UCSC-NLP at SemEval-2026 Task 13: Multi-View Generalization and Diagnostic Analysis of Machine-Generated Code Detection

Kargi Chauhan*
University of California, Santa Cruz
kchauha3@ucsc.edu

Sadiba Nusrat Nur*
University of California, Santa Cruz
sanur@ucsc.edu

Abstract

With the rapid growth of large language models for code generation, distinguishing between human-written and AI-generated code has become increasingly critical for academic integrity, hiring evaluations, and software security. We present our system for SemEval-2026 Task 13: Multilingual Machine-Generated Code Detection, participating in Subtask A (binary detection) and Subtask B (multi-class attribution across 10 LLM families). For Subtask A, we fine-tune UniXcoder-base with a multi-view training framework that promotes generator-invariant representations. The framework combines domain-specific structural prefixes, delexicalization with symmetric KL consistency loss, token dropout, and mixed-content augmentation. Our system achieves 0.993 macro F1 on validation and 0.845 macro F1 on the test set, which spans unseen languages and domains. For Subtask B, we show that severe class imbalance (88.4% human code, 221:1 majority-to-minority ratio) causes catastrophic minority-class failure under standard fine-tuning, with macro F1 collapsing to 0.086 despite 88.4% accuracy. A class-weighted extension trained for 3 epochs recovers macro F1 to 0.345 (+301% relative), confirming that multi-class attribution requires imbalance-aware training strategies.[1]

1 Introduction

Large language models (LLMs) have reshaped software development. AI-assisted tools such as GitHub Copilot, used by over 1.8 million developers (Dohmke, 2023), are now embedded in educational and professional workflows. While improving productivity, they raise concerns about authorship, academic integrity, intellectual property, and security auditing (Pearce et al., 2022). Consequently, distinguishing human-written from machine-generated code has become both critical and technically challenging, as detectors often overfit to generator-specific artifacts and degrade under prompt variation or minor edits (Pan et al., 2024; Liu et al., 2024). SemEval-2026 Task 13 (Orel et al., 2026) builds upon the DROID resource suite (Orel et al., 2025b), a multilingual, multi-generator benchmark designed to evaluate robustness under distribution shift. Prior work shows that detectors trained on limited languages or domains generalize poorly to unseen settings (Orel et al., 2025a; Pan et al., 2024). These issues are exacerbated in fine-grained attribution, where severe class imbalance leads to systematic neglect of minority generator classes. We frame this task as a domain generalization problem under lexical, structural, and stylistic shifts. To promote generator-invariant representations, we adopt: (i) domain-specific structural prefixes for content-type alignment (Ganin and Lempitsky, 2016), (ii) delexicalization with symmetric KL consistency regularization (Miyato et al., 2018), (iii) token dropout to reduce reliance on spurious cues (Wei and Zou, 2019), and (iv) mixed-content augmentation for distributional robustness (Arjovsky et al., 2019). We further show that multi-class attribution collapses under severe imbalance, where high accuracy masks near-complete minority-class failure. Our work makes two distinct contributions: (1) a multi-view generalization framework for robust binary detection under distribution shift (Subtask A), and (2) a diagnostic analysis demonstrating that standard fine-tuning catastrophically fails under extreme class imbalance in multi-class attribution (Subtask B).

2 Background and Related Work

2.1 Machine-Generated Code Detection

Early work on machine-generated code detection adapts text-based AI detection methods to programming languages. GPTSniffer (Nguyen et al., 2024) introduced CodeBERT-based classification

*Both authors contributed equally.

[1] https://github.com/Kargichauhan/Detecting-Machine-Generated-Code

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2461–2468

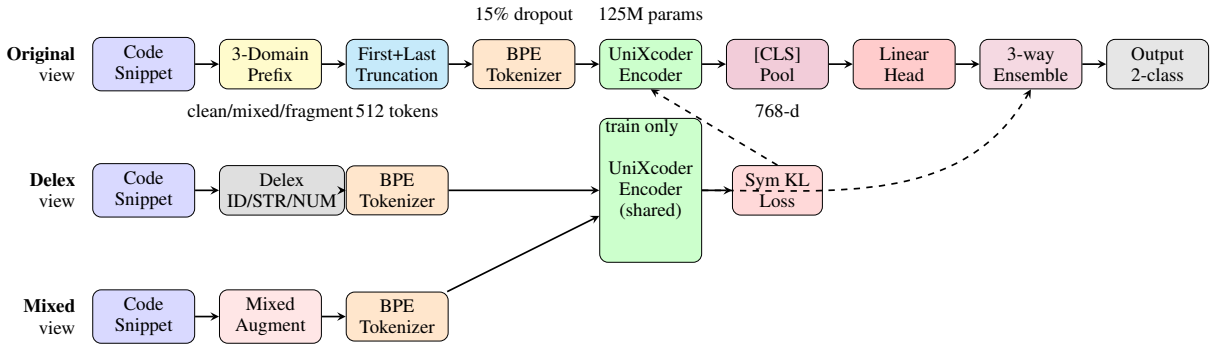

Figure 1: Subtask A architecture. Three views of each input are processed through a shared UniXcoder encoder; symmetric KL loss enforces prediction consistency across views during training, while inference averages logits as a zero-cost ensemble.

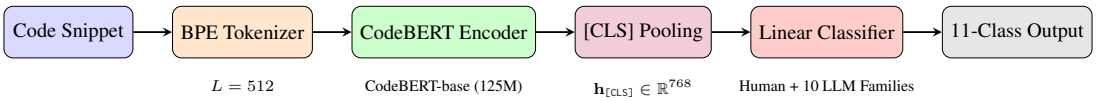

Figure 2: Subtask B architecture. A standard CodeBERT encoder with softmax classification head serves as a diagnostic baseline to isolate the effect of class imbalance on 11-way generator attribution.

and achieved strong performance on ChatGPT-generated code, but showed limited generalization across generators. CodeGPTSensor (Xu et al., 2025) mitigated this using contrastive learning to better separate human-written and machine-generated representations.

Prior studies highlight the difficulty of the task. General-purpose AI detectors perform near chance level on code (Pan et al., 2024), and learned stylistic cues are often generator-specific and brittle to prompt variation or post-editing (Liu et al., 2024). Alternative signals, such as entropy-based detection (Mitchell et al., 2023), repetition statistics (Gehrmann et al., 2019), and likelihood-based methods (Kirchenbauer et al., 2023), are largely designed for natural language and degrade on source code due to its syntactic regularity. Recent benchmarks (Wang et al., 2024) therefore emphasize robustness across programming languages and generators as a key challenge.

2.2 Pretrained Code Models

CodeBERT (Feng et al., 2020) introduced bimodal pretraining on natural language and code, enabling strong downstream classification. Subsequent models incorporate richer structural signals: GraphCodeBERT (Guo et al., 2021) integrates dataflow graphs, while UniXcoder (Guo et al., 2022) unifies encoder-only, decoder-only, and encoder–decoder objectives within a structure-aware pretraining framework, demonstrating strong cross-task transfer.

Sequence-to-sequence approaches such as CodeT5 (Wang et al., 2021) and PLBART (Ahmad et al., 2021) further extend pretraining for code generation and understanding. Despite increasing architectural complexity, encoder-based models remain competitive for classification under moderate computational budgets (Kanade et al., 2020).

3 Dataset Analysis

3.1 Subtask A: Cross-Lingual Generalization

The binary dataset exhibits near-ideal label balance (47.7% human, 52.3% machine) but presents a critical language and domain mismatch between training and evaluation. Training is restricted to Python (91%), C++, and Java, all from the algorithmic domain. The test set, however, spans unseen languages (JavaScript, C#, PHP, Go, C) and unseen domains (Research, Production), with 33.4% of samples being mixed-content snippets of unknown type. This train–test distribution shift, rather than label imbalance, is the central challenge of Subtask A (Table 1).

3.2 Subtask B: Severely Imbalanced Multi-Class

Table 2 reveals the core challenge of Subtask B: the training data is dominated by human-written code (442K samples, 88.4%), while machine-generated samples are spread across 10 LLM families with counts ranging from 2K to 10K. The imbalance

Table 1: Subtask A language and domain distribution. The test set introduces five unseen languages and two unseen domains absent from training.

Split	Lang.	Domain	%
Train	Python	Algo.	91%
	C++	Algo.	5%
	Java	Algo.	4%
Test	Python	Algo.	28.3%
	JavaScript	Mixed	13.3%
	Java	Mixed	12.0%
	C#	Res./Prod.	5.4%
	PHP	Res./Prod.	5.4%
	C++	Algo.	1.7%
	Go	Res./Prod.	0.5%
	Unknown	Mixed	33.4%

Table 2: Subtask B class distribution. The 221:1 imbalance ratio between the majority class and the smallest minority classes poses a fundamental optimization challenge.

Class	Samples	% of Train
Human	442,000	88.4%
OpenAI	10,000	2.0%
Meta-LLaMA	8,000	1.6%
IBM-Granite	8,000	1.6%
Qwen	8,000	1.6%
Phi	5,000	1.0%
DeepSeek-AI	4,000	0.8%
Mistral	4,000	0.8%
01-ai	3,000	0.6%
BigCode	2,000	0.4%
Gemma	2,000	0.4%
Total	**500,000**	**100%**

ratio between the largest class (Human) and the smallest (BigCode, Gemma) reaches **221:1**, fundamentally biasing gradient-based optimization toward the majority class. Standard cross-entropy training under this distribution causes the model to effectively predict "human" for the vast majority of inputs regardless of true label.

4 Methodology

4.1 Subtask A: Multi-View Generalization Framework

Our initial CodeBERT baseline achieved near-perfect validation F1 but failed to generalize to the test set due to the language and domain mismatch described in Section 3.1. We address this by fine-tuning UniXcoder-base (Guo et al., 2022) with a multi-view training framework (Figure 1). We introduce four targeted modifications, each addressing a specific failure mode identified in Section 3.1.

3-Domain Structural Prefix. The test set contains 33.4% mixed-content snippets entirely absent from training, causing the model to apply incorrect decision boundaries to unseen content types. We address this by prepending a structured prefix classifying each snippet into `clean`, `mixed`, or `fragment` based on the ratio of syntactic to natural-language lines, along with structural features (loops, functions, classes).

Delexicalization with Symmetric KL Consistency Loss. Lexical surface cues — variable names, library identifiers, string constants — vary systematically across programming languages and do not transfer to unseen domains. We replace all identifiers, string literals, and numeric literals with generic tokens (`ID`, `STR`, `NUM`) and enforce prediction consistency between the original and delexicalized views via symmetric KL divergence.

Token Dropout. Language-specific keywords act as spurious features that the model can exploit on the training distribution but that do not generalize to unseen languages. We randomly replace 15% of non-special tokens with `[MASK]` during training, simulating exposure to unknown syntax.

Mixed-Content Augmentation. Training data contains no mixed-content samples, yet 33.4% of test samples are mixed code–text snippets of unknown type. We randomly inject generic text fragments into training snippets at 40% probability with a softer consistency loss, directly bridging this distributional gap. The combined training loss is:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \cdot \text{SymKL}(p \| q_{\text{delex}}) + \frac{\lambda}{2} \cdot \text{SymKL}(p \| q_{\text{mixed}}) \quad (1)$$

At inference, logits from all three views (original, delexicalized, mixed) are averaged as a zero-cost test-time ensemble.

4.2 Subtask B: Diagnostic Baseline and Class-Weighted Extension

Subtask B is formulated as an 11-class classification problem, where each code snippet is labeled as either human-written or generated by one of 10 LLM families. We fine-tune `microsoft/codebert-base` (Feng et al., 2020) with a linear softmax classification head over the pooled `[CLS]` representation ($\mathbf{h}_{\texttt{[CLS]}} \in \mathbb{R}^{768}$), using a maximum sequence length of 512 tokens (Figure 2).

We present two systems. First, an **unweighted diagnostic baseline** that deliberately avoids class reweighting, focal loss, or oversampling to isolate

Hyperparameter	Task A	Task B
Backbone	UniXcoder-base	CodeBERT-base
Max sequence length	512 (first+last)	512
Learning rate	2×10^{-5}	2×10^{-5}
Batch size	16	32
Epochs	1	3
Warmup steps	470	500
Weight decay	0.01	0.01
Label smoothing	0.1	—
KL weight λ	0.5	0
Token dropout rate	15%	0
Class weighting	0	Inv.-freq.

Table 3: Training hyperparameters. Task A uses a single epoch with multi-view regularization; Task B trains for 3 epochs, with the competitive system applying inverse-frequency class weighting.

Table 4: Subtask A results. The val–test gap reflects cross-lingual and cross-domain distribution shift, not overfitting.

Split	Accuracy	Macro F1	AUC
Validation	0.993	0.993	0.999
Test	0.892	0.845	0.851

the impact of extreme label skew the divergence between macro F1 and accuracy directly measures minority-class collapse. Second, a **class-weighted competitive system** that applies inverse-frequency weighting to the cross-entropy loss, upweighting each minority LLM class proportionally to its underrepresentation. Both systems are trained for 3 epochs.

4.3 Training Configuration

Table 3 summarizes hyperparameters for both subtasks. For Subtask A, we use first+last truncation (concatenating the first and last 256 tokens to form a 512-token input) to preserve both opening signatures and closing patterns.

5 Results

5.1 Subtask A: Binary Classification

Table 4 presents our Subtask A results. The system achieves 0.993 macro F1 on the in-distribution validation set and 0.845 on the out-of-distribution test set. The 14.8-point gap reflects the difficulty of cross-lingual and cross-domain generalization, though the test performance remains strong given that training uses only three languages in a single domain. Per-class test performance is balanced: Human (P = 0.920, R = 0.760, F1 = 0.833) and Machine (P = 0.794, R = 0.938, F1 = 0.860), indicating no systematic bias toward either class.

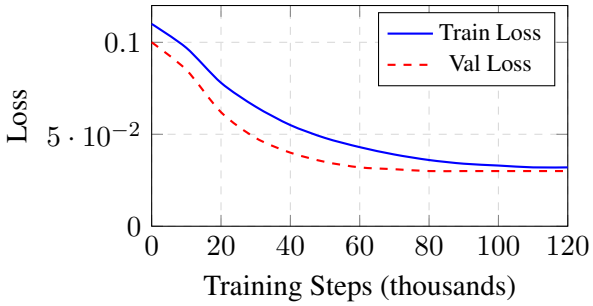

Figure 3: Subtask A training dynamics. Validation loss closely tracks training loss, indicating stable convergence without overfitting across 120K steps.

Figure 3 confirms healthy training dynamics: validation loss tracks training loss throughout optimization with no divergence, indicating that the multi-view regularization framework prevents overfitting despite training for only a single epoch.

5.2 Subtask B: Multi-Class Attribution

Table 5: Subtask B results. The diagnostic baseline exposes majority-class collapse; the class-weighted system recovers macro F1 by 301% relative.

System	Sp.	Acc	Mac. F1	Wtd. F1
Diagnostic base.	Val	0.884	0.086	0.876
	Test	0.884†	0.086†	0.876†
Class-weighted	Val	0.787	0.345	0.836

†Identical to val; see §5.2 for explanation.

The stark contrast between accuracy (88.4%) and macro F1 (0.086) in Table 5 reveals systematic minority-class failure. The identical validation and test metrics are not coincidental: with both splits sharing the same 88.4% human proportion, a near-constant majority predictor yields identical aggregate scores on each split, confirming that no genuine per-sample discrimination is occurring. A naive majority-class baseline achieves identical accuracy at macro F1 = 0.091, meaning the unweighted model *performs below* the trivial baseline on the metric that matters.

Class-Weighted System. Applying inverse-frequency class weighting and training for 3 epochs substantially improves macro F1 to 0.345 (+301% relative) at the cost of majority-class accuracy (78.7%). This confirms that imbalance remediation is both necessary and achievable. The accuracy decrease reflects the model abandoning its majority-class bias each point of accuracy lost corresponds to an LLM-generated sample now correctly attributed rather than collapsed

into the human class. More aggressive strategies such as focal loss, contrastive pretraining, and class-balanced sampling are discussed in Section 7 as the most promising next steps.

6 Analysis

6.1 Embedding Space Visualization

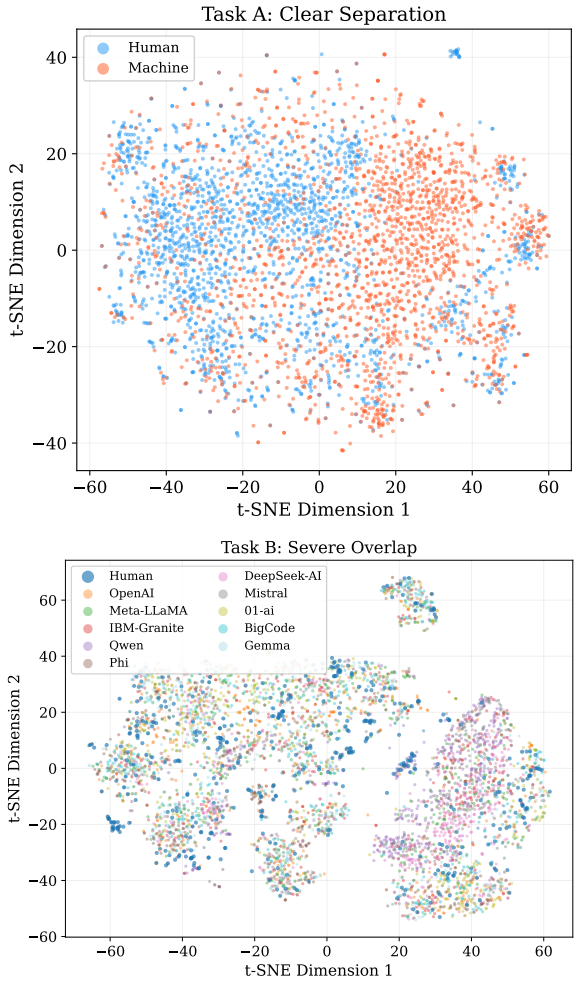

Figure 4: Empirical t-SNE projections (perplexity=30, seed=42) of pretrained `[CLS]` embeddings extracted from held-out validation samples (2,000 per class for Task A; up to 500 per class for Task B). **Top**: pretrained UniXcoder embeddings for Task A form separable human/machine clusters. **Bottom**: pretrained CodeBERT embeddings for Task B show extensive class overlap, illustrating why fine-grained generator attribution collapses under standard fine-tuning.

Figure 4 provides empirical insight via t-SNE projections of pretrained `[CLS]` embeddings, UniXcoder for Task A and CodeBERT for Task B extracted from held-out validation samples.

Table 6: Ablation study on validation sets. Sequence length significantly impacts Task A; class weighting with full training yields the largest Task B improvement (+301% relative macro F1) at the cost of majority-class accuracy.

Configuration	Acc	M-F1	ΔF1
Subtask A – Validation (Binary)			
Full system (512 tok)	99.3	99.3	—
− 256 tokens	99.0	99.0	−0.3
− 128 tokens	98.1	98.1	−1.2
− 64 tokens	94.7	94.6	−4.7
− No KL loss	98.8	98.8	−0.5
− No warmup	99.1	99.1	−0.2
Subtask B – Validation (Multi-class)			
Full system (512 tok)	88.4	8.6	—
− 256 tokens	88.2	8.2	−0.4
− 128 tokens	87.9	7.8	−0.8
− 1 epoch	88.3	7.9	−0.7
+ Class weights (3 ep.)	78.7	34.5	+25.9

Task A. Human and machine code form distinct, well-separated clusters, consistent with the near-perfect validation classification.

Task B. All classes overlap substantially. LLM families are not only indistinguishable from each other but also from human code in the embedding space, suggesting that CodeBERT's representations lack the granularity for fine-grained generator attribution.

6.2 Why Does LLM-Generated Code Look Similar?

We hypothesize three contributing factors. First, **shared training data**: most LLMs train on overlapping code corpora (GitHub, StackOverflow), learning comparable patterns. Second, **architectural homogeneity**: most generators use decoder-only transformer architectures with similar tokenization and generation strategies. Third, **problem constraints**: competitive programming solutions have limited valid approaches, reducing inter-generator stylistic variance.

6.3 Ablation Study

Table 6 reveals several insights from validation-set experiments:

Sequence Length. Reducing from 512 to 64 tokens drops Task A macro F1 by 4.7 points, confirming that stylistic signals are distributed across moderate-length contexts and cannot be captured from short prefixes alone.

Example 1: Human → Machine (False Positive)

```
def solve(n, arr):
    return sorted(arr)[:n//2]
```

Analysis: Unusually concise human code misclassified as AI-generated due to lack of comments and one-liner style.

Example 2: Machine → Human (False Negative)

```
# This function calculates...
def calc_sum(numbers):
    total = 0 # initialize
    for num in numbers:
        total += num
    return total
```

Analysis: GPT-4 output with verbose inline comments mimics human tutorial style, evading detection.

Figure 5: Representative misclassifications. Both errors arise from style transfer: the model relies on surface-level stylistic patterns rather than deeper structural signals.

KL Consistency Loss. Removing the symmetric KL regularization decreases validation F1 by 0.5 points, with a larger expected impact on out-of-distribution test performance since this component specifically targets cross-lingual robustness.

Class Weighting Trade-off. Inverse-frequency class weighting trained for 3 epochs improves Task B macro F1 from 8.6 to 34.5 (+301% relative) but reduces accuracy from 88.4% to 78.7%. This confirms that imbalance remediation is both necessary and effective: the accuracy decrease reflects the model abandoning its majority-class bias in favour of genuine multi-class discrimination.

6.4 Error Analysis

Figure 5 presents characteristic errors, both arising from *style transfer*: concise human code resembles AI-typical efficiency, while verbose AI output mimics human pedagogical patterns. This confirms that the detector relies primarily on surface-level stylistic features, motivating the incorporation of deeper structural signals (ASTs, control flow) in future work.

7 Discussion and Future Work

Our results reveal two distinct regimes in machine-generated code detection. Binary human-vs-machine classification is highly effective: pre-trained code models capture fundamental distributional differences, and our multi-view regularization framework achieves strong generalization even under severe language and domain shift. Fine-grained generator attribution, however, remains an open challenge. The embedding-space overlap across LLM families (Figure 4) combined with extreme class imbalance creates a setting where standard fine-tuning is insufficient.

Several directions appear promising for addressing attribution. Contrastive learning methods (Xu et al., 2025) that explicitly maximize inter-class separation may overcome embedding overlap. Incorporating structural features such as AST patterns and control flow graphs (Allamanis et al., 2018; Guo et al., 2021) could capture generator-specific signatures invisible at the token level. Focal loss (Lin et al., 2017) provides a more principled alternative to class weighting by dynamically downweighting easy majority-class examples; given that class weighting alone yields +301% relative macro F1, focal loss would be expected to offer further improvement. Class-balanced sampling and few-shot meta-learning may additionally improve minority-class representations given the limited per-class training data.

Limitations

Our work has four limitations. First, we evaluate only encoder-based architectures (UniXcoder, CodeBERT); decoder-based or encoder–decoder models such as CodeT5 may exhibit different generalization properties for both subtasks. Second, Subtask B exploration remains incomplete: while inverse-frequency class weighting improves macro F1 by 301% relative, techniques such as focal loss, contrastive pretraining, and class-balanced sampling are unexplored and would likely yield further gains. Third, the Python-dominant training set (91%) limits our cross-language generalization claims, and performance on truly low-resource languages such as Go (0.5% of test) remains undertested. Finally, both training and evaluation samples originate predominantly from competitive programming, leaving generalization to production codebases, library code, and multi-file projects unknown.

Ethics Statement

This work targets high-stakes contexts such as academic integrity and hiring evaluation. our system's 0.760 human recall (Table 4) indicates that human review is essential before any consequential decision. False positives may disproportionately affect developers whose coding style diverges from the training distribution. We release our code to encourage reproducible, transparent, and responsible deployment.

References

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *International Conference on Learning Representations*.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Thomas Dohmke. 2023. GitHub Copilot: The world's most widely adopted AI developer tool. GitHub Blog.

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 the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547. Association for Computational Linguistics.

Yaroslav Ganin and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*.

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 (ACL)*.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7212–7225. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Aditya Kanade, Petros Maniatis, Gogul Balakrishnan, and Kensen Shi. 2020. Learning and evaluating contextual embedding of source code. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, and Tom Goldstein. 2023. A watermark for large language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988.

Yixin Liu, Yida Chen, Yang Liu, and 1 others. 2024. Detecting machine-generated text: A survey of methods and challenges. *Transactions of the Association for Computational Linguistics*.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. DetectGPT: Zero-shot machine-generated text detection using probability curvature. In *International Conference on Machine Learning*, pages 24950–24962. PMLR.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. 2018. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Massimiliano Di Penta. 2024. GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. *Journal of Systems and Software*, 214:112059.

Daniil Orel, Dilshod Azizov, and Preslav Nakov. 2025a. CoDet-M4: Detecting machine-generated code in multi-lingual, multi-generator and multi-domain settings. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10570–10593. Association for Computational Linguistics.

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)*. Association for Computational Linguistics. To appear.

Daniil Orel, Indraneil Paul, Iryna Gurevych, and Preslav Nakov. 2025b. Droid: A resource suite for AI-generated code detection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31263–31289. 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 *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 1–11. IEEE Computer Society.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. IEEE.

Yifan Wang, Yujie Li, Rui Zhang, and Minghao Chen. 2024. Evaluating the robustness of ai-generated content detection. In *Proceedings of the 2024 Confer-*

ence on Empirical Methods in Natural Language Processing (EMNLP).

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 the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. In *EMNLP*.

Xiaozhou Xu, Cuiyun Gao, and Michael R. Lyu. 2025. Distinguishing LLM-generated from human-written code by contrastive learning. *ACM Transactions on Software Engineering and Methodology*.