Team Duo at SemEval-2026 Task 13: Fine-tuning CodeBERT for Out-of-Distribution AI-Generated Code Detection

Subhiksha G, Sanjai M, Rajalakshmi Sivanaiah, Angel Deborah Suseelan
Department of Computer Science and Engineering
Sri Sivasubramaniya Nadar College of Engineering, Chennai – 603110, Tamil Nadu, India
{subhiksha2410492, sanjai2410498, rajalakshmis, angeldeborahs}@ssn.edu.in

Abstract

This paper describes the system submitted to SemEval-2026 Task 13 Subtask A: Detecting AI-generated code in OOD settings. The task requires binary classification of code snippets as either human written or machine generated, with a strong emphasis on generalization to unseen programming languages and unseen code domains. This system fine-tunes `microsoft/codebert-base`, a transformer model pre-trained on code and natural language pairs. Analysis reveals that while the model performs near perfectly on the training distribution, it struggles significantly under OOD conditions due to language and domain shift.

1 Introduction

The rapid growth of the LLMs (Large Language Models) is capable of generating high-quality source code, such as Github Copilot, ChatGPT, and Code Llama, which has raised increasing concerns about the origin of code found in the repositories, student submissions and professional codebases of the production. Automatically detecting whether the code is generated by a human or generated by a machine poses an important and practically relevant challenge.

SemEval-2026 Task 13 Subtask A (Orel et al., 2026b) directly addresses this challenge by presenting binary classification under rigorously controlled OOD conditions. Unlike the prior work that evaluates detectors only on the same language and domain using the pretrained models, this task tests generalization in two ways: (i) programming language shift (training on C++, Python, Java; test on Go, PHP, C#, C and JavaScript) (ii) domain shift (training on algorithmic code; test on research code and production code directly from the GitHub).

This system fine-tunes `microsoft/codebert-base` (Feng et al., 2020), a transformer pre-trained bimodal natural language and code, which is hypothesized to transfer well to unseen languages owing to its broad pre-training corpus.

The main contributions are:

- CodeBERT is adapted for OOD binary detection of AI-generated code demonstrating the effectiveness of code-specific pre-training for this task.
- A detailed error analysis is presented identifying the key failure modes of the system under OOD conditions.

2 Background

SemEval-2026 Task 13 Subtask A (Orel et al., 2026b) is a binary classification task, with a given code snippet, predict whether that is human written or generated by machine. The evaluation emphasizes OOD generalization across four scenarios which includes: (i) seen language, seen domain (ii) seen language, unseen domain (iii) unseen language, seen domain and (iv) unseen language, unseen domain.

The training dataset contains code in C++, Python and Java from algorithmic programming problems. The test dataset contains code in unseen languages such as Go, PHP, C#, C and JavaScript and in unseen domains such as research-oriented and production-deployed code. Systems are evaluated using macro F1-score.

2.1 Related Work

Detecting machine generated code has received considerable attention after the era of AI has begun (Radford et al., 2019). Early methods relied mainly on statistical properties includes perplexity and burstiness (Gehrmann et al., 2019). More recent work enhances language models for detection, achieving strong performance but fails in domain shift (Bakhtin et al., 2019; Ippolito et al., 2020).

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2104–2107

Pan et al. (2024) evaluate existing AI detectors on student code submissions and highlight their limitations in educational settings. Simmons et al. (2024) conduct a literature mapping of AI-generated code plagiarism detection in computer science courses. Xu and Sheng (2024) propose a perplexity-based approach using CodeBERT to detect AI-generated code assignments. Bashir (2025) introduce a framework using pseudo-AI submissions to identify AI-generated student code without supervised training.

Yang et al. (2023) study zero shot detection and showed that existing tools struggle with programming languages beyond English. Suh et al. (2025) demonstrate that state-of-the-art detectors break down under OOD conditions, directly motivating this shared task. CodeBERT (Feng et al., 2020) is a pretrained bimodal model for programming and natural languages. It has been fine tuned, which makes it a strong candidate for AI generated code detection.

3 System Description

Figure 1 illustrates the overall architecture of the proposed system.

3.1 Pre-trained Model

`microsoft/codebert-base` is the base for training this system. CodeBERT is a transformer model with 12 layers, 768 hidden dimensions and 12 attention heads (125M parameters), pre-trained on a bimodal corpus of natural language - code pairs across six programming languages (Python, Java, JavaScript, PHP, Ruby, Go).

A two-class linear classification head is added on top of the `[CLS]` token representation. Given input tokens $\mathbf{x} = (x_1, \ldots, x_n)$,the model computes Equation 1:

$$\hat{y} = \text{softmax}(W \cdot \mathbf{h}_{\texttt{[CLS]}} + b) \tag{1}$$

where $\mathbf{h}_{\texttt{[CLS]}} \in \mathbb{R}^{768}$ is the pooled representation from the final encoder layer, and $W \in \mathbb{R}^{2\times768}$, $b \in \mathbb{R}^2$ are learned classification parameters.

3.2 Tokenization and Preprocessing

The CodeBERT tokeniser (byte-pair encoding, 50,265) with a maximum sequence length of 128 tokens is used. Code snippets exceeding 128 tokens are truncated from the right. No language-specific preprocessing (no stripping of comments, no identifier normalisation) is applied, since preserving surface-level patterns may serve as a useful signal for distinguishing human from machine-generated code.

3.3 Training Procedure

CodeBERT is trained for one epoch using the Hugging Face Trainer API. Mixed-precision training (`fp16=True`) is enabled throughout. Gradient checkpointing reduces peak GPU memory consumption by approximately 35%. AdamW (Loshchilov and Hutter, 2018) is used with a linear learning rate schedule, a warmup ratio of 0.06 and a learning rate of 5×10^{-5}.

4 Experimental Setup

4.1 Data

The official training and validation splits are used, which were provided by the task organisers (Orel et al., 2026b,a). Table 1 shows the distribution across splits.

Split	Human (0)	Machine (1)	Total
Train	238,475	261,525	500,000
Validation	47,695	52,305	100,000

Table 1: Dataset distribution across splits. Training and validation sets are slightly skewed toward machine-generated code.

The training set contains code in Java, C++ and Python from an algorithmic programming challenge. The validation set mirrors the training distribution. The held-out test set covers all four OOD evaluations scenarios described in Section 2. No external data or data augmentation was used in the primary submission.

4.2 Evaluation Metric

The primary evaluation metric is **macro F1-Score**, computed as shown in Equation 2:

$$\text{Macro F1} = \frac{1}{2} \sum_{c\in\{0,1\}} \frac{2 \cdot P_c \cdot R_c}{P_c + R_c} \tag{2}$$

where P_c and R_c are the precision and recall for class c, and classes 0 and 1 correspond to human-written and machine-generated code respectively. Accuracy, precision and recall are additionally reported on the training set for diagnostic purposes.

4.3 Baselines

The trained model is compared against the following baselines:

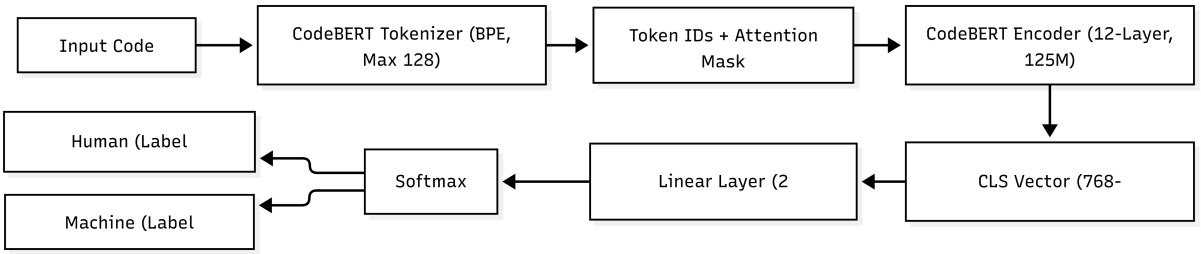

Figure 1: Architecture of the proposed CodeBERT-based classification system.

- **Majority class**: always predicts the majority label (human-written). Obtains F1 $\approx$ 0.33 (macro) by design.
- **TF-IDF + Logistic Regression**: character-level TF-IDF features (1-3 grams) with an ℓ_2-regularised logistic regression classifier.
- **TF-IDF + SVM**: same features with a linear SVM (penalty $C = 1$).

4.4 Implementation Details

All experiments used the Hugging Face `transformers` library (v4.x) and `datasets` library. Training is performed on a single NVIDIA Tesla T4 GPU (15 GB) via Kaggle notebooks. Checkpoints are saved every 2,600 steps (~ 40 minutes of training), and the best checkpoint (by the validation of the macro F1) is selected at the end of the epoch.

5 Results

Table 2 presents the results on the training and official test sets.

Split	Accuracy	F1	Precision	Recall
Training	0.9932	0.9935	0.9943	0.9927
Test Sample	0.2820	0.3532	0.2210	0.8789

Table 2: Results on training and official test sets. The large gap indicates significant OOD generalization challenges.

This system achieved a macro F1-score of 0.2509 on the official leaderboard test set. The model achieves near-perfect performance on the training distribution (F1 = 0.9935), but the sharp drop on the official test set reveals significant OOD generalization challenges. The model tends to over-predict the machine-generated class in OOD settings, suggesting high recall but low precision.

6 Analysis

6.1 OOD Generalization Analysis

This system achieves near perfect performance on training distribution (F1 = 0.9935) but drops sharply to F1 = 0.3532 on official test set with a leaderboard score of 0.2509. This severe degradation confirms that fine-tuning on a single epoch of the algorithmic code in three languages is insufficient for unseen languages and unseen domains.

The high recall (0.8789) but low precision (0.2210) on the test data set suggests that the model predicts machine generated class because CodeBERT's pretraining causes it to associate clean structured code with AI generation regardless of the actual source language and domain.

This behaviour is consistent with findings by Suh et al. (2025), who found that detectors trained on a narrow distribution fail under OOD conditions.

6.2 Error Analysis

The high recall (0.8789) and low precision (0.2210) indicate the model overwhelmingly predicts the machine generated code leading to a large number of false positives. The identified likely causes are:

- **Domain mismatch**: The model was fine-tuned only on algorithmic code so research and generic deployed code which follows different conventions, is misclassified as machine generated.
- **Unseen language syntax**: Code in Go, PHP, C#, C and JavaScript contains syntax patterns which were never seen during fine-tuning, causing the model to treat unfamiliar structure as a signal of AI generation.
- **Single epoch underfitting**: With only one epoch of fine-tuning, the model may not have learned a robust decision boundary, defaulting to majority prediction in uncertain cases.

7 Conclusion

This system is based on CodeBERT for SemEval-2026 Task 13 Subtask A, achieving a macro F1 score of 0.2509. While the model achieves near perfect performance on training distribution (F1 = 0.9935), the sharp drop on test set reveals the difficulty of OOD generalisation when both language and domain shift simultaneously.

8 Future Work

Future work will explore multi-epoch training, data augmentation across diverse programming languages and domains will be explored. Finally, larger code-specific models such as CodeT5+ or StarCoder could be explored as alternative backbones to CodeBERT.

Limitations

This system fine-tunes CodeBERT for only one epoch which likely contributes to poor OOD generalisation. Code snippets longer than 128 tokens are truncated, potentially losing important structural information. The model was trained on algorithmic code in three languages, limiting its exposure to diversity in test set. No data augmentation or multi-language training strategies were explored.

References

Anton Bakhtin, Sam Gross, Myle Ott, Yuntian Deng, Marc'Aurelio Ranzato, and Arthur Szlam. 2019. Real or fake? learning to discriminate machine from human generated text. *arXiv preprint arXiv:1906.03351*.

Shariq Bashir. 2025. Using pseudo-AI submissions for detecting AI-generated code. *Frontiers in Computer Science*, 7:1549761.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and 1 others. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547. Association for Computational Linguistics.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander M. Rush. 2019. GLTR: Statistical detection and visualization of generated text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 111–116. Association for Computational Linguistics.

Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, and Douglas Eck. 2020. Automatic detection of generated text is easiest when humans are fooled. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1808–1822. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026a. AICD bench: A challenging benchmark for ai-generated code detection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, Rabat, Morocco. Association for Computational Linguistics.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026b. SemEval-2026 task 13: Detecting machine-generated code with multiple programming languages, generators, and application scenarios. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Wei Hung Pan, Ming Jie Chok, Jonathan Leong Shan Wong, Yung Xin Shin, Yeong Shian Poon, Zhou Yang, Chun Yong Chong, David Lo, and Mei Kuan Lim. 2024. Assessing AI detectors in identifying AI-generated code: Implications for education. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. ACM.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Archer Simmons, Maristela Holanda, Christiana Chamon, and Dilma Da Silva. 2024. AI generated code plagiarism detection in computer science courses: A literature mapping. In *2024 IEEE Frontiers in Education Conference (FIE)*, pages 1–7. IEEE.

Hyunjae Suh, Mahan Tafreshipour, Jiawei Li, Adithya Bhattiprolu, and Iftekhar Ahmed. 2025. An empirical study on automatically detecting AI-generated source code: How far are we? In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)*.

Zhenyu Xu and Victor S. Sheng. 2024. Detecting AI-generated code assignments using perplexity of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 23155–23162.

Xianjun Yang, Kexun Zhang, Haifeng Chen, Linda Petzold, William Yang Wang, and Wei Cheng. 2023. Zero-shot detection of machine-generated codes. *arXiv preprint arXiv:2310.05103*.