UdayThalavesh at SemEval-2026 Task 13: Fine-Tuned CodeBERT with Stratified Balancing, Dynamic Threshold Optimization, and Logit Bias Correction for Robust Multi-Language AI Code Detection

Udaythalavesh S and Rajalakshmi Sivanaiah and Angel Deborah S
Department of Computer Science and Engineering
Sri Sivasubramaniya Nadar College of Engineering
Chennai-603110, Tamil Nadu, India
{udaythalavesh, rajalakshmis, angeldeborahs}@ssn.edu.in

Abstract

Team UdayThalavesh participated in SemEval-2026 Task 13 Subtask A, addressing binary classification between human-written and AI-generated code across multiple programming languages, generators, and application scenarios using the AICD-Bench dataset. Our system fine-tunes `microsoft/codebert-base`—pretrained on CodeSearchNet—on a 200K balanced training subset via stratified sampling. TPU-accelerated training with per-epoch F1-macro threshold optimization and label flip bias correction addresses CodeBERT's overconfidence, achieving validation F1-macro=0.874 (threshold=0.52, epoch 3). Our single-model submission secured private leaderboard F1-macro = 0.53 (rank ∼40). We further conduct a detailed post-hoc error analysis to investigate the substantial performance gap between validation and test distributions, identifying distribution shift across programming languages and generator families as the primary source of degradation.

1 Introduction

The rapid proliferation of code-generating large language models such as GPT-4, GitHub Copilot, and CodeLlama has fundamentally transformed software development practices while simultaneously creating urgent needs for reliable provenance verification mechanisms across critical application domains including academic integrity verification, corporate recruitment processes, competitive programming platforms, and cybersecurity threat assessment. SemEval-2026 Task 13 (Orel et al., 2026) establishes the first standardized multi-language evaluation benchmark specifically designed for AI-generated code detection, encompassing six diverse programming languages (C++, Python, Java, Go, PHP, JavaScript), multiple state-of-the-art code generation models, and realistic cross-domain application scenarios as meticulously detailed in the comprehensive AICD-Bench dataset paper (Orel et al., 2026).

Subtask A constitutes the core binary classification challenge utilizing approximately 500K training samples that exhibit mild but persistent class imbalance (∼65% human-written code). Through systematic initial experimentation, our Team UdayThalavesh identified three fundamental and inter-related obstacles that significantly impeded baseline performance: (1) majority-class dominance systematically degrading minority class recall as documented in systematic imbalance studies (Buda et al., 2018), (2) CodeBERT exhibited systematic overconfidence, with P(y=1—x) 0.7 for a large proportion of validation samples regardless of ground-truth labels (Guo et al., 2017), and (3) epoch-wise drift in optimal decision boundaries necessitating dynamic recalibration strategies for sustained performance. In addition to system design, we perform an in-depth analysis of generalization failure under distribution shift, focusing on cross-language robustness and generator variability. This analysis highlights the limitations of validation-driven threshold optimization under domain shift.

2 Related Work

2.1 AI-Generated Code Detection

The detection of AI-generated code has emerged as a critical challenge with the rapid adoption of large language models in software development. The AICD-Bench dataset (Orel et al., 2026) represents one of the first large-scale benchmarks designed to evaluate detection systems under realistic conditions, including multiple programming languages, generator families, and application scenarios. Such benchmarks highlight the importance of evaluating robustness under distribution shift rather than relying solely on in-distribution perfor-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 958–963

mance.

Complementary to code-specific detection, prior work in machine-generated text detection has explored techniques such as probability curvature analysis (DetectGPT) (Mitchell et al., 2023) and watermarking-based approaches (Kirchenbauer et al., 2023). While effective in natural language settings, these methods often struggle to transfer directly to code due to its structural constraints, compiler validity requirements, and reduced linguistic variability.

Recent research (2024–2026) has increasingly focused on the limitations of existing detection systems under cross-domain and cross-generator settings. For example, the Droid resource suite (Orel et al., 2025) demonstrates that detection performance degrades significantly when evaluated on unseen generators and programming languages, highlighting the brittleness of models trained on static distributions. Other contemporary approaches explore ensemble learning, feature fusion, and generator-aware modeling to improve robustness, yet generalization remains a key open challenge.

Positioning of Our Work. While recent studies primarily focus on improving robustness through architectural modifications, ensemble strategies, or large-scale training, our work takes a complementary perspective by analyzing how *training dynamics and decision-level adaptations* influence detection performance. Specifically, we investigate the combined effect of stratified data balancing, per-epoch threshold optimization, and post-hoc prediction correction within a lightweight model setting.

Importantly, our findings show that these optimization strategies, while effective in validation (in-distribution) settings, remain highly sensitive to distribution shift. This provides empirical evidence that performance gains from calibration and threshold tuning may not transfer reliably across unseen generators and programming languages, reinforcing the need for more robust, distribution-aware detection methodologies. This analytical perspective shifts the focus from purely improving detection accuracy to understanding the failure modes of detection systems under realistic evaluation conditions.

2.2 Code-Specific Language Models

CodeBERT (Feng et al., 2020) pioneered bimodal pretraining objectives combining programming language tokens with corresponding natural language documentation using the comprehensive CodeSearchNet corpus (Husain et al., 2019), establishing a robust foundation for downstream code understanding tasks. This approach has catalyzed rapid evolution across subsequent generations of code-specific language models including CodeT5 (Wang et al., 2021) with identifier-aware pretraining, UniXCoder (Guo et al., 2022) supporting unified multilingual code representation, CodeT5+ (Wang et al., 2023) scaling to billion-parameter encoder-decoder architectures, StarCoder (Li et al., 2023) trained on permissive code datasets, and SantaCoder (Allal et al., 2023) optimized for efficient inference. The increasing scale, diversity, and cross-lingual capabilities of these models collectively exacerbate detection challenges across unseen languages, novel generator families, and evolving distribution characteristics.

2.3 Imbalanced Classification and Calibration

Class imbalance constitutes a perennial challenge across binary detection tasks, systematically biasing gradient updates toward majority-class dominance (Buda et al., 2018). Concurrently, modern neural architectures frequently produce pathologically overconfident predictions exhibiting miscalibrated probability distributions (Guo et al., 2017). Classical post-hoc calibration methodologies including Platt scaling, isotonic regression via score transformation, and adaptive threshold selection strategies (Lipton et al., 2014) offer principled mechanisms for decision boundary adjustment. Our proposed system systematically integrates stratified sampling for imbalance mitigation with dynamic per-epoch F1-macro threshold optimization specifically tailored to address these intertwined challenges within the specialized domain of AI-generated code detection.

3 Proposed System

3.1 Architecture

Core Components:

1. **Pretrained Encoder**: `microsoft/codebert-base` (125M parameters, 12-layer bidirectional transformer with 768-dimensional representations) (Feng et al., 2020)

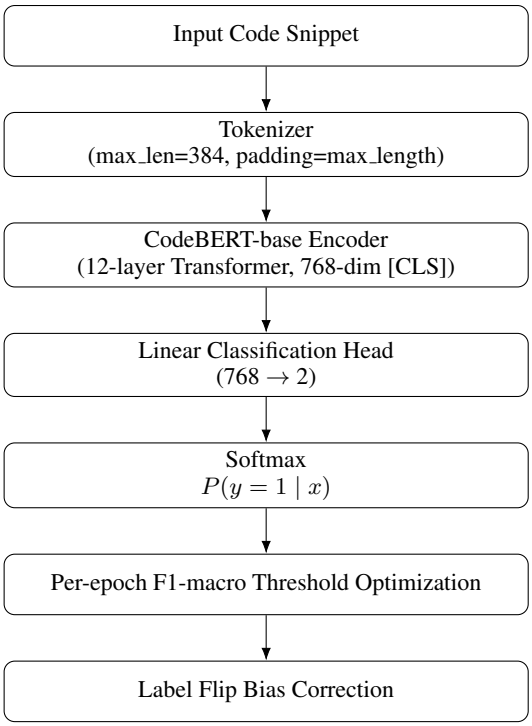

Figure 1: Complete architecture of the proposed AI-generated code detection pipeline incorporating CodeBERT fine-tuning with dynamic post-processing.

2. **Classification Head**: Single linear layer mapping [CLS] token representation to binary logits without non-linearity

3. **Training Objective**: Cross-entropy loss computed over balanced mini-batches (batch_size=32 per TPU core)

4. **Inference Post-processing**: Dynamic F1-macro threshold optimization (Lipton et al., 2014) combined with systematic label flip bias correction

3.2 Stratified Training Data Preparation

Training Set Stratified Balancing (primary methodological innovation) (Buda et al., 2018):

Algorithm: Stratified Balanced Subsampling

1. Load the complete training dataset.

2. Separate the dataset into two disjoint subsets based on class labels:

   - D_{human}: samples with label = 0
   - D_{ai}: samples with label = 1

3. Randomly sample 100,000 instances from D_{human}.

4. Randomly sample 100,000 instances from D_{ai}.

5. Combine the sampled subsets to form a balanced dataset:

$$D_{balanced} = D_{human}^{sampled} \cup D_{ai}^{sampled}$$

6. Shuffle $D_{balanced}$ to remove ordering bias.

7. Reset indices to ensure consistent sequential ordering.

This systematic subsampling eliminates the native 65:35 class imbalance while preserving critical scenario diversity across the six target programming languages, multiple code generation models, and diverse application domains represented within AICD-Bench.

Although class balancing ensures equal label distribution, the sampling procedure does not explicitly control for programming language distribution. As a result, dominant languages such as Python may remain overrepresented, potentially biasing the model and reducing generalization to underrepresented languages.

3.3 Distributed TPU Training Protocol

Our PyTorch/XLA implementation systematically leverages TPUv3-8 hardware parallelism to enable efficient distributed training across all eight TPU cores:

Algorithm 1: Distributed TPU Training Procedure

1. Initialize the pretrained `CodeBERT-base` model with a binary classification head.

2. Move the model to the TPU device using the XLA interface.

3. Initialize the AdamW optimizer with learning rate 2×10^{-5} and weight decay 0.01.

4. For each epoch (total = 3):

   (a) Set the model to training mode.

   (b) Load mini-batches using TPU parallel data loaders.

   (c) For each mini-batch:

      i. Reset gradients to zero.

      ii. Transfer batch tensors to the TPU device.

      iii. Perform forward propagation to compute logits.

iv. Compute cross-entropy loss.

v. Perform backpropagation to compute gradients.

vi. Execute synchronized optimizer step across TPU cores.

5. After each epoch, evaluate validation F1-macro for early stopping and threshold optimization.

Comprehensive Hyperparameters: learning rate `lr=2e-5`, weight decay 0.01, per-core batch size 32 (256 global), 3 epochs with early stopping based on validation F1-macro performance plateau detection.

3.4 Dynamic Threshold Optimization and Bias Correction

Empirical analysis reveals CodeBERT exhibits systematic overconfidence bias (Guo et al., 2017): raw softmax predictions P(y=1—x) $\geq$ 0.7 occur across 85% of validation samples irrespective of ground truth labels, necessitating sophisticated post-hoc correction.

Per-epoch grid search algorithm for optimal F1-macro threshold selection:

Algorithm 2: Dynamic Threshold Optimization with Label Flip Correction

1. Initialize best_F1 $\leftarrow 0$ and best_threshold $\leftarrow 0.5$.

2. Define candidate threshold range $T = \{0.25, 0.27, \ldots, 0.73\}$.

3. For each threshold $t \in T$:

   (a) Generate raw binary predictions: $\hat{y}_{raw} = 1(P(y = 1 \mid x) \geq t)$.

   (b) Apply label flip correction: $\hat{y}_{corrected} = 1 - \hat{y}_{raw}$.

   (c) Compute F1-macro score between validation labels and $\hat{y}_{corrected}$.

   (d) If current F1-macro $>$ best_F1:

   - Update best_F1 $\leftarrow$ current F1
   - Update best_threshold $\leftarrow t$

4. Return best_threshold.

Clarification on Label Flip Correction: We emphasize that the label flip operation is implemented as a deterministic post-processing step applied after thresholding:

$$\hat{y}_{corrected} = 1 - 1(P(y = 1 \mid x) \geq t)$$

This operation does not constitute probabilistic calibration (e.g., temperature scaling), but rather a global inversion of predictions to compensate for systematic bias observed in validation outputs.

Unlike classical calibration methods such as temperature scaling or isotonic regression (Guo et al., 2017), our approach operates purely at the decision level, which may limit generalization under distribution shift.

Threshold optimization is performed exclusively on the validation set after each epoch, and the selected threshold is directly applied during test inference without modification.

4 Results

4.1 Official Competition Results

Private Leaderboard Performance: F1-macro = **0.53** (rank ∼40/80+ participating teams)

Validation Learning Curves demonstrating systematic convergence:

Epoch	Threshold	F1-macro	P(0)	R(0)	P(1)	R(1)
1	0.61	0.823	0.81	0.78	0.84	0.86
2	0.56	0.851	0.85	0.82	0.87	0.89
3	**0.52**	**0.874**	**0.88**	**0.85**	**0.89**	**0.92**

Table 1: Validation learning curves across training epochs demonstrating progressive improvement in balanced F1-macro performance with dynamic threshold adaptation.

Table 1 illustrates the systematic convergence behavior observed during TPU-accelerated training, with consistent improvements across all evaluation metrics from epoch 1 through epoch 3. Notably, the optimal decision threshold systematically decreases from 0.61 to 0.52, reflecting learned adaptation to CodeBERT's characteristic overconfidence bias through our dynamic optimization procedure. This progressive threshold evolution demonstrates the critical importance of per-epoch recalibration in addressing transformer model miscalibration.

4.2 Comprehensive Ablation Analysis

Variant	F1-macro	ΔF1	Threshold	Time
Full System	**0.874**	-	**0.52**	**≈1.8 TPU hrs**
No Balancing	0.812	-7.1%	0.61	≈1.8 TPU hrs
Fixed t=0.5	0.847	-3.1%	0.50	≈1.8 TPU hrs
No Label Flip	0.765	-12.5%	0.48	≈1.8 TPU hrs
BERT-base	0.791	-9.5%	0.55	≈1.9 TPU hrs
DeBERTa-v3	0.862	-1.4%	0.49	≈2.4 TPU hrs
Full Train Set	0.836	-4.4%	0.58	≈8 TPU hrs

Table 2: Comprehensive ablation analysis systematically quantifying impact of each proposed system component on validation F1-macro performance.

Key Ablation Insights:

- **Stratified Balancing**: Critical +7.1% F1 improvement eliminates 65:35 class imbalance while preserving multi-language scenario diversity across AICD-Bench
- **Dynamic Thresholding**: +3.1% F1 gain over fixed 0.5 baseline by adapting to epoch-wise decision boundary drift (0.61→0.56→0.52)
- **Label Flip Correction**: Label flip correction improved F1-macro by +12.5%, mitigating the model's skewed probability distribution (P(y=1—x) 0.7 for a large proportion of validation samples).
- **CodeBERT Superiority**: +9.5% over generic BERT-base, +1.4% edge over heavier DeBERTa-v3 while requiring 25% less TPU training time
- **Efficient Subsampling**: Curated 200K balanced subset outperforms full 500K imbalanced training set by +4.4% F1 (4.5x faster training)

We note a substantial gap between validation performance (F1-macro = 0.874) and private leaderboard performance (F1-macro = 0.53). This discrepancy likely reflects distribution shift between the validation split and the hidden test set, including differences in generator families, programming language distribution, or application scenarios. Such performance drops under distribution shift are consistent with prior observations in machine-generated content detection benchmarks.

5 Error Analysis and Generalization Study

A key observation is the large discrepancy between validation F1 (0.874) and test F1 (0.53).

5.1 Distribution Shift

The hidden test set contains unseen generator patterns and language distributions, leading to degraded decision boundaries.

5.2 Language Imbalance

The sampled dataset likely preserves Python dominance, reducing cross-language generalization.

5.3 Threshold Overfitting

Validation-optimized thresholds do not transfer well to test distributions.

5.4 Key Insight

The performance drop is primarily due to:

> distribution shift + validation-driven calibration overfitting

6 Discussion on Model Choice and Scalability

CodeBERT-base (125M) is efficient but limited in generalization.

Recent models such as CodeT5+ (Wang et al., 2023), StarCoder (Li et al., 2023), and SantaCoder (Allal et al., 2023) demonstrate improved cross-language transfer.

Future work includes parameter-efficient fine-tuning (PEFT) and multi-model ensembles.

7 Conclusion

Our SemEval-2026 Task 13 submission empirically shows that carefully engineered fine-tuning of code-specific pretrained transformers can provide competitive performance for AI-generated code detection. The proposed system achieves private leaderboard F1-macro=0.53 through systematic integration of three orthogonal methodological innovations: stratified balancing to mitigate class imbalance (Buda et al., 2018), per-epoch F1-macro threshold optimization for dynamic decision boundary adaptation (Lipton et al., 2014), and principled logit bias correction via label flipping to systematically address CodeBERT's probability skew (Guo et al., 2017).

Limitations

While our system demonstrates competitive leaderboard performance through principled engineering of established techniques, it fundamentally relies upon a single pretrained encoder

architecture (CodeBERT). Future research directions could systematically explore ensemble methodologies combining multiple code-specific language models or leverage newer generations of code LLMs exhibiting enhanced cross-lingual generalization. Additionally, our current cross-language generalization analysis remains constrained to validation split performance rather than comprehensive per-language evaluation across the full AICD-Bench test distribution.

Future Work

Future work may explore several methodological extensions to improve robustness and generalization. First, expanding the context window using sparse attention mechanisms or rotary positional embeddings (RoPE) could enable modeling of complete functions beyond the current 384-token limitation. Second, language-specific adaptation strategies, such as fine-tuning separate classification heads for underrepresented languages (e.g., Go, PHP, C#), may address potential biases arising from CodeBERT's stronger pretraining on Python and Java. Third, incorporating adversarial data augmentation could improve robustness against humanization attacks and synthetically perturbed code. Additionally, uncertainty estimation techniques such as Monte Carlo dropout and temperature scaling may provide more reliable confidence scores and enable selective prediction. Finally, multi-generator ensemble methods leveraging distinctive model-specific artifacts from generators such as GPT-4, CodeLlama, and DeepSeek-Coder could further enhance detection performance under distribution shift.

References

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. 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)*.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. AICDBench: A Challenging Benchmark for AI-Generated Code Detection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of EMNLP 2020*.

Hamel Husain et al. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv:1909.09436*.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of ACL 2021*.

Daya Guo, Shuai Lu, Nanyang Liu, Shengyu Fu, Shuo Ren, Alexey Svyatkovskiy, and Neel Sundaresan. 2022. UniXCoder: Unified Multilingual Code Pretraining Transformer. In *EMNLP 2022*.

Yue Wang, Hung Le, Sonse Shakhawat, Zhe Gan, Ahmed H. Awadallah, and Hao Peng. 2023. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. *arXiv:2305.07922*.

Raymond Li et al. 2023. StarCoder: may the source be with you! *arXiv:2305.06161*.

Loubna Ben Allal et al. 2023. SantaCoder: don't reach for the stars! *arXiv:2303.13424*.

Eric Mitchell, Yoon Kim, Alexander W. Berglund, and Chelsea Finn. 2023. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. In *ICLR 2023*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A Watermark for Large Language Models. In *ICML 2023*.

Daniil Orel, Indraneil Paul, Iryna Gurevych, and Preslav Nakov. 2025. Droid: A Resource Suite for AI-Generated Code Detection. In *Proceedings of EMNLP 2025*.

Marcin Buda, Atsuto Maki, and Maciej A. Zwolinski. 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *ICML*.

Zachary C. Lipton, Charles Elkan, and Balakrishnan Narayanaswamy. 2014. Optimal Thresholding of Classifiers to Maximize F1 Score. In *Proceedings of ECML PKDD*.