INTERNATIONAL JOURNAL OF NOVEL RESEARCH AND DEVELOPMENT (IJNRD)
© 2026 IJNRD | Volume 11, Issue 5, May 2026 | ISSN: 2456-4184 | IJNRD.ORG

Efficient Multi-Label Source Code Error Classification Using DistilBERT Model

SURISETTI SANDHYA
Master Of Computer Applications
Ideal College Of Arts & Sciences,
Autonomous, Affiliated To Adikavi Nannaya University - Rajamahendravaram
Kakinada

V. JEEVANKANTH
Assistant.Prof, Master Of Computer Applications
Ideal College Of Arts & Sciences,
Autonomous, Affiliated To Adikavi Nannaya University - Rajamahendravaram
Kakinada

Dr. VSV DEEPAK
HOD, Department of computer science
Ideal College Of Arts & Sciences,
Autonomous, Affiliated To Adikavi Nannaya University - Rajamahendravaram
Kakinada

***Abstract*— Accurate identification of programming errors remains a challenge due to the complexity and multi-label nature of source code defects. The existing system employs fine-tuned BERT models to classify errors with strong performance, but it introduces high computational overhead and slower execution. To improve efficiency, this work extends the original model by incorporating DistilBERT, a compressed transformer architecture that reduces model size while maintaining performance. The extended approach follows the same preprocessing pipeline, including dataset preparation, label encoding, and model training using the CodeNet dataset. Evaluation results indicate that the DistilBERT-based model achieves an accuracy of 93.84%, outperforming the baseline BERT-Cased model while significantly reducing execution time. The proposed extension enhances scalability and makes the system more suitable for real-time applications. This demonstrates that optimized transformer models can deliver both high accuracy and improved efficiency in source code error classification tasks.**

Keywords*— *Source Code, DistilBERT, Code Analysis, Transformer Models

I. INTRODUCTION

Programming has become a core skill across computer science and engineering domains, yet writing error-free code remains a persistent challenge. Developers, especially beginners, frequently encounter syntax, logical, and runtime errors that are often difficult to interpret and resolve. These issues not only slow down development but also affect software reliability and learning efficiency. Traditional debugging approaches rely heavily on manual inspection or compiler-generated messages, which are often limited in clarity and fail to provide deeper insight into the nature of errors. As software systems grow in complexity, the need for intelligent and automated error understanding mechanisms becomes increasingly important.

Recent advancements in machine learning and natural language processing have opened new directions for analyzing source code. Unlike conventional rule-based systems, data-driven models can learn patterns from large code repositories and identify relationships between code structures and error types. This shift has enabled more accurate classification and interpretation of programming errors. Multi-label classification has gained attention in this context, as a single piece of code may contain multiple types of errors simultaneously, requiring more flexible and robust modeling techniques.

Large-scale datasets such as CodeNet have further supported research in this area by providing diverse and structured code samples. These datasets allow models to generalize better across different programming styles and error patterns. However, handling such data also introduces challenges related to computational cost, memory usage, and scalability. As a result, there is a growing focus on developing efficient models that can balance performance with practical constraints.

II. RELATED WORK

Early research in programming education focused on intelligent learning environments and automated assessment systems. The work of A. C. Graesser (2012) introduced intelligent tutoring systems that adapt to learner behavior and provide personalized feedback, forming the conceptual base for automated learning support. Later, R. Romli (2015) and N. A. Rashid (2015) emphasized improving automated programming assessment by focusing on system usability and structured evaluation frameworks. These studies highlighted the limitations of traditional grading approaches and established the importance of meaningful feedback in programming environments. Further advancements were seen in the work of R. Yera (2017), which incorporated data preprocessing and recommendation techniques to enhance programming learning systems through personalization.

With the growth of machine learning, research shifted toward data-driven approaches for analyzing programming errors. M. M. Rahman (2020) applied attentive LSTM models to classify source code errors based on probability, demonstrating the effectiveness of deep learning in capturing sequential patterns in code. Similarly, Z. Li (2021) proposed a deep learning-based framework for compilation error classification, focusing on improving support for novice programmers. In parallel, studies such as Y.-T. Lin (2022) explored cognitive aspects of programming by analyzing expert coding practices, linking learning behavior with error understanding. Additionally, M. M. Rahman (2022) utilized educational data mining to examine problem-solving patterns, reinforcing the importance of behavioral data in error analysis.

Recent research has increasingly adopted transformer-based models for multi-label classification tasks. Q. Chen (2022) demonstrated the effectiveness of BERT-based architectures in

handling complex multi-label problems, highlighting their ability to capture contextual dependencies. Extending this direction, M. F. I. Amin (2024) combined CodeT5 with machine learning classifiers to improve source code error classification performance. These studies confirm that transformer models outperform traditional techniques in accuracy and scalability. Overall, the literature shows a clear evolution from rule-based systems to advanced deep learning and transformer-based approaches, establishing a strong foundation for intelligent and efficient source code error classification systems.

Table: Summary of Key Literature Contributions and Their Impact on Current Research:

Author	Contribution	Impact on Research
A. C. Graesser (2012)	Introduced intelligent tutoring systems for adaptive learning.	Forms the base for automated and AI-based learning systems.
R. Romli (2015)	Improved automated programming assessment with better feedback.	Shows the need for clear and useful error explanations.
N. A. Rashid (2015)	Proposed a framework for automatic code assessment.	Helps in structuring error classification systems.
R. Yera (2017)	Developed recommendation-based programming systems.	Supports personalized error detection and learning.
M. M. Rahman (2020)	Used LSTM for source code error classification.	Shows deep learning can handle code patterns effectively.
Z. Li (2021)	Proposed deep learning model for error classification.	Supports multi-label error detection in programming.
Y.-T. Lin (2022)	Studied learning through expert coding behavior.	Connects learning process with error understanding.
M. M. Rahman (2022)	Applied data mining for programming learning analysis.	Supports use of datasets for training models.
Q. Chen (2022)	Used BERT for multi-label classification.	Shows transformers are effective for complex tasks.
M. F. I. Amin (2024)	Combined CodeT5 with ML models for classification.	Provides comparison baseline for advanced models.

III. PROPOSED APPROACH

Improving the accuracy and efficiency of source code error classification requires a structured pipeline that handles both data complexity and model performance. The approach begins with collecting a large dataset of source code samples paired with multiple error labels. Raw code is preprocessed to remove noise, normalize formatting, and retain meaningful syntactic structure. Each error category is converted into a numerical multi-label format so that the system can learn overlapping error patterns within the same code snippet.

The processed dataset is split into training and testing sets, ensuring a balanced distribution of error classes. Tokenization is then applied to transform source code into sequences compatible with transformer-based architectures. Care is taken to preserve context and code semantics during this step, since minor structural differences can significantly affect error interpretation.

A DistilBERT model is selected as the core classifier due to its reduced size and faster execution compared to traditional transformer models. The model is fine-tuned on the prepared dataset, enabling it to capture relationships between code tokens and corresponding error labels. A sigmoid-based classification layer is used to support multi-label prediction, allowing the model to assign multiple error types to a single input.

Training is performed with continuous monitoring of key performance metrics such as accuracy, precision, recall, and F1-score. Optimization techniques are applied to improve convergence and prevent overfitting. After training, the model is evaluated on unseen test data to validate its generalization capability.

The final stage involves integrating the trained model into a user-facing interface where source code can be uploaded and analyzed. The system outputs predicted error categories, providing quick and reliable assistance for debugging and learning.

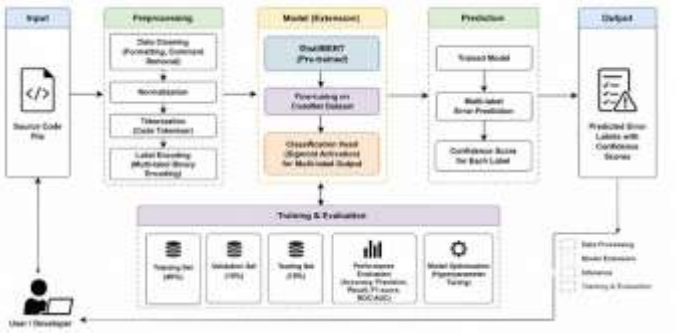

Figure 1: **Multi-Label Error Classification wrokflow**

IV. METHODOLOGIES

Algorithm: DistilBERT for Multi-Label Error Classification

Input:
D ← Dataset of source code samples with error labels
Output:
Trained model M
Predicted error labels for test data

Begin

1. Load Dataset D
2. Preprocess Data:
For each sample s in D:
Clean source code (remove comments, extra spaces)
Normalize code format
End For

3. Encode Labels:
Convert error labels into multi-label binary vectors Y

4. Split Dataset:
D_train, D_test ← Split(D, ratio = 80:20)

5. Tokenization:
For each sample s in D_train and D_test:
tokens ← Tokenize(s.code) using DistilBERT tokenizer
input_ids, attention_mask ← Generate inputs
End For

6. Initialize Model:
M ← Load pre-trained DistilBERT model
Add classification layer with sigmoid activation

7. Training Phase:
For epoch = 1 to N:
For each batch b in D_train:
inputs ← (input_ids, attention_mask)
labels ← Y_batch

outputs ← M(inputs)
loss ← BinaryCrossEntropy(outputs, labels)

Backpropagate loss
Update model parameters
End For
End For

8. Evaluation Phase:
For each sample t in D_test:
outputs ← M(t.inputs)
predictions ← Apply threshold (e.g., 0.5)
End For

9. Compute Metrics:
Calculate Accuracy, Precision, Recall, F1-score, ROC-AUC

10. Prediction Phase:
For new input code c:
c_processed ← Preprocess(c)
c_tokens ← Tokenize(c_processed)
output ← M(c_tokens)
predicted_labels ← Threshold(output)
End For

11. Return trained model M and predicted_labels

End

Dataset Selection and Understanding

A large-scale dataset containing source code samples and associated error labels is selected. The CodeNet dataset is used due to its diversity in programming problems and error categories. Each record includes a code snippet and one or more labels representing error types, making it suitable for multi-label classification tasks.

Data Cleaning and Preprocessing

Raw source code often contains noise such as unnecessary whitespace, comments, and inconsistent formatting. Preprocessing is applied to standardize the code, remove irrelevant symbols, and retain only meaningful syntactic elements. This step ensures that the model focuses on essential patterns rather than noise.

Multi-Label Encoding

Since a single code snippet can contain multiple errors, labels are transformed into a multi-label format using binary encoding. Each error type is represented as a vector position, allowing the system to predict multiple classes simultaneously instead of restricting to a single-label output.

Exploratory Data Analysis

The dataset is analyzed to understand the distribution of error types, frequency of labels, and imbalance among classes. Visualization techniques are used to identify dominant and rare error categories. This step helps in designing strategies to handle class imbalance and improve model learning.

Data Splitting Strategy

The dataset is divided into training and testing sets, typically using an 80:20 ratio. Stratified sampling is applied to maintain the distribution of error labels across both sets. This ensures that the model is trained on diverse patterns and evaluated fairly on unseen data.

Tokenization and Input Representation

Source code is converted into token sequences using a tokenizer compatible with transformer models. Special tokens are added to preserve contextual boundaries. Attention is given to maintaining the structure of code, as programming syntax differs significantly from natural language.

Model Selection and Extension Design

The baseline system uses BERT-based models for classification. The extension introduces DistilBERT, a compressed version of BERT that reduces the number of parameters while retaining most of its representational power. This design choice directly targets improved efficiency without sacrificing performance.

Model Fine-Tuning

The DistilBERT model is fine-tuned using the prepared dataset. A classification head with sigmoid activation is added to support multi-label output. During training, the model learns relationships between code tokens and corresponding error categories through backpropagation and optimization techniques.

Performance Evaluation Metrics

Model performance is evaluated using multiple metrics, including accuracy, precision, recall, F1-score, and ROC-AUC. These metrics provide a comprehensive understanding of classification performance, especially in multi-label scenarios where simple accuracy is insufficient.

Comparative Analysis with Baseline Models

The extended DistilBERT model is compared with baseline models such as BERT-Cased, CodeT5 with Decision Tree, and CodeT5 with Random Forest. The comparison focuses on

accuracy, computational efficiency, and execution time. Results demonstrate that the extension achieves higher accuracy while reducing resource requirements.

VI RESULTS & DISCUSSION

	Algorithm Name	Accuracy	Precision	Recall	FSCORE
0	CodeT5 Decision Tree	70.00	50.69	63.93	55.09
1	CodeT5 Random Forest	75.00	37.09	42.86	39.60
2	Propose Bert_Cased	92.12	93.21	77.90	83.15
3	Extension DistillBert_Cased	93.84	89.10	85.06	84.12

The experimental results clearly show the performance difference between traditional models, baseline transformer models, and the proposed extension. The CodeT5 combined with Decision Tree achieved an accuracy of 70.00%, with precision of 50.69%, recall of 63.93%, and F1-score of 55.09%. While this model captures basic patterns, its lower precision indicates a high number of false positives. Similarly, CodeT5 with Random Forest improved accuracy to 75.00%, but its precision dropped to 37.09% and F1-score to 39.60%, showing inconsistency in prediction quality despite better accuracy.

The baseline BERT-Cased model significantly outperforms these traditional combinations, achieving an accuracy of 92.12%, precision of 93.21%, recall of 77.90%, and F1-score of 83.15%. This indicates strong learning of contextual relationships in source code, especially in reducing false positives due to high precision.

The extension model, DistilBERT-Cased, delivers the best overall performance. It achieves the highest accuracy of 93.84%, along with improved recall of 85.06% and F1-score of 84.12%. Although its precision (89.10%) is slightly lower than BERT-Cased, the higher recall shows that it detects more relevant error labels. This balance results in better overall classification performance.

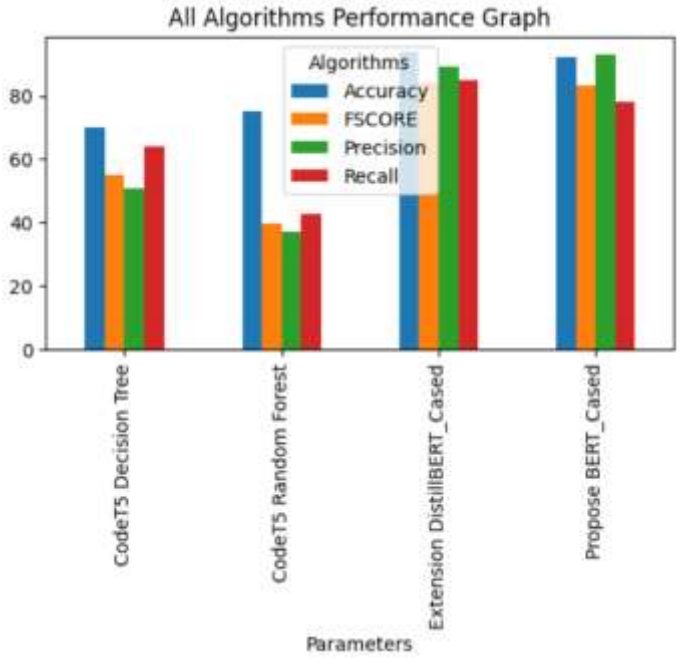

Figure 2: All Algorithms Performance Graph

The results show a clear pattern: classical ML pipelines struggle with multi-label code errors, while transformer models handle context much better. CodeT5 with Decision Tree and Random Forest produces unstable precision and low F1-scores, which means the models are not learning consistent error patterns. The Random Forest result is particularly weak, with precision dropping to 37.09%. That level of false positives makes it unreliable in any practical system.

The BERT-Cased model fixes most of these issues. Its precision of 93.21% proves it is highly confident in predictions, but the recall of 77.90% shows it still misses a portion of actual errors. This gap matters because missing errors is more damaging than slightly over-predicting them in debugging systems.

The DistilBERT extension improves exactly where BERT is weak. Recall increases to 85.06%, and the F1-score reaches 84.12%, which is the best balance across all metrics. The slight drop in precision to 89.10% is acceptable because it comes with a significant gain in error detection.

VII. CONCLUSION

The study demonstrates that accurate multi-label classification of source code errors requires models that can capture contextual relationships while maintaining computational efficiency. Traditional machine learning approaches show limited capability in handling complex error patterns, resulting in inconsistent performance. Transformer-based models significantly improve classification quality, with BERT-Cased achieving strong precision and overall accuracy. However, its computational cost limits practical deployment.

The extension using DistilBERT addresses this limitation by reducing model complexity while preserving performance.

Experimental results confirm that the extended model achieves higher accuracy (93.84%) and better recall, leading to improved overall F1-score. This balance ensures more reliable detection of multiple error types within a single code sample.

The findings confirm that lightweight transformer models can deliver both efficiency and accuracy, making them suitable for real-time programming support systems. This work provides a practical direction for developing scalable and intelligent code analysis tools.

REFERENCES

[1] Md. 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.-T. Lin, M. K.-C. Yeh, and S.-R. Tan, ''Teaching programming by revealing thinking process: Watching experts' live coding videos with reflection annotations,'' IEEE Trans. Educ., vol. 65, no. 4, pp. 617–627, Nov. 2022.

[3] M. Kaleem, M. A. Hassan, and S. K. Khurshid, ''A machine learningbased adaptive feedback system to enhance programming skill using computational thinking,'' IEEE Access, vol. 12, pp. 59431–59440, 2024.

[4] Md. 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] 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.

[6] 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.

[7] 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, Jun. 2015.

[8] 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 Proc. Adv. Comput. Commun. Eng. Technol. Cham, Switzerland: Springer, Dec. 2015, pp. 967–977.

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

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

[11] H. Wan, H. Luo, M. Li, and X. Luo, ''Automated program repair for introductory programming assignments,'' IEEE Trans. Learn. Technol., vol. 17, pp. 1705–1720, 2024.

[12] 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.

[13] Z. Li, F. Sun, H. Wang, Y. Ding, Y. Liu, and X. Chen, ''CLACER: A deep learning-based compilation error classification method for novice students' programs,'' in Proc. IEEE 45th Annu. Comput., Softw., Appl. Conf. (COMPSAC), Jul. 2021, pp. 74–83.

[14] M. F. I. Amin, A. Shirafuji, M. M. Rahman, and Y. Watanobe, ''Multilabel code error classification using CodeT5 and ML-KNN,'' IEEE Access, vol. 12, pp. 100805–100820, 2024. 3820 VOLUME 13, 2025 M. F. I. Amin et al.: Source Code Error Understanding Using BERT for Multi-Label Classification

[15] D. Chen and R. Zhang, ''COVID-19 vaccine adverse event detection based on multi-label classification with various label selection strategies,'' IEEE J. Biomed. Health Informat., vol. 27, no. 9, pp. 4192–4203, Sep. 2023.

[16] Q. Chen, J. Du, A. Allot, and Z. Lu, ''LitMC-BERT: Transformer-based multi-label classification of biomedical literature with an application on COVID-19 literature curation,'' IEEE/ACM Trans. Comput. Biol. Bioinf., vol. 19, no. 5, pp. 2584–2595, Sep. 2022.

[17] I. Ameer, G. Sidorov, H. Gómez-Adorno, and R. M. A. Nawab, ''Multilabel emotion classification on code-mixed text: Data and methods,'' IEEE Access, vol. 10, pp. 8779–8789, 2022

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

[19] U. Malik, S. Bernard, A. Pauchet, C. Chatelain, R. Picot-Clémente, and J. Cortinovis, ''Pseudo-labeling with large language models for multilabel emotion classification of French tweets,'' IEEE Access, vol. 12, pp. 15902–15916, 2024.

[20] 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. 20

SURISETTI SANDHYA Is Currently Pursuing The MCA (Master Of Computer Applications)In Ideal College Of Science And Technology Vidyuth Nagar Kakinada. Her Research Interests Include Machine learning

Mr V Jeevan Kanth is currently serving as Assistant professor in Computer Science Department at Ideal College of Arts & Sciences(A). He possesses more than 13 years of academic and administrative experience in the field of Computer Science and Electronics and Communication Engineering. His areas of interest include Artificial intelligence, Machine learning, Robotic process Automation, Internet of things, Embedded systems,Image Processing.

He completed his M.Tech in Electronics and Communication Engineering, Aditya Engineering College, Surampalem. Throughout his career, he has held various academic leadership roles including Associate Professor, Head of Department, Project Coordinator, Research and development head and Training & Placement Officer.

Dr. V. S. V. Deepak is currently serving as the Head of the Department of Computer Science at Ideal College of Arts & Sciences (A). He possesses more than 18 years of academic and administrative experience in the field of Computer Science and Engineering. His areas of interest include Medical Image Processing, Cyber Security, Artificial Intelligence, Software Testing and Networking. He completed his Ph.D. research in Medical Image Processing from Swami Vivekananda University.

He has actively contributed to curriculum development, academic planning, and student mentoring. He has served as Chairman of the Board of Studies (BOS) for BCA, B.Sc. (Computer Science), B.Sc. (Artificial Intelligence), and MCA programs.