NUST CodeIntel at SemEval-2026 Task 13: Cross-Domain Detection of Machine-Generated Code via Stylometric Features and Transformer Models

Azher Ali and Mehwish Fatima
School of Electrical Engineering and Computer Science (SEECS),
National University of Sciences and Technology (NUST), Islamabad, Pakistan
{aali.msds2024seecs, mehwish.fatima}@seecs.edu.pk

Abstract

We present our submission to SemEval-2026 Task 13 on cross-language and cross-domain detection of machine-generated code. We compare TF-IDF-based models with stylometric features against LoRA-tuned transformer encoders. While transformers achieve near-perfect in-distribution performance, they degrade sharply on unseen languages and domains. In contrast, a TF-IDF + Logistic Regression model attains the best test Macro-F1 and shows greater robustness. These results highlight the limitations of neural models under distribution shift and the strength of lexical baselines for cross-domain generalization.

1 Introduction

SemEval-2026 Task 13 (Orel et al., 2026) studies binary detection of Machine-Generated Code (MGC) under cross-language and cross-domain generalization (Orel et al., 2025a). The task trains models on algorithmic code written in Python, C++, and Java, and evaluates their ability to generalize to unseen programming languages and problem domains.

We systematically compare two modeling paradigms: (i) TF-IDF-based classifiers augmented with stylometric features, and (ii) pretrained transformer encoders fine-tuned using Low-Rank Adaptation (LoRA) (Hu et al., 2022). Transformer-based models achieve near-saturated validation performance (Macro-F1 = 99.0), but their performance collapses on the hidden test set (Macro-F1 = 36.0), indicating poor cross-domain generalization. In contrast, TF-IDF-based Logistic Regression achieves the best test performance (Macro-F1 = 43.0) and exhibits significantly lower degradation between validation and test settings.

This discrepancy highlights a fundamental trade-off: pretrained code models tend to exploit domain- and language-specific artifacts that do not transfer beyond the training distribution, whereas lexical and stylometric representations capture more stable, language-agnostic signals. The feature distributions in Figure 1 further support this observation by showing systematic differences in formatting-based stylometric patterns across labels. Additionally, Figure 2 reveals language-dependent variations in identifier casing conventions, suggesting that stylistic cues contribute to cross-domain robustness.

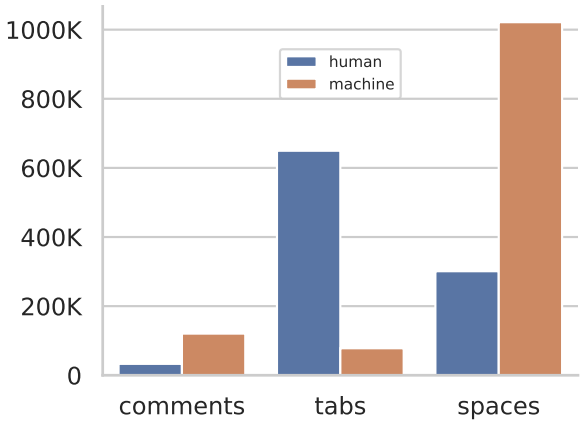


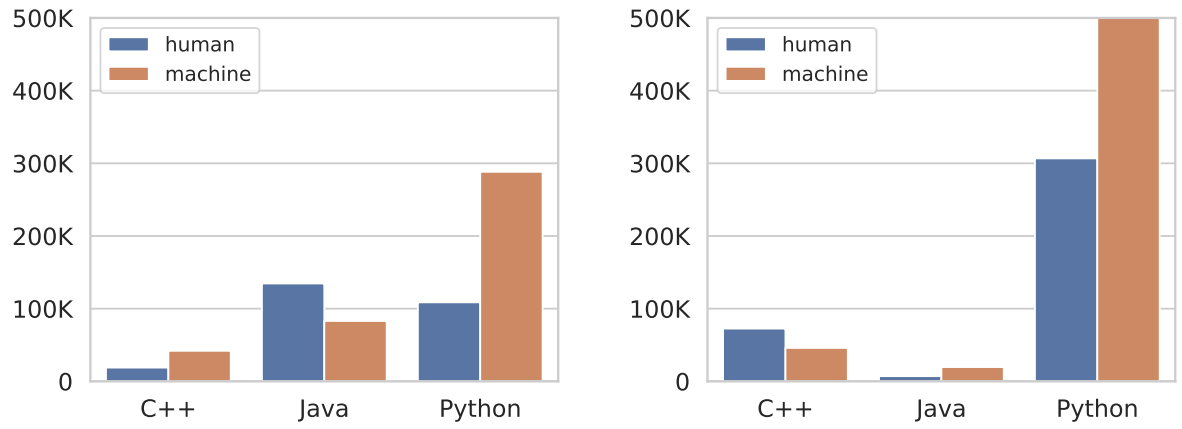
Figure 1: Distribution of formatting-based stylometric features across human-written (HWC) and MGC.

2 Related Work

Early work on source code authorship attribution and classification adopts lexical n-grams, TF-IDF representations, and stylometric features—such as indentation patterns, identifier naming conventions, and Abstract Syntax Tree (AST) statistics—combined with classical machine learning classifiers (Caliskan-Islam et al., 2015; Abuhamad et al., 2018). These approaches effectively capture surface-level regularities but remain highly sensitive to dataset-specific artifacts and superficial cues.

Recent work shifts toward pretrained transformer-based models for code, including CodeBERT, GraphCodeBERT, and UniXcoder, to detect MGC (Feng et al., 2020; Guo et al., 2020, 2022). These models encode richer syntactic and

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 426–433



(a) camelCase usage

(b) snake_case usage

Figure 2: Language-wise distribution of identifier casing conventions, comparing camelCase and snake_case usage across programming languages.

semantic structures through large-scale pretraining on code corpora. However, empirical evidence shows that neural detectors frequently overfit generator- or dataset-specific signatures, leading to substantial performance degradation under domain and language shift (Gehrmann et al., 2019; Mitchell et al., 2023).

To address this limitation, recent hybrid approaches integrate lexical and neural representations to improve robustness and generalization (Marchitan et al., 2024).

In this work, we conduct a controlled comparison between TF-IDF-based stylometric classifiers and LoRA-adapted pretrained code encoders under cross-domain evaluation. We explicitly quantify the validation-to-test performance gap to diagnose generalization behavior under distribution shift and to identify which representation paradigm yields more transferable signals.

3 Task and Dataset

We participate in SemEval-2026 Task 13 (Subtask A) (Orel et al., 2026), which formulates binary detection of MGC. Each instance consists of a standalone code snippet annotated as either human-written code (HWC) or MGC[1,2].

3.1 Data Splits

The dataset (Orel et al., 2025b) contains 600K samples, with 500K allocated for training and 100K for validation. The training set includes 238K HWC and 262K MGC snippets. The organizers provide a held-out test set of 1K instances for final evaluation.

3.2 Distributional Characteristics

Languages and Domains. The training data covers three programming languages (C++, Python, Java) and originates exclusively from the algorithmic domain. The test set introduces controlled distribution shift along two axes. It includes both unseen programming languages (Go, PHP, C#, C, JavaScript) and unseen domains (Research, Production), enabling evaluation of cross-language and cross-domain generalization.

Class Distribution. The dataset remains moderately balanced, with 52% MGC and 48% HWC instances.

3.3 Stylistic and Lexical Patterns

Formatting and Comment Patterns. Our exploratory analysis identifies systematic stylistic differences between classes. HWC exhibits higher tab usage and greater formatting variability, whereas MGC relies predominantly on spaces and shows higher comment density (Figure 1). TThese patterns suggest that generative models follow more standardized formatting conventions with reduced stylistic variance.

Identifier Naming Patterns. We analyze camelCase and snake_case usage across languages (Figure 2). MGC adheres more consistently to canonical naming conventions such as snake_case in Python, camelCase in Java, while HWC demonstrates greater variability and inconsistency.

[1] SemEval-2026-Task13

[2] SemEval-2026-Task13-Data

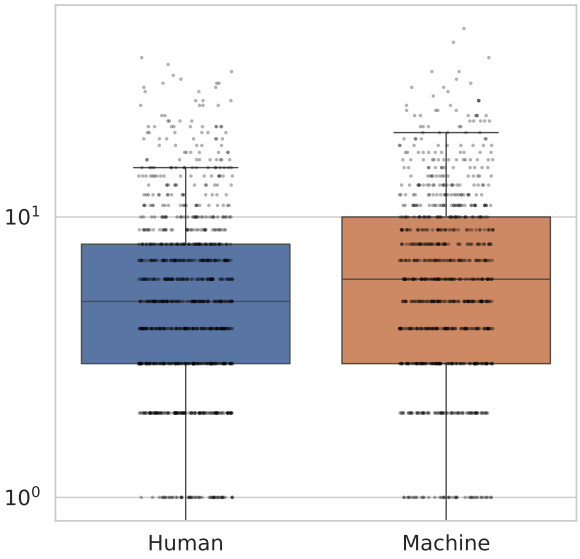


Figure 3: Log-scaled distribution of structural complexity scores for HWC and MGC. MGC exhibits a slightly higher median and variance, although substantial overlap limits discriminative power.

3.4 Structural Complexity

We quantify syntactic complexity using a lightweight structural metric defined as the frequency of control-flow and object-oriented keywords (`for`, `while`, `if`, `elif`, `class`, `def`). To reduce length-induced bias, we exclude the top 10% longest files prior to computing complexity scores. We plot the resulting distribution on a logarithmic scale to account for its right-skewed nature. Figure 3 shows that MGC exhibits a slightly higher median complexity and greater variance, although substantial overlap limits the discriminative power of structural features.

These observations motivate the inclusion of language-agnostic stylometric features—such as indentation ratios, comment density, and identifier casing proportions—to capture transferable structural regularities. However, their limited separability further necessitates complementary lexical and contextual modeling.

4 Our Pipelines

We implement two pipelines: (i) an ML pipeline based on TF-IDF and stylometric features, and (ii) a transformer pipeline using LoRA-adapted pretrained encoders. The first pipeline emphasizes interpretable, surface-level statistics, whereas the second captures deeper syntactic and contextual representations. We release all code and trained models for reproducibility[3].

[3] Our code

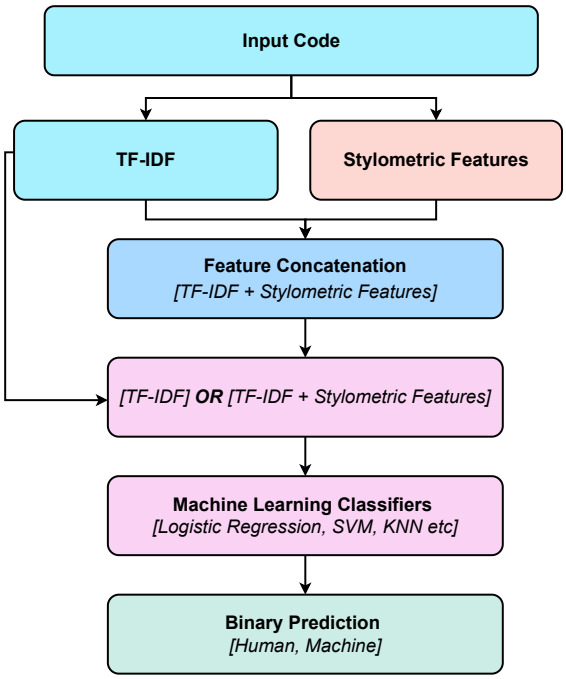


Figure 4: ML pipeline: TF-IDF lexical features, optionally combined with stylometric features, are used to train classifiers for binary code classification.

4.1 ML Pipeline

Figure 4 shows the architecture of our machine learning pipeline.

4.1.1 TF-IDF

We encode each code snippet as a sparse TF-IDF vector derived from word-level tokenization while preserving indentation, comments, and identifiers. We restrict the representation to unigram features with a maximum vocabulary size of 30,000 and apply a minimum document frequency threshold of 0.0015 to filter rare tokens. We perform Unicode normalization to ensure consistency across programming languages. This representation captures lexical regularities associated with MGC, including repetitive patterns, boilerplate constructs, and template-like structures.

4.1.2 Stylometric Features

We design 13 language-agnostic stylometric features that approximate coding style without requiring syntactic parsing. These features include length-based statistics, comment density, indentation ratios, keyword frequencies, and identifier casing proportions (camelCase vs. snake_case). Let $x_{\text{style}} \in \mathbf{R}^{13}$ denote the stylometric feature vector. We concatenate lexical and stylometric representations to form the final input: $\mathbf{X} = [\mathbf{x}_{\text{tfidf}}; \mathbf{x}_{\text{style}}]$. Stylometric features encode structural regularities

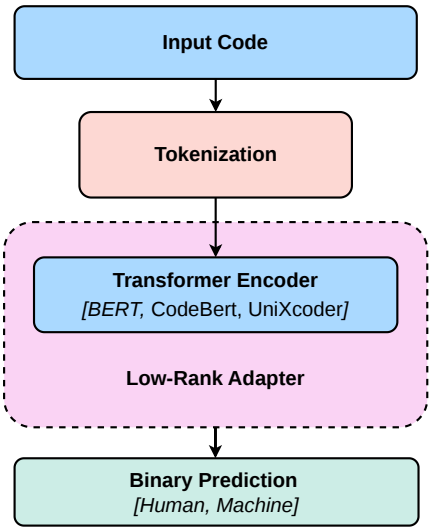


Figure 5: Transformer-based pipeline: input code is tokenized and encoded using models such as BERT, CodeBERT, and UniXcoder, with LoRA-based adaptation for binary classification.

that complement TF-IDF's lexical distributional signals.

4.1.3 Classification Models

We train and evaluate a diverse set of classifiers on this representation, including Logistic Regression, Linear Support Vector Machine, Multinomial Naive Bayes, Decision Tree, Random Forest, Gradient Boosting, XGBoost, LightGBM, and k-Nearest Neighbors.

4.1.4 Hyperparameter Optimization

We optimize hyperparameters using Optuna on the validation set. The best-performing configuration selects Logistic Regression with C=8.85, `class_weight = balanced`, and the `liblinear` solver, combined with a TF-IDF vectorizer using 30,000 maximum features, unigram tokens, and `min_df` = 0.0015. This configuration achieves the most stable precision-recall trade-off under mild class imbalance.

4.2 Transformer-Based Pipeline

Figure 5 illustrates the transformer-based pipeline.

4.2.1 Pretrained Encoders

Pretrained code language models capture contextual, syntactic, and semantic dependencies beyond surface-level statistics. We fine-tune multiple transformer encoders, including BERT, RoBERTa, CodeBERT, and UniXcoder, for binary classification.

4.2.2 Parameter-Efficient Fine-Tuning

We apply Low-Rank Adaptation (LoRA) to reduce the number of trainable parameters while maintaining expressive capacity. We inject low-rank adapters into attention layers and update only these parameters during fine-tuning.

4.2.3 Training Setup

We train models using the HuggingFace Trainer API with FP16 mixed precision and early stopping based on validation performance. The training setup uses 500K instances with validation on 100K samples.

4.2.4 Generalization Behavior

Transformer models achieve near-saturated validation performance (Macro-F1 = 99.0), indicating strong in-distribution fitting. However, performance drops sharply on the held-out cross-language and cross-domain test set (Macro-F1 = 36.0), demonstrating limited robustness under distribution shift. This behavior suggests that pretrained encoders rely on training-specific artifacts rather than learning transferable invariances across programming languages and domains.

5 Experiments

We evaluate three modeling paradigms: (i) TF-IDF lexical representations, (ii) TF-IDF augmented with stylometric features, and (iii) fine-tuned transformer-based encoders. We train classical classifiers on fixed representations and fine-tune transformers for 5 epochs with batch size 16 and learning rate 2×10^{-5}. We evaluate all models using Accuracy and Macro-F1 on the test set.

6 Results

We report Accuracy and Macro-F1 on the test set, with Macro-F1 as the primary metric due to mild class imbalance and cross-domain evaluation.

6.1 ML Pipelines

Table 1 compares classifier performance under different feature representations. Using TF-IDF features alone, Logistic Regression achieves the best performance (Macro-F1 = 43.0), followed by Linear SVM (41.9). Augmenting TF-IDF with stylometric features shifts the performance landscape. KNN achieves the highest Macro-F1 (42.7), while linear models degrade relative to the TF-IDF-only setting. We provide detailed per-class performance

Model	TF-IDF		TF-IDF + Stylometric	
	Acc	Macro-F1	Acc	Macro-F1
Logistic Regression	**50.8**	**43.0**	43.7	39.0
Linear SVM	46.8	41.9	43.2	39.0
KNN	47.6	39.7	**47.7**	**42.7**
Multinomial NB	35.1	38.0	42.0	41.6
Decision Tree	31.4	38.4	32.2	38.0
Random Forest	29.5	38.1	29.4	34.3
XGBoost	28.8	37.8	29.3	38.5
LightGBM	30.1	38.4	30.3	38.7
Gradient Boosting	33.6	35.7	40.2	42.1

Table 1: Test performance of machine learning classifiers using TF-IDF features and TF-IDF augmented with stylometric features.

Model	Validation		Test	
	Acc	F1	Acc	F1
CodeBERT	**99.0**	**99.1**	**32.1**	36.2
BERT-base	98.1	98.1	29.7	**37.6**
UniXcoder	97.1	97.2	27.7	35.9
RoBERTa	98.5	98.5	28.3	35.1

Table 2: Validation and test performance of pretrained encoder models.

Model	Macro-F1
LR (TF-IDF)	**47.3**
CodeBERT	30.1
BERT-base	27.0
UniXcoder	25.1
RoBERTa	28.3

Table 3: Final submission performance (Macro-F1) on the official test set.

and error characteristics of the best-performing model in Appendix B.

6.2 Transformer Pipeline

Table 2 reports validation and test performance of pretrained encoder models. All models achieve near-perfect validation scores ($F1 > 97$), but test performance drops substantially under cross-language and cross-domain evaluation (Macro-F1: 35.1–37.6).

Validation–Test Discrepancy. The validation set follows the training distribution, whereas the test set introduces unseen languages and domains. As a result, validation performance reflects in-distribution generalization, while test performance captures robustness under distribution shift. The substantial performance gap highlights limited cross-domain and cross-lingual generalization.

Table 3 reports the official evaluation results on the hidden test set, which differ from local results due to the unseen test distribution.

7 Analysis

This section compares ML classifiers with fine-tuned transformer encoders, focusing on cross-language and cross-domain robustness under the official SemEval-2026 evaluation protocol.

7.1 Effect of Stylometric Features

TF-IDF representations alone yield strong performance with linear models, indicating high linear separability in lexical space. These features capture token-level frequency patterns and recurring lexical structures that differentiate MGC from HWC.

Augmenting TF-IDF with stylometric features introduces additional structural signals, such as indentation patterns, comment density, and identifier casing. However, this augmentation reduces performance for linear classifiers, suggesting that the combined feature space introduces redundancy or violates linear separability assumptions. In contrast, KNN improves under this setting, indicating that stylometric features contribute useful local structure that instance-based methods can exploit.

Tree-based models consistently underperform across both settings. This behavior likely stems from the sparsity and high dimensionality of lexical representations, which limit the effectiveness of hierarchical partitioning and lead to unstable decision boundaries.

7.2 Robustness Under Distribution Shift

Transformer models exhibit near-perfect in-distribution performance but degrade sharply on the test set, highlighting their sensitivity to cross-language and cross-domain shifts. This gap suggests that pretrained encoders learn representations that are closely tied to the training distribution.

In contrast, TF-IDF-based models show smaller degradation, indicating greater robustness to distributional variation. Unlike transformers, which rely on contextual embeddings, TF-IDF captures surface-level lexical statistics that remain relatively stable across programming languages and coding styles.

This contrast reveals a fundamental trade-off: transformer models achieve strong representation learning under homogeneous conditions, while simpler lexical models generalize better when the test distribution diverges from the training data.

7.3 Feature-Level Insights

To better understand the robustness of TF-IDF models, we analyze the highest-weighted features learned by Logistic Regression. We identify three dominant feature groups associated with MGC. First, the model assigns high weights to dataset and generation artifacts, including explicit language identifiers ("python", "cpp", "java") and special tokens such as "endoftext" (Table 4). These features suggest partial reliance on signals introduced during data generation or preprocessing.

#	MGC	LR-weight	HWC	LR-weight
1	python	82.0	class	-22.4
2	endoftext	52.5	provide	-19.8
3	cpp	37.1	stdc	-18.9
4	explanation	30.8	iowrapper	-15.2
5	python3	25.3	parsedouble	-11.5

Table 4: Top 5 most discriminative TF-IDF features for machine-generated (positive) vs. human-written (negative) code.

Second, the model emphasizes instructional and explanatory natural language ("example", "explanation", "approach"), indicating that MGC frequently contains tutorial-style descriptions or meta-commentary.

Third, canonical boilerplate constructs ("main", "init", "namespace") appear prominently, reflecting standardized coding patterns produced by generative systems.

In contrast, HWC exhibits more irregular and domain-specific lexical patterns, including low-level system calls ("fstat", "st-size") and input/output handling constructs ("parsedouble", "inputstream"). These features reflect greater variability and less standardized structure. We provide top-30 most discriminative TF-IDF features in Appendix A in Table 5 .

Overall, TF-IDF captures stable distributional signals spanning lexical usage, natural language content, and generation artifacts. These signals persist across domains and programming languages, which explains the improved robustness of TF-IDF-based models under distribution shift compared to transformer-based representations. .

8 Conclusions

We present a comparative study for SemEval-2026 Task 13 on cross-language and cross-domain detection of MGC. Transformer models achieve near-perfect validation performance but degrade sharply under distribution shift, while a TF-IDF + Logistic Regression baseline remains more robust and attains the highest test Macro-F1. This contrast indicates that pretrained code models capture distribution-specific patterns, whereas lexical statistics provide more stable cross-domain signals. Overall, the results expose a robustness gap in deep models and motivate future work on domain-invariant representations, including structural normalization and AST-level abstraction.

Limitations

Both pipelines exhibit clear limitations under distribution shift. Transformer models achieve strong in-distribution performance but degrade sharply on unseen languages and domains, indicating reliance on language- and domain-specific artifacts. The TF-IDF pipeline remains comparatively more robust but achieves only modest performance due to limited semantic capacity. Additionally, the ML pipeline operates on snippet-level lexical and stylometric features without leveraging structural program representations or repository-level context. The transformer pipeline relies solely on fine-tuning over algorithmic data and does not incorporate domain adaptation or invariance-driven objectives, which further constrains cross-language generalization.

Acknowledgments

We thank the organizers of SemEval-2026 Task 13 for designing a challenging shared task on machine-generated code detection and for providing a large-scale, multilingual dataset that enables rigorous cross-language and cross-domain evaluation. We also thank the reviewers for their constructive feedback, which helps improve the clarity and quality of this work.

References

Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang. 2018. Large-scale and language-oblivious code authorship identification. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 101–114, New York, NY, USA. Association for Computing Machinery.

Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 255–270, Washington, D.C. USENIX Association.

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, Online. 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.

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, Dublin, Ireland. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, and 1 others. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Teodor-George Marchitan, Claudiu Creanga, and Liviu P Dinu. 2024. Transformer and hybrid deep learning based models for machine-generated text detection. *arXiv preprint arXiv:2405.17964*.

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.

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, and Yuxia Wang. 2025b. Semeval-2026-task13-subtask-a. `https://kaggle.com/competitions/sem-eval-2026-task-13-subtask-a`. Kaggle.

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)*, San Diego, USA. Association for Computational Linguistics.

Appendix

A Top Weighted TF-IDF Features by Logistic Regression

The TF-IDF features listed correspond to the highest positive and negative coefficients learned by the logistic regression model trained on TF-IDF representations. Positive values indicate features that contribute toward predicting machine-generated code, while negative values indicate features associated with human-written code. These coefficients highlight the most influential lexical patterns used by the classifier to distinguish between the two classes.

	MGC	LR-weight	HWC	LR-weight
1	python	82.0	class	-22.4
2	endoftext	52.5	provide	-19.8
3	cpp	37.1	stdc	-18.9
4	explanation	30.8	iowrapper	-15.2
5	python3	25.3	parsedouble	-11.5
6	__name__	24.5	called	-11.2
7	namespace	21.4	lang	-10.3
8	example	21.0	him	-9.7
9	here	20.4	nextdouble	-9.2
10	anything	19.7	signed	-9.2
11	usage	19.1	override	-9.1
12	the	17.5	online_judge	-9.1
13	__init__	17.0	util	-8.4
14	with	16.9	become	-7.9
15	__main__	15.3	ob	-7.3
16	only	15.2	throw	-7.2
17	comments	15.0	fstat	-7.0
18	standard	14.4	writable	-6.5
19	stdio	14.0	hasmoreelements	-6.4
20	java	13.5	st_size	-6.4
21	leetcode	13.5	__starting_point	-6.0
22	raw_input	13.2	such	-6.0
23	based	12.8	typename	-5.9
24	please	12.5	except	-5.9
25	examples	12.3	running	-5.9
26	code	12.2	input	-5.8
27	xrange	12.0	iostream	-5.6
28	without	11.4	online	-5.2
29	constraints	11.3	inputstream	-5.0
30	approach	11.2	hpp	-4.8

Table 5: Top 30 most discriminative TF-IDF features for machine-generated (positive) vs. human-written (negative) code.

B Detailed Per-Class Evaluation of the Final System

The Logistic Regression model exhibits asymmetric behavior, achieving high precision for HWC but low recall, while showing the opposite trend for machine. The overall performance (Macro-F1 = 50.8) remains modest, indicating that purely lexical TF-IDF features struggle to robustly discriminate between the two classes.

Metric	HWC	MGC	Macro Avg
Precision	**89.7**	29.0	59.3
Recall	41.4	**83.4**	**62.4**
F1-score	**56.7**	43.1	49.8
Support	777	223	-
Accuracy		**50.8**	

Table 6: Detailed Classification Results of the Best-Performing Logistic Regression Model on the test set.