and methods, making it suitable for large-scale applications. It can be utilized not only for multi-label error classification, but also for tasks such as classifying algorithms, function classification, and other classification scenarios. It is evident that the proposed multi-label error classification model holds utility and scalability for various programming-related tasks.

C. SUITABILITY FOR PROGRAMMING LEARNING

One of our objectives is to explore how the model can aid programmers in real-world programming environments. With this goal in mind, the proposed model has been developed. The dataset used in this study consists of solution codes from the course Introduction to Programming 1 (ITP1) obtained from an Online Judge (OJ) system. The ITP1 course encompasses a wide range of problems, including basic, interesting, and practical problems, aimed at fostering fundamental programming skills. These problems cover diverse topics such as conditional branching, computation, arrays, strings, mathematical functions, structured programming, and object-oriented concepts. Therefore, the dataset derived from this course is heterogeneous, containing various types of errors and presenting diverse learning opportunities.

By leveraging this heterogeneous dataset, our model offers valuable insights and assistance to programmers and learners. The model's ability to effectively classify errors in solution codes from a diverse range of programming problems enhances its utility in practical programming scenarios. Moreover, it contributes to the improvement of programming skills by providing targeted feedback and guidance tailored to the specific errors encountered. Thus, the employment of our model proves beneficial not only for individual programmers but also for enhancing programming education and learning experiences.

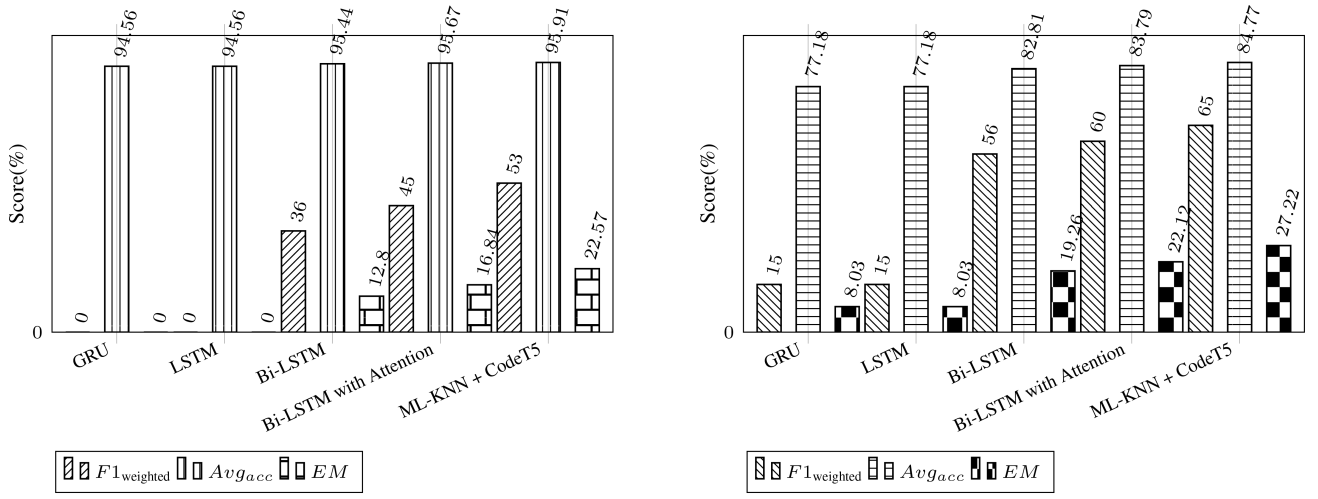
The experimental results suggest that the study has the potential to be valuable in programming learning. Given the continuous generation of programming code from various sources, such as academia, industry, programming platforms, and OJs, the proposed multi-label error classification model offers programmers, especially novices, an advantage in identifying and recognizing errors within their code in extensive repositories. This model has the potential to streamline programming methods and enhance technical skills. The experimental outcomes demonstrate that the proposed model excels in classifying diverse and complex codes with a higher degree of accuracy. Additionally, the model can be integrated into existing programming learning platforms, such as OJ systems.

D. SCOPE FOR SOFTWARE ENGINEERING

Repositories of real-world program codes are integral to building effective machine learning/deep learning models in software engineering. These models find applicability in various software engineering fields, including design and analysis, classifying errors, code review, code reuse, and intelligent programming assistants (IPA). Our proposed multi-label error classification model specializes in classifying program codes. Consequently, this model can be directly or indirectly applied to various software engineering tasks such as code review, error classification, and assisting with code refactoring. In particular, the proposed model can serve as a supporting component for other machine-learning/deep-learning models in software engineering that deal with program codes.

E. PRE-TRAINED LLMS FOR CODING TASKS

In recent times, pre-trained LLMs performance for code-related tasks has been promising. For example, CodeT5+ which is an iteration and updated version of the CodeT5. CodeT5 utilizes an identifier-aware, unified pre-trained encoder-decoder architecture [48]. On the other hand, CodeT5+ expands on this by integrating a flexible architecture that allows it to operate as encoder-only, decoder-only, or both [65]. The CodeT5+ model sizes range from 220M to 16B which enables it to adapt to a wide range of code-related tasks more efficiently. CodeT5+ uses off-the-shelf



(a) OEL Dataset (b) SEL Dataset

FIGURE 13. Overview of the $F1_{\text{weighted}}$, Avg_{acc}, and *EM* for OEL and SEL datasets.

Large Language Models (LLMs) for initialization, which is an approach that differs from the earlier CodeT5. This method allows CodeT5+ to scale up more efficiently by leveraging existing pre-trained models. It employs a ''shallow encoder and deep decoder'' architecture aimed at enhancing computational efficiency.

In our proposed approach CodeT5 is used for the matrix embedding and as CodeT5+ is the updated version of CodeT5, it might be fitted with our proposed approach and could increase the performance. In addition, other LLMs such as Codebert [51], and LLaMA [66] could be suitable for our proposed approach. Our future plan is to use fine-tuned transformer-based models for the multi-label classification task, we will consider CodeT5+ as well as other LLMs.

F. THREATS TO VALIDITY

This study introduced an innovative approach from data preprocessing to model development for the classification task, resulting in notable achievements in classifying errors during experimentation. However, the proposed model may encounter challenges or potential threats due to the following reasons: (*i*) the results could vary from dataset to dataset, (*ii*) other hyperparameters could influence the model's performance (*iii*) data preprocessing and segmentation strategy may affect the quality of input data, (*iv*) may differ when applied to other programming languages

VIII. CONCLUSION

In this study, we employed CodeT5 with ML-KNN for the multi-label error classification task in the program code, utilizing real-world program codes collected from an online judge system. Two different datasets, named the OEL and SEL, were used. Comprehensive data preprocessing, including filtering, transformation, error labeling, and summarization, was conducted for model training. The model performance for both datasets is promising. For the OEL, the Avg_{acc} and *EM* accuracy are 95.91% and 22.57%, respectively. The Avg_{acc} and *EM* accuracy for the SELs are 84.77% and 27.22%, respectively. The precision *P*, recall *R*, and $F1$ scores for the weighted metrics for the OEL are 0.66, 0.46, and 0.53, respectively. For the SELs, the precision *P*, recall *R*, and $F1$ scores for the weighted metrics are 0.70, 0.62, and 0.65, respectively. The model performance for both datasets is significant, but the SELs dataset produces better results than the OELs. Additionally, we employed GRU, LSTM, BiLSTM, and BiLSTM-A as baseline models for comparison. The proposed model outperformed the baseline models across all evaluation metrics used.

In the future, fine-tuned transformer-based models can be considered for the multi-label error classification tasks, providing an opportunity for improved performance. Reproducibility can be enhanced by exploring this task with different approaches. Moreover, expanding the dataset to include more problem-solving data from various courses and additional programming languages could contribute to a more comprehensive analysis.

REFERENCES

[1] M. M. Rahman, Y. Watanobe, R. U. Kiran, T. C. Thang, and I. Paik, ''Impact of practical skills on academic performance: A data-driven analysis,'' *IEEE Access*, vol. 9, pp. 139975–139993, 2021.

[2] Y. Watanobe, M. M. Rahman, T. Matsumoto, U. K. Rage, and P. Ravikumar, ''Online judge system: Requirements, architecture, and experiences,'' *Int. J. Softw. Eng. Knowl. Eng.*, vol. 32, no. 6, pp. 917–946, Jun. 2022.

[3] R. Yera and L. Martínez, ''A recommendation approach for programming online judges supported by data preprocessing techniques,'' *Appl. Intell.*, vol. 47, no. 2, pp. 277–290, Sep. 2017.

[4] M. M. Rahman, Y. Watanobe, T. Matsumoto, R. U. Kiran, and K. Nakamura, ''Educational data mining to support programming learning using problem-solving data,'' *IEEE Access*, vol. 10, pp. 26186–26202, 2022.

[5] R. Romli, S. Sulaiman, and K. Z. Zamli, ''Improving automated programming assessments: User experience evaluation using FaSt-generator,'' *Proc. Comput. Sci.*, vol. 72, pp. 186–193, Jan. 2015.

[6] I. Mekterovic, L. Brkic, B. Milašinovic, and M. Baranovic, ''Building a comprehensive automated programming assessment system,'' *IEEE Access*, vol. 8, pp. 81154–81172, 2020.

[7] N. A. Rashid, L. W. Lim, O. S. Eng, T. H. Ping, Z. Zainol, and O. Majid, "A framework of an automatic assessment system for learning programming," in *Advanced Computer and Communication Engineering Technology*. Cham, Switzerland: Springer, 2016, pp. 967–977.

[8] Y. Watanobe, "Aizu online judge," Tech. Rep., 2018.

[9] M. A. Revilla, S. Manzoor, and R. Liu, "Competitive learning in informatics: The UVA online judge experience," *Olympiads Informat.*, vol. 2, no. 10, pp. 131–148, 2008.

[10] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "URI online judge academic: A tool for algorithms and programming classes," in *Proc. 9th Int. Conf. Comput. Sci. Educ.*, Aug. 2014, pp. 149–152.

[11] M. Mirzayanov, O. Pavlova, P. Mavrin, R. Melnikov, A. Plotnikov, V. Parfenov, and A. Stankevich, "Codeforces as an educational platform for learning programming in digitalization," *OLYMPIADS Informat.*, vol. 14, nos. 133–142, p. 14, 2020.

[12] J. Petit, S. Roura, J. Carmona, J. Cortadella, J. Duch, O. Gimnez, A. Mani, J. Mas, E. Rodrguez-Carbonell, E. Rubio, E. D. S. Pedro, and D. Venkataramani, "Jutge.Org: Characteristics and experiences," *IEEE Trans. Learn. Technol.*, vol. 11, no. 3, pp. 321–333, Jul. 2018.

[13] A. C. Graesser, M. W. Conley, and A. Olney, "Intelligent tutoring systems," in *APA Educational Psychology Handbook: Application to Learning and Teaching*, vol. 3, 2012, pp. 451–473.

[14] A. Shirafuji, T. Matsumoto, M. F. Ibne Amin, and Y. Watanobe, "Rule-based error classification for analyzing differences in frequent errors," in *Proc. IEEE Int. Conf. Teach., Assessment Learn. Eng. (TALE)*, Nov. 2023, pp. 1–7.

[15] M. M. Rahman, Y. Watanobe, and K. Nakamura, "Source code assessment and classification based on estimated error probability using attentive LSTM language model and its application in programming education," *Appl. Sci.*, vol. 10, no. 8, p. 2973, Apr. 2020.

[16] Q. Liu, J. Chen, F. Chen, K. Fang, P. An, Y. Zhang, and S. Du, "MLGN: A multi-label guided network for improving text classification," *IEEE Access*, vol. 11, pp. 80392–80402, 2023.

[17] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning-based text classification: A comprehensive review," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–40, Apr. 2022.

[18] M. Han, H. Wu, Z. Chen, M. Li, and X. Zhang, "A survey of multi-label classification based on supervised and semi-supervised learning," *Int. J. Mach. Learn. Cybern.*, vol. 14, no. 3, pp. 697–724, Mar. 2023.

[19] M.-L. Zhang, Y.-K. Li, X.-Y. Liu, and X. Geng, "Binary relevance for multi-label learning: An overview," *Frontiers Comput. Sci.*, vol. 12, no. 2, pp. 191–202, Apr. 2018.

[20] A. Law, K. Chakraborty, and A. Ghosh, "Functional link artificial neural network for multi-label classification," in *Proc. Int. Conf. Mining Intell. Knowl. Explor.*, Hyderabad, India. Cham, Switzerland: Springer, 2017, pp. 1–10.

[21] J. M. Moyano, E. L. Gibaja, K. J. Cios, and S. Ventura, "Review of ensembles of multi-label classifiers: Models, experimental study and prospects," *Inf. Fusion*, vol. 44, pp. 33–45, Nov. 2018.

[22] M. M. Rahman, Y. Watanobe, R. U. Kiran, and R. Kabir, "A stacked bidirectional LSTM model for classifying source codes built in MPLs," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Cham, Switzerland: Springer, 2021, pp. 75–89.

[23] Y. Watanobe, M. M. Rahman, M. F. I. Amin, and R. Kabir, "Identifying algorithm in program code based on structural features using CNN classification model," *Appl. Intell.*, vol. 53, no. 10, pp. 12210–12236, May 2023.

[24] M. M. Rahman and Y. Watanobe, "Multilingual program code classification using *n*-layered bi-LSTM model with optimized hyperparameters," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 8, no. 2, pp. 1452–1468, Apr. 2024.

[25] Md. F. Ibne Amin, Md. M. Rahman, Y. Watanobe, and M. M. Daniel, "Impact of programming language skills in programming learning," in *Proc. IEEE 15th Int. Symp. Embedded Multicore/Many-Core Syst.-Chip (MCSoC)*, Dec. 2022, pp. 271–277.

[26] M. Mostafizer Rahman, Y. Watanobe, A. Shirafuji, and M. Hamada, "Exploring automated code evaluation systems and resources for code analysis: A comprehensive survey," 2023, *arXiv:2307.08705*.

[27] M. M. Rahman, Y. Watanobe, U. K. Rage, and K. Nakamura, "A novel rule-based online judge recommender system to promote computer programming education," in *Proc. 34th Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst.*, Kuala Lumpur, Malaysia. Cham, Switzerland: Springer, 2021, pp. 15–27.

[28] T. Saito and Y. Watanobe, "Learning path recommendation system for programming education based on neural networks," *Int. J. Distance Educ. Technol.*, vol. 18, no. 1, pp. 36–64, Jan. 2020.

[29] A. Shirafuji, Y. Watanobe, T. Ito, M. Morishita, Y. Nakamura, Y. Oda, and J. Suzuki, "Exploring the robustness of large language models for solving programming problems," 2023, *arXiv:2306.14583*.

[30] Y. Teshima and Y. Watanobe, "Bug detection based on LSTM networks and solution codes," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC)*, Oct. 2018, pp. 3541–3546.

[31] A. Law and A. Ghosh, "Multi-label classification using binary tree of classifiers," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 6, no. 3, pp. 677–689, Jun. 2022.

[32] Y. Zhou, S. Cui, and Y. Wang, "Machine learning based embedded code multi-label classification," *IEEE Access*, vol. 9, pp. 150187–150200, 2021.

[33] A. Alshanqiti and A. Namoun, "Predicting Student performance and its influential factors using hybrid regression and multi-label classification," *IEEE Access*, vol. 8, pp. 203827–203844, 2020.

[34] Q. Wu, M. Tan, H. Song, J. Chen, and M. K. Ng, "ML-FOREST: A multi-label tree ensemble method for multi-label classification," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 10, pp. 2665–2680, Oct. 2016.

[35] Z. Sun, X. Liu, K. Hu, Z. Li, and J. Liu, "An efficient multi-label SVM classification algorithm by combining approximate extreme points method and divide-and-conquer strategy," *IEEE Access*, vol. 8, pp. 170967–170975, 2020.

[36] Z.-B. Yu and M.-L. Zhang, "Multi-label classification with label-specific feature generation: A wrapped approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5199–5210, Sep. 2022.

[37] W. Weng, D.-H. Wang, C.-L. Chen, J. Wen, and S.-X. Wu, "Label specific features-based classifier chains for multi-label classification," *IEEE Access*, vol. 8, pp. 51265–51275, 2020.

[38] Z. Liu, C. Tang, S. E. Abhadiomhen, X.-J. Shen, and Y. Li, "Robust label and feature space co-learning for multi-label classification," *IEEE Trans. Knowl. Data Eng.*, pp. 1–14, 2022.

[39] M. Wever, A. Tornede, F. Mohr, and E. Hüllermeier, "AutoML for multi-label classification: Overview and empirical evaluation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 3037–3054, Sep. 2021.

[40] W. Huang, E. Chen, Q. Liu, H. Xiong, Z. Huang, S. Tong, and D. Zhang, "HmcNet: A general approach for hierarchical multi-label classification," *IEEE Trans. Knowl. Data Eng.*, pp. 1–16, 2022.

[41] X. Xiao and K. Li, "Multi-label classification for power quality disturbances by integrated deep learning," *IEEE Access*, vol. 9, pp. 152250–152260, 2021.

[42] N. Masuyama, Y. Nojima, C. K. Loo, and H. Ishibuchi, "Multi-label classification via adaptive resonance theory-based clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–18, 2022.

[43] J. Gong, Z. Teng, Q. Teng, H. Zhang, L. Du, S. Chen, M. Z. A. Bhuiyan, J. Li, M. Liu, and H. Ma, "Hierarchical graph transformer-based deep learning model for large-scale multi-label text classification," *IEEE Access*, vol. 8, pp. 30885–30896, 2020.

[44] L. Cai, Y. Song, T. Liu, and K. Zhang, "A hybrid BERT model that incorporates label semantics via adjustive attention for multi-label text classification," *IEEE Access*, vol. 8, pp. 152183–152192, 2020.

[45] H. Peng, J. Li, S. Wang, L. Wang, Q. Gong, R. Yang, B. Li, P. S. Yu, and L. He, "Hierarchical taxonomy-aware and attentional graph capsule RCNNs for large-scale multi-label text classification," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 6, pp. 2505–2519, Jun. 2021.

[46] R. C. Morales-Hernández, J. G. Jagüey, and D. Becerra-Alonso, "A comparison of multi-label text classification models in research articles labeled with sustainable development goals," *IEEE Access*, vol. 10, pp. 123534–123548, 2022.

[47] M. M. Rahman, Y. Watanobe, and K. Nakamura, "A bidirectional LSTM language model for code evaluation and repair," *Symmetry*, vol. 13, no. 2, p. 247, Feb. 2021.

[48] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder–decoder models for code understanding and generation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2021, pp. 8696–8708.

[49] A. Shirafuji, M. M. Rahman, M. F. Ibne Amin, and Y. Watanobe, "Program repair with minimal edits using CodeT5," in *Proc. 12th Int. Conf. Awareness Sci. Technol. (iCAST)*, Nov. 2023, pp. 178–184.

[50] J. Wei, G. Durrett, and I. Dillig, "TypeT5: Seq2seq type inference using static analysis," 2023, *arXiv:2303.09564*.

[51] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.
[52] W. Wang, Y. Wang, S. Joty, and S. C. H. Hoi, "RAP-Gen: Retrieval-augmented patch generation with CodeT5 for automatic program repair," in *Proc. 31st ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2023, pp. 146–158.
[53] L. Gong, M. Elhoushi, and A. Cheung, "AST-T5: Structure-aware pretraining for code generation and understanding," 2024, *arXiv:2401.03003*.
[54] J. Zhang, J. Cambronero, S. Gulwani, V. Le, R. Piskac, G. Soares, and G. Verbruggen, "Repairing bugs in Python assignments using large language models," 2022, *arXiv:2209.14876*.
[55] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," *Pattern Recognit.*, vol. 40, no. 7, pp. 2038–2048, Jul. 2007.
[56] W. Akanda and A. Uddin, "Multi-label Bengali article classification using ML-KNN algorithm and neural network," in *Proc. Int. Conf. Inf. Commun. Technol. Sustain. Develop. (ICICT4SD)*, Feb. 2021, pp. 466–471.
[57] M.-L. Zhang and Z.-H. Zhou, "A k-nearest neighbor based algorithm for multi-label classification," in *Proc. IEEE Int. Conf. Granular Comput.*, 2005, pp. 718–721.
[58] S. Gao, X. Yang, L. Zhou, and S. Yao, "The research of multi-label *k*-nearest neighbor based on descending dimension," in *Proc. IEEE 16th Int. Conf. Softw. Eng. Res., Manage. Appl. (SERA)*, Jun. 2018, pp. 129–135.
[59] H. Suyal and A. Gupta, "An improved multi-label k-Nearest neighbour algorithm with prototype selection using DENCLUE," in *Proc. 9th Int. Conf. Rel., INFOCOM Technol. Optim., Trends Future Direction (ICRITO)*, Sep. 2021, pp. 1–6.
[60] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.
[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[62] M. K. Dahouda and I. Joe, "A deep-learned embedding technique for categorical features encoding," *IEEE Access*, vol. 9, pp. 114381–114391, 2021.
[63] Z. Hameed and B. Garcia-Zapirain, "Sentiment classification using a single-layered BiLSTM model," *IEEE Access*, vol. 8, pp. 73992–74001, 2020.
[64] M. M. Rahman, Y. Watanobe, and K. Nakamura, "A neural network based intelligent support model for program code completion," *Sci. Program.*, vol. 2020, pp. 1–18, Jul. 2020.
[65] Y. Wang, H. Le, A. Deepak Gotmare, N. D. Q. Bui, J. Li, and S. C. H. Hoi, "CodeT5+: Open code large language models for code understanding and generation," 2023, *arXiv:2305.07922*.
[66] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models," 2023, *arXiv:2302.13971*.

MD. FAIZUL IBNE AMIN (Graduate Student Member, IEEE) received the B.Sc. degree in engineering from the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh, in 2018, and the M.Sc. degree from the Graduate School of Computer Science and Engineering, Department of Computer and Information Systems, Aizuwakamatsu, Fukushima, Japan, in 2022. He is currently pursuing the Ph.D. degree with the Database Systems Laboratory, Department of Computer and Information Systems, The University of Aizu, Aizuwakamatsu. His research interests include machine learning, deep learning, blockchain, payment channel networks, educational data mining, and machine learning applications in programming.

ATSUSHI SHIRAFUJI (Member, IEEE) received the B.S. degree from the School of Computer Science and Engineering, Department of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Fukushima, Japan, in 2022, where he is currently pursuing the M.S. degree with the Graduate School of Computer Science and Engineering, Department of Computer and Information Systems. His research interests include software engineering, programming education, natural language processing, and machine learning.

MD. MOSTAFIZER RAHMAN received the B.Sc. degree in engineering from the Department of Computer Science and Engineering, Hajee Mohammad Danesh Science and Technology University, Dinajpur, Bangladesh, in 2009, the M.Sc. degree in engineering from Dhaka University of Engineering and Technology, Gazipur, Bangladesh, in 2014, and the Ph.D. degree from the Department of Computer and Information Systems, The University of Aizu, Japan, in 2022. He is currently working with Dhaka University of Engineering and Technology. His research interests include machine learning, machine learning applications, AI for Code, NLP, data mining, information visualization, and big data analytics.

YUTAKA WATANOBE (Member, IEEE) received the M.S. and Ph.D. degrees from The University of Aizu, Japan, in 2004 and 2007, respectively. He was a Research Fellow at Japan Society for the Promotion of Science (JSPS), The University of Aizu, in 2007. He is currently a Senior Associate Professor with the School of Computer Science and Engineering, The University of Aizu. He is also the Director of i-SOMET and a Developer of Aizu Online Judge (AOJ) system. His research interests include intelligent software, programming environment, smart learning, machine learning, data mining, cloud robotics, and visual languages. He is a member of IPSJ.