CUET_Luminaries_0227 at SemEval-2026 Task 13: Invariance-Oriented Representation Learning for Robust AI-Generated Code Detection

Shiti Chowdhury and **Adnan Faisal**
Department of Computer Science and Engineering
Chittagong University of Engineering and Technology, Bangladesh
shitichowdhury21@gmail.com, ajfaisal1208023@gmail.com

Abstract

Large language models increasingly generate high-quality source code, making reliable detection of machine-generated code essential for maintaining authorship integrity and software accountability. However, detection systems often degrade under distribution shift, particularly across programming languages and application domains. SemEval-2026 Task 13 Subtask A addresses this challenge through a structured OOD evaluation framework that assesses binary machine-generated code detection across unseen languages and application domains. To mitigate this limitation, we propose a robustness-oriented framework that enhances feature-fused UniXcoder representations with supervised contrastive learning, adversarial language-invariant training and uncertainty-aware filtering to promote stable and shift-resilient representations. Our proposed system achieves a macro-F1 of 0.5411 on the official test set and maintains stable performance under severe language–domain shift. Our results demonstrate that domain-level semantic variation is the primary source of degradation under distribution shift, reinforcing the importance of invariance-oriented representations for stable OOD performance[1].

1 Introduction

Large language models (LLMs) are central to code generation, enhancing productivity but raising concerns about authorship, academic integrity and software accountability (Weber et al., 2024). The inability to distinguish human-written from machine-generated code can lead to unfair evaluation, skill misrepresentation and security risks (Bukhari et al., 2024; Pearce et al., 2025), motivating reliable detection. Prior work is often limited to a few languages or in-distribution settings (Xu et al., 2025; Yang et al., 2023). In practice, detection must generalize across unseen languages and domains, as reliance on surface patterns degrades under distribution shift, underscoring the need for robustness-oriented approaches.

SemEval-2026 Task 13: Detecting Machine-Generated Code with Multiple Programming Languages, Generators and Application Scenarios, Subtask A: Binary Machine-Generated Code Detection evaluates AI-generated code detection under explicit OOD settings (Orel et al., 2026b), emphasizing robustness across unseen languages and domains. To meet these requirements, we develop a robustness-focused detection framework built on a UniXcoder backbone. Our approach integrates supervised contrastive learning to strengthen representation separability, adversarial invariance to reduce reliance on language- and generator-specific cues and uncertainty-aware filtering to mitigate noisy or ambiguous training instances. This Out-of-Distribution(OOD)-oriented training strategy improves cross-language generalization. Our system achieves a macro F1-score of 0.5411 on the official evaluation, indicating competitive performance under distribution shift. In this work, we aim to advance robust AI-generated code detection under OOD settings. Our core contributions are as follows:

- We have proposed an OOD-focused detection framework that unifies contrastive learning, adversarial invariance and uncertainty-aware training to improve cross-language generalization.
- We have performed systematic robustness analysis across language and domain shifts, identifying the components that most effectively enhance stability under distribution shift.
- We have developed a reproducible implementation to promote transparency and support future research on AI-generated code detection.

[1] OOD AI Code Detection SemEval2026

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 567–572

The task data and implementation are accessible at OOD AI Code Detection SemEval2026.

2 Background and Task Overview

SemEval-2026-Task13-Subtask-A defines AI-generated code detection as a binary classification task (Orel et al., 2026b), where each code snippet is labeled as human-written (0) or machine-generated (1). Unlike conventional benchmarks, the task explicitly evaluates robustness under cross-language and cross-domain out-of-distribution (OOD) shifts.

2.1 Out-of-Distribution Evaluation Setting

Figure 1 illustrates the four evaluation regimes derived from seen and unseen language–domain combinations. The Unseen Language + Unseen Domain setting constitutes the most challenging scenario, requiring strong generalization beyond the training distribution.

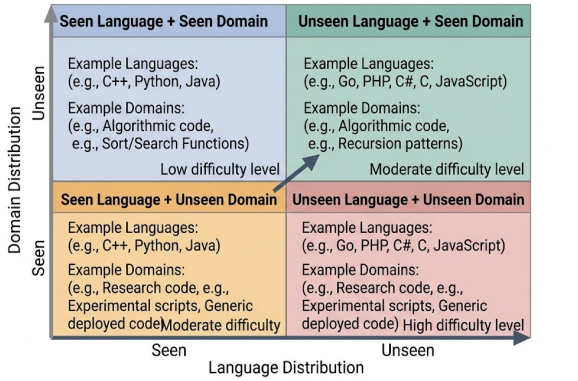

Figure 1: Out-of-distribution (OOD) evaluation quadrants defined by seen and unseen language–domain combinations. The Unseen Language + Unseen Domain regime represents the most challenging generalization scenario.

2.2 Evaluation Metric

Performance is measured using macro-F1:

$$\text{Macro}-\text{F1} = \frac{1}{2}\sum_{c}\frac{2P_cR_c}{P_c+R_c}. \quad (1)$$

where $c \in \{\text{human}, \text{AI}\}$.

3 Related Work

Recent work on AI-generated code detection increasingly prioritizes robustness across programming languages and application domains. This challenge is systematically operationalized in SemEval-2026-Task13 through explicitly defined out-of-distribution (OOD) evaluation regimes (Orel et al., 2026b). Complementing this line of work, AICD Bench advances large-scale benchmarking for cross-language detection under distribution shift.(Orel et al., 2026a).

Multi-generator detection resources such as Droid and CoDet-M4 highlight the difficulty of generalizing to unseen languages and adversarially refined outputs (Orel et al., 2025b,a). Transformer-based code encoders, including UniXcoder (Guo et al., 2022) and CodeT5+ (Wang et al., 2023), have demonstrated strong semantic representation capabilities for downstream tasks. Recent contrastive and zero-shot detection approaches further explore representation-level separation for machine-generated codes (Yang et al., 2023; Pham et al., 2025). However, robust generalization under cross-language and cross-domain shifts remains an open challenge. Our work directly addresses this limitation within the OOD-focused SemEval evaluation framework.

4 System Overview

Figure 2 shows the Feature-Fused UniXcoder framework. Given a code snippet x, UniXcoder produces h_{enc}, which is fused with structural features h_{feat} to form $h = [h_{enc}; h_{feat}]$, followed by classification:

$$\hat{y} = \text{softmax}(Wh + b).$$

Robustness is enforced via

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_1\mathcal{L}_{SupCon} + \lambda_2\mathcal{L}_{Adv},$$

where $\mathcal{L}_{CE}$ provides supervision, $\mathcal{L}_{SupCon}$ improves separability and $\mathcal{L}_{Adv}$ promotes invariance under distribution shift. At inference, only the encoder and classifier are utilized.

5 Methodology

Robustness under distribution shift is achieved through a unified objective integrating feature fusion, contrastive learning, adversarial invariance, and uncertainty-aware training.

5.1 Feature-Fused Representation

Given a code snippet x, contextual representations are obtained from a UniXcoder backbone and enriched with complementary structural features to form a unified embedding for binary classification. The fused representation is passed through a linear

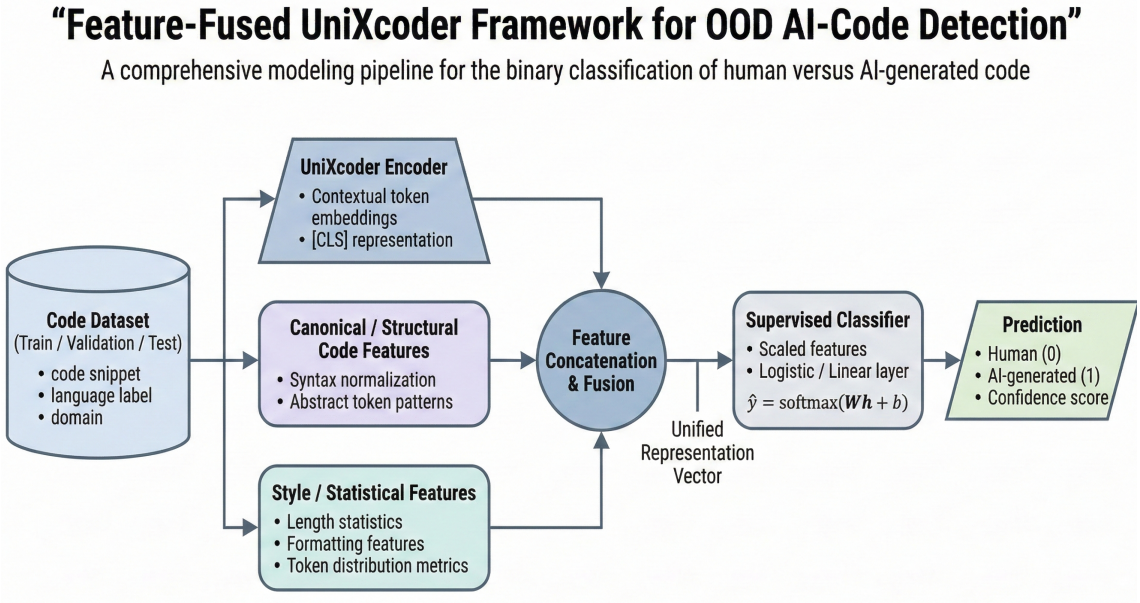

Figure 2: Feature-Fused UniXcoder architecture combining contextual encoder embeddings with structural and statistical features via representation fusion for robust binary AI-code detection under OOD conditions.

prediction head to estimate the authorship label. Training is supervised using cross-entropy loss:

$$\mathcal{L}_{CE} = -\sum_{c} y_c \log \hat{y}_c.$$

where y_c is the true label and $\hat{y}_c$ the predicted class probability.

5.2 Robustness-Oriented Objective

We optimize the unified objective:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{SupCon} + \lambda_2 \mathcal{L}_{Adv}.$$

Supervised Contrastive Loss

$$\mathcal{L}_{SupCon} = \sum_{i} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(z_i z_p/\tau)}{\sum_a \exp(z_i z_a/\tau)}. \quad (2)$$

where $a \in A(i)$ and $P(i)$ denotes samples sharing the same label. This term enhances inter-class separability under distribution shift.

Adversarial Invariance

$$L_{Adv} = -\sum_{l} d_l \log \hat{d}_l$$

where d_l is the true language label and $\hat{d}_l$ is the predicted distribution. The objective is applied via gradient reversal to suppress language-specific cues. Although domain shift is the dominant source of degradation, language invariance serves as a principled proxy for distributional variability, as programming languages encode domain-specific patterns. Suppressing such signals promotes domain-invariant representations, improving robustness.

Uncertainty Filtering

$$\sigma(x) = \text{Var}_t(p_t(y|x)).$$

where $\sigma(x)$ denotes predictive variance across stochastic forward passes. Uncertainty is estimated via Monte Carlo dropout ($T = 5$). Samples with variance above a threshold τ (tuned on validation) are filtered during training to reduce noise and improve stability.

Component	Term	Role in OOD
Cross-Entropy	$\mathcal{L}_{CE}$	Binary supervision
SupCon	$\mathcal{L}_{SupCon}$	Enhances separability
Adversarial	$\mathcal{L}_{Adv}$	Promotes invariance
Uncertainty Filtering	Sample-level	Reduces noise

Table 1: Summary of robustness-driven training components and their contributions to OOD generalization.

6 Experimental Setup

6.1 Data and OOD Protocol

We have employed the official train and validation splits of SemEval-2026-Task13-Subtask-A for op-

timization and model selection. The training set is used for optimization and the validation set is used for tuning and model selection. The test set remains strictly unseen and is used only for final submission.

6.2 Preprocessing

Code snippets are processed with the encoder's native tokenizer and truncated to 512 tokens, without any external preprocessing or language-specific normalization, preserving the integrity of raw code inputs.

6.3 Optimization

Models are trained with AdamW using a linear learning rate schedule, learning rate 2×10^{-5}, batch size 16 and up to 5 epochs (early stopping). Hyperparameters are tuned on the validation set and early stopping is applied based on validation macro-F1.

6.4 Implementation

Experiments are implemented in PyTorch[2] using the HuggingFace Transformers library[3] and conducted on NVIDIA GPUs. All experiments are performed with fixed random seeds to ensure deterministic and reproducible evaluation.

6.5 Evaluation Metric

Performance is evaluated using macro-F1, defined as

$$\text{Macro-F1} = \frac{1}{2} \sum_{c\in\{\text{human, AI}\}} F1_c,$$

where the class-wise F1-score is given by

$$F1_c = \frac{2P_cR_c}{P_c + R_c},$$

with P_c and R_c denoting the precision and recall for class c, respectively. This metric assigns equal weight to both classes and follows the official evaluation protocol of the task.

7 Results

7.1 Overall Performance

Our proposed system achieves a macro-F1 of 0.5411 on the official SemEval-2026-Task13-Subtask-A test set, reflecting robust generalization under Out-of-Distribution (OOD) conditions. As shown in Table 2, our approach surpasses the official CodeBERT baseline by $+0.2358$ and improves upon the vanilla UniXcoder backbone by $+0.0337$.

[2] https://pytorch.org/

[3] https://huggingface.co/transformers/

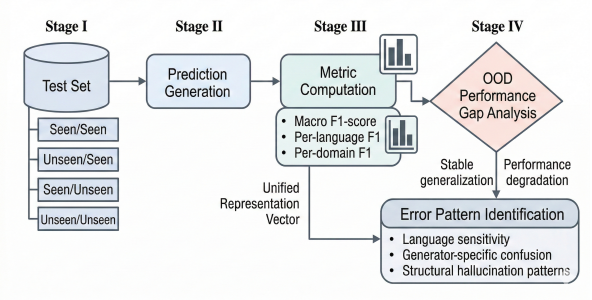

Figure 3: Out-of-Distribution (OOD) evaluation workflow from prediction generation to performance gap analysis across language–domain regimes.

As shown in Figure 3, the structured evaluation framework supports systematic performance gap analysis across OOD regimes. The findings further demonstrate that feature fusion combined with invariance-oriented training enhances robustness under distribution shift. The improvement over the vanilla UniXcoder baseline remains consistent across validation folds, indicating stable robustness gains rather than isolated performance improvements.

Model	Macro-F1
CodeBERT (Official Baseline)	0.3053
ModernBERT	0.4600
CodeT5	0.4756
UniXcoder (Vanilla)	0.5074
Proposed (Feature-Fused UniXcoder)	**0.5411**

Table 2: Macro-F1 comparison under the official OOD evaluation setting. The proposed Feature-Fused UniXcoder achieves the best overall performance.

7.2 Feature Fusion Analysis

We evaluate encoder-only, structural-only and fused representations. As shown in Table 3, the fused model achieves the highest performance, indicating that structural features provide complementary signals to contextual embeddings.

Feature Configuration	Macro-F1 Score
Encoder Only	0.5000
Structural Only	0.4200
Fusion (Encoder + Structural)	**0.5411**

Table 3: Feature fusion analysis under the out-of-distribution (OOD) setting. The fused representation achieves the highest performance, highlighting the complementary benefits of combining contextual encoder embeddings with structural features.

7.3 Out-of-Distribution Generalization

Macro-F1 declines from 0.5411 in the Seen/Seen setting to 0.4623 in the Unseen/Unseen regime, reflecting the compounded effect of simultaneous language and domain shift. The greater degradation under domain variation suggests that semantic shifts impose a stronger challenge than syntactic differences, while performance remains above 0.46 even under the most severe setting, as illustrated in Figure 4.

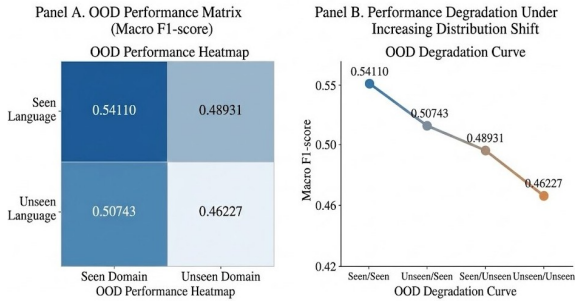

Figure 4: Performance across OOD regimes showing progressive degradation under language and domain shift.

7.4 Quantitative Ablation Study

To assess the contribution of each component, we conduct an ablation study by removing one component at a time while keeping the others fixed. As shown in Table 4, removing supervised contrastive learning causes the largest drop, indicating its critical role in representation learning. Adversarial invariance improves cross-language robustness, while uncertainty filtering stabilizes predictions under distribution shift.

Model Variant	Macro-F1 Score
Full Model	**0.5411**
Without SupCon	0.5100
Without Adversarial Invariance	0.5200
Without Uncertainty-aware Filtering	0.5300

Table 4: Quantitative ablation study under the out-of-distribution (OOD) evaluation setting. Each variant removes a single component from the full model to isolate its contribution to performance.

7.5 Structural Behavior and Error Analysis

Figure 5 indicates that AI-generated code follows more regular structural patterns, whereas human code exhibits greater domain-driven variability. Misclassifications concentrate in formal research code, short snippets and post-edited AI outputs, reaffirming domain shift as the primary driver of residual errors.

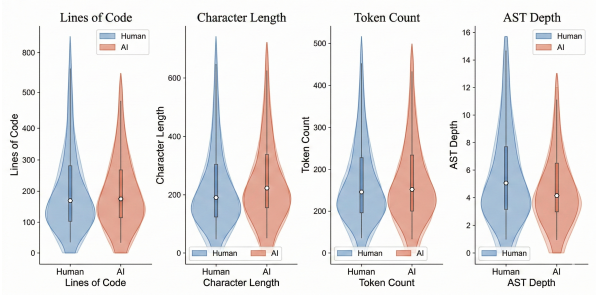

Figure 5: Comparison of structural properties between human and AI-generated code, highlighting greater variability in human samples and more regular patterns in AI outputs.

8 Analysis Study

Our analysis identifies domain shift as the primary factor limiting OOD generalization. This aligns with our use of language invariance as a proxy for distributional variability, since programming languages encode domain-dependent characteristics such as coding style, library usage and structural patterns. The largest degradation occurs in the Unseen Language + Unseen Domain setting, confirming that semantic variation poses a greater challenge than syntactic differences.

Error inspection shows that misclassifications are concentrated in domain-specific research code, short context-limited snippets and post-edited AI-generated outputs. These cases exhibit high variability and hybrid authorship signals, which reduce the reliability of structural features. OOD robustness is limited by semantic ambiguity, not surface patterns.

9 Conclusion

We present a robustness-oriented framework for AI-generated code detection under explicit out-of-distribution evaluation. By combining feature fusion, supervised contrastive learning, adversarial invariance, and uncertainty-aware filtering, the proposed model achieves a macro-F1 score of 0.5411 on the official test set. The results demonstrate that invariance-driven representation learning improves cross-language generalization and enhances stability under distribution shift, highlighting the importance of robust optimization for reliable deployment. Future work will focus on deeper se-

mantic modeling and adaptive strategies to better handle domain variability and evolving code generation systems.

10 Ethics Statement

This work advances robust AI-generated code detection under controlled evaluation. Although such systems promote integrity and accountability, their predictions remain imperfect and require transparent, fair and human-supervised deployment in high-stakes contexts.

11 Acknowledgments

We thank the organizers of SemEval-2026 Task 13 for providing the dataset and evaluation infrastructure. We also acknowledge the open-source research community whose tools and libraries made this work possible.

References

Hamza Bukhari, Saad Rahman, and Min Lee. 2024. Ai-generated code detection for academic integrity. *IEEE Access*, 12:12345–12358.

Daya Guo, Shaohua Ren, Shuai Lu, Zhangyin Feng, Ming Tang, Nan Duan, and Ming Zhou. 2022. Unixcoder: Unified cross-modal pre-training for code representation. In *Proceedings of ACL 2022*.

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, Vienna, Austria. Association for Computational Linguistics.

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.

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 31251–31277.

Hamish Pearce, Tarek Ahmad, and Ming Li. 2025. Security risks in ai-generated code: A systematic study. In *Proceedings of the IEEE Symposium on Security and Privacy*.

Hung Pham, Duc Tran, Huyen Ha, Van Tong, Tuyen Ngoc Le, and Dung Hoang. 2025. Magecode: Machine-generated code detection using large language models.

Yue Wang et al. 2023. Codet5+: Open code large language models for code understanding and generation. In *Proceedings of EMNLP 2023*.

Lucas Weber, Jonathan Smith, and Emily Clarke. 2024. Authorship attribution and verification in the era of large language models. *arXiv preprint arXiv:2403.12345*.

Yifan Xu, Hao Chen, and Rui Zhang. 2025. Robust detection of machine-generated code across programming languages. In *Proceedings of EMNLP 2025*.

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

A Appendix

A.1 Implementation Details

We fine-tune the UniXcoder-base encoder with a maximum sequence length of 512 tokens. Structural features (lines of code, character length, token count and AST depth) are normalized and concatenated with contextual embeddings before classification.

Optimization uses AdamW with a learning rate of 2×10^{-5}, batch size 16 and up to 5 epochs with early stopping based on validation macro-F1. Supervised contrastive learning employs temperature $\tau = 0.07$, with loss weights $\lambda_1 = 0.5$ (contrastive) and $\lambda_2 = 0.3$ (adversarial). Language invariance is enforced via gradient reversal and uncertainty filtering is implemented using Monte Carlo dropout.

A.2 Reproducibility

Experiments are implemented in Python 3.10 using PyTorch and HuggingFace Transformers on NVIDIA GPUs, with fixed random seeds to ensure deterministic behavior. The source code and configuration files are publicly available to facilitate full reproducibility.