contestant001 at SemEval-2026 Task 13: Stylometric and TF-IDF-Based Detection of Machine-Generated Code

Bora Özaylar **Banafsheh Gholinejad**
University of Tübingen, Germany
{bora.ozaylar, banafsheh.gholinejad}@student.uni-tuebingen.de

Abstract

Reliable detection of machine-generated code has become increasingly important for academic integrity and software quality as code generation is largely being undertaken by large language models. This paper presents our approach to SemEval-2026 Task 13, Subtask A: detecting machine-generated code in a binary classification setting, where we propose an ensemble approach combining TF-IDF lexical representations with 23 hand-crafted stylometric features for binary classification of AI-generated code. Our system aims to address the challenge of cross-language generalization by extracting language-agnostic patterns and combining them with TF-IDF. While we observed that transformer-based models (CodeBERT, UniXcoder) noticeably underperformed under distribution shift, our analysis revealed that AI-generated code exhibits distinct stylometric patterns and our TF-IDF ensemble achieved 0.5175 Macro F1 on the task submission.

1 Introduction

Large language models have significantly advanced automatic code generation across multiple programming languages and application domains. Modern generative systems can produce correct and coherent code that closely resembles human-written solutions. As such systems become widely used in education and software development, reliably distinguishing machine-generated code from human-written code has become an important research challenge.

SemEval-2026 Task 13 (Orel et al., 2026a) addresses this issue by evaluating machine-generated code detection under controlled distribution shifts. In Subtask A, models should be trained on algorithmic solutions in C++, Python, and Java, and evaluated on unseen programming languages and domains such as research and production code. In addition, the class distribution differs between training and evaluation sets, further increasing the difficulty of robust generalization.

In this work, we adopt a data-driven and interpretable approach. We first conduct an analysis of stylistic properties to identify systematic differences between human-written and machine-generated code. Our analysis reveals consistent patterns in verbosity, documentation density, whitespace usage, and naming behavior. Motivated by these observations, we design a lightweight detection system that combines language-agnostic stylometric features with TF-IDF lexical representations and linear classification. By emphasizing interpretable and computationally efficient signals, our approach aims to improve robustness under cross-language and cross-domain shifts.

2 Related Work

Previous work in this domain highlights the difficulty of robust detection under cross-language and cross-domain settings. Droid (Orel et al., 2025b) and CoDet-M4 (Orel et al., 2025a) show that many detection systems degrade substantially when evaluated on unseen programming languages, generator families, or application scenarios, establishing cross-language generalization as a central open challenge. AICD Bench (Orel et al., 2026b) further formalizes this by providing a benchmark explicitly designed to stress-test detectors across diverse languages, generators, and domains.

On the model side, CodeBERT (Feng et al., 2020) introduced bimodal pre-training over code and natural language, while UniXcoder (Guo et al., 2022) extended this with unified cross-modal pre-training for code representation. Such models that rely on semantic understanding are susceptible to distribution shift, despite strong in-distribution performance as their tokenizers normalize whitespace and formatting, with the surface-level signals be-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3124–3129

ing most discriminative for authorship detection. These findings motivate our shift toward explicit stylometric feature engineering as a more robust alternative.

3 Data Analysis

3.1 Dataset and Distribution

Split	Size	Labels	Purpose
Training	500K	✓	Model training
Validation	100K	✓	Development
Practice test	1K	✓	Local evaluation
Competition test	500K		Final ranking

Table 1: Dataset splits. All results in this paper are reported on the practice test set.

The training set contains 500K samples (238K human, 262K machine) from LeetCode-style algorithmic problems (C++, Python, Java). Test data (500k samples) extends to unseen languages (Go, PHP, C#, JavaScript, C) in different domains such as research and production to increase calibration and generalization difficulty.

3.2 Stylometric signals

Figure 2 shows selected stylometric feature distributions on the main test set. Machine-generated code consistently exhibits higher documentation density (`comment_ratio`), larger empty-line ratios, and greater verbosity (log character length); other proxies (complexity, raw line count) are less discriminative.

3.3 Feature bias and relations

Figure 2 summarizes per-feature bias (Machine − Human) and a compact profile: machine outputs are associated with more comments/docstrings, higher space usage, and slightly longer identifiers, while human samples show higher tab usage and marginally greater compression variability. The correlation heatmaps in Figure 4 reveal stronger, more coherent dependencies among length/structure features for machine-generated code compared to human code, supporting the view that generated code follows more regular templates.

3.4 Implication

These language-agnostic stylistic patterns, especially differences in documentation, blank-line usage, and verbosity, remain consistent across unseen languages and domains. This motivates our lightweight and interpretable feature-based detection approach.

4 Methodology

Our proposed system adopts a dual-path ensemble architecture that combines complementary signal types for robust detection under distribution shift. Our design is motivated by our observations of lexical patterns capturing language-specific surface regularities, and stylometric features providing language-agnostic authorship signals that transfer across programming languages.

Our complete pipeline diagram (Figure 5) illustrates the two parallel paths process the sample independently from a given input code snippet, where Path 1 applies TF-IDF vectorization to extract lexical features, which are fed into a logistic regression classifier to produce a probability score p_{tfidf}. Path 2 computes hand-crafted stylometric features we extract, which are then fed into a random forest classifier to produce the probability score p_{feat}. Thereby we aim to ensure with this approach that the system leverages both lexical and stylometric information, with TF-IDF capturing possible local language-specific word patterns while hand-crafted features measuring formatting and documentation behaviors that we observed to remain consistent across languages. The ensemble weights were determined from validation performance on given task validation set, where we observed stylometric features alone (0.51 Macro F1) outperformed TF-IDF alone (0.44 Macro F1). The elevated threshold (0.7 vs. default 0.5) compensates for the large class distribution shift between training (48% human) and test (78% human) in the dataset provided by the task.

Lexical Representation with TF-IDF (Path 1) In parallel, we construct lexical representations using TF-IDF vectorization where experiment two configurations: TF-IDF: word-level 1–5 grams with 100K features, incorporating language tags (e.g., [LANG=Python]) and language-based sample weighting, trained with Logistic Regression. This representation captures local lexical regularities while remaining computationally efficient.

Stylometric Feature Extraction (Path 2) Motivated by the structural differences identified in Section 2, we extract 23 hand-crafted stylometric features capturing formatting behavior, documentation density, naming patterns, and structural regu-

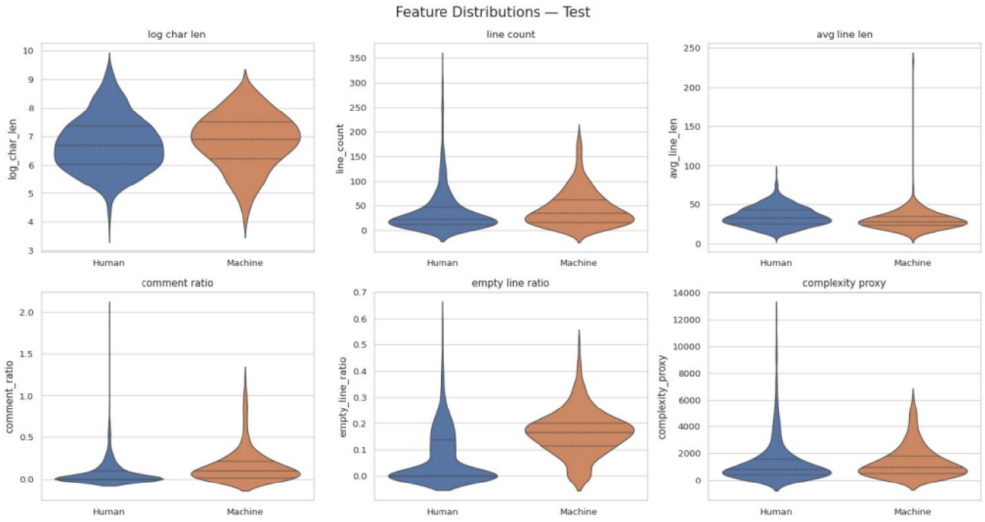

Figure 1: Selected stylometric feature distributions on the evaluation set (unseen languages/domains).

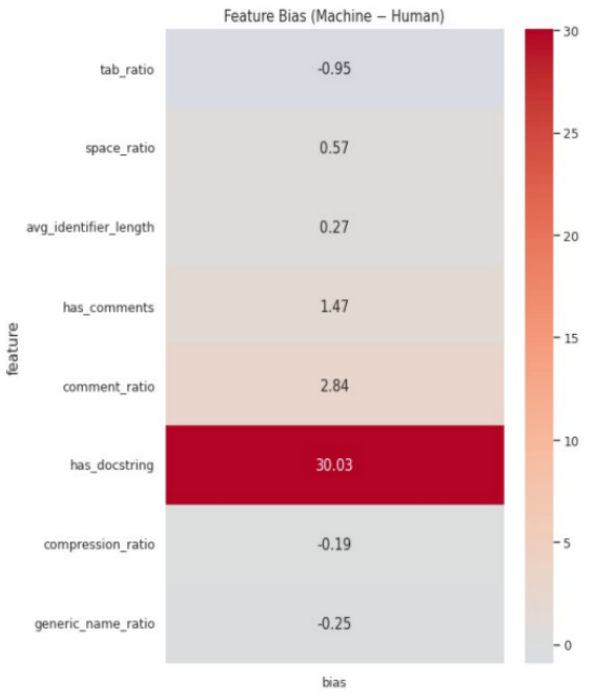

Figure 2: Feature bias (Machine − Human).

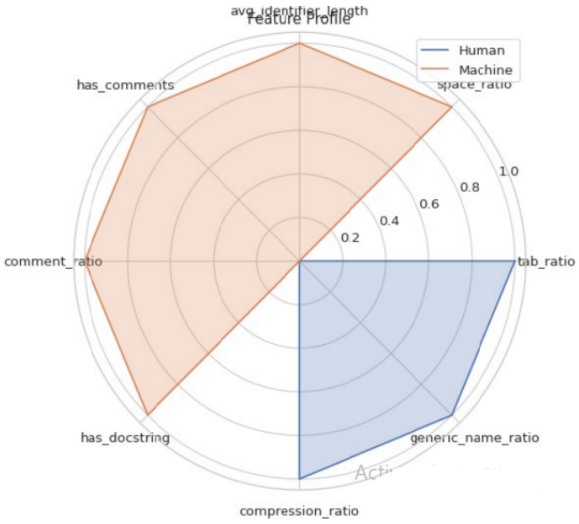

Figure 3: Stylometric profile for selected features.

larities.

Whitespace and formatting Tab ratio and space ratio.

Documentation features Comment ratio, presence of comments, and presence of docstrings.

Naming characteristics Average identifier length and generic naming ratio.

Verbosity and layout Log-scaled character length, line count, average line length, and empty-line ratio.

Regularity proxies Compression ratio and other lightweight structural indicators.

We computed all features directly from raw source code without parsing or semantic analysis, to ensure language-agnostic applicability across unseen programming languages.

Ensemble Strategy The final system employs a weighted ensemble combining predictions from the TF-IDF model and the stylometric Random Forest classifier where p_{tfidf} and p_{feat} denote the predicted probabilities from the lexical and stylometric models, respectively. The ensemble score is computed as:

$$p = 0.4 \cdot p_{\text{tfidf}} + 0.6 \cdot p_{\text{feat}}$$

A classification threshold of 0.7 is applied to

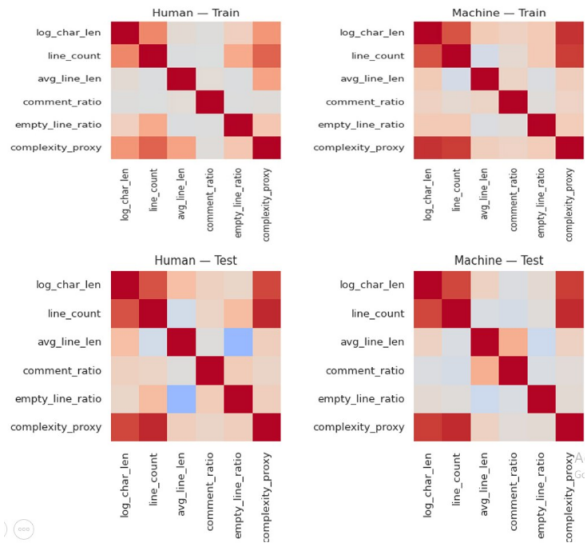

Figure 4: Feature relation heatmaps (train/test × human/machine) demonstrating stronger feature dependencies in machine-generated samples.

obtain the final prediction to enables the system to leverage both lexical cues and structural stylometric signals under distribution shift.

5 Experimental Setup

We use the provided data splits without modification: 500K samples for training, 100K samples for validation (development), and 1K samples for test evaluation. No additional external data sources were used beyond the official training set.

5.1 Implementation Details

TF-IDF Models. We use `scikit-learn` (version 1.3.0) for TF-IDF vectorization and Logistic Regression. For TF-IDF (improved), we set word-level n-gram range to (1,5), `max_features=100000`, `min_df=2`, and `sublinear_tf=True`. Language tags are prepended to code as `[LANG={language}]`. Sample weights are computed as inverse language frequency to balance the three training languages. Logistic Regression uses `C=1`, `solver='saga'`, `max_iter=1000`, and `class_weight='balanced'`.

Stylometric Features. We extract 23 features using custom Python scripts operating on raw source code strings. Features include: `tab_ratio`, `space_ratio`, `comment_ratio`, `has_comments`, `has_docstring`, `avg_identifier_length`, `generic_name_ratio`, `compression_ratio`, `char_length`, `line_count`, `avg_line_length`, and others capturing whitespace, documentation, naming, and structural patterns. Random Forest uses 100 trees with `class_weight='balanced'` and default `scikit-learn` hyperparameters.

Approach	Val F1	Test F1
CodeBERT	0.99	0.24
UniXcoder	0.98	0.27
TF-IDF	0.78	0.44
Features (Random Forest)	–	0.51
Ensemble (ours)	–	**0.53**

Table 2: Performance comparison on validation and test sets. Transformers exhibit severe overfitting. Our ensemble achieves the best test performance.

Ensemble. The ensemble weights (0.4 for TF-IDF, 0.6 for features) and classification threshold (0.7) were selected based on validation set performance through manual grid search over weight combinations $\{0.3, 0.4, 0.5, 0.6, 0.7\}$ and thresholds $\{0.5, 0.6, 0.7, 0.8\}$.

Transformer Baselines. CodeBERT and UniXcoder were fine-tuned using the Hugging Face `transformers` library (version 4.57.1). We trained for 3 epochs with batch size 16, learning rate 2×10^{-5}, maximum sequence length 512 tokens, and used the AdamW optimizer. Training was conducted on Google Colab with T4 GPU and 51GB RAM.

Computational Resources. All experiments were conducted on Google Colab. TF-IDF vectorization and training required approximately 15-20 minutes. Feature extraction on 500K samples took 20-30 minutes. Ensemble training completed within 30 minutes total. Transformer fine-tuning required 2-3 hours per model.

6 Results

6.1 Main Results

Table 2 summarizes the performance of all approaches on the test set. Our ensemble system achieves 0.53 macro F1, substantially outperforming transformer baselines (0.24-0.27) and single-model approaches.

Transformer-based models (CodeBERT, UniXcoder) achieve near-perfect validation F1 (0.99, 0.98) but poor test F1 (0.24, 0.27), suggesting noticeable overfitting to the training distribution. This failure is attributed to distribution shift (class imbalance and unseen languages) and the models learning semantic patterns rather than stylometric signals.

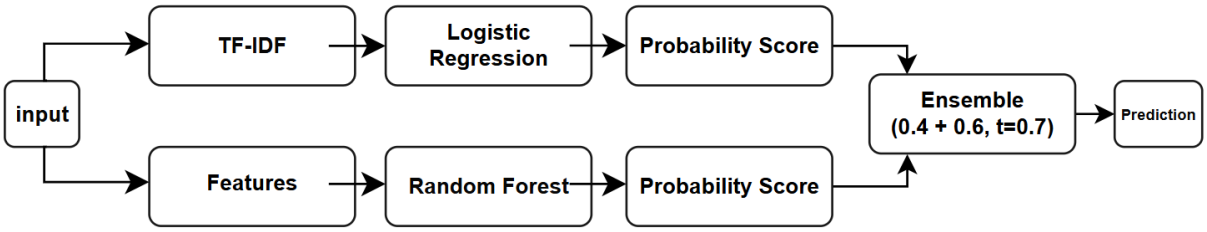

Figure 5: Ensemble model pipelinne using double-path with TDF-ID and sylometric features

Configuration	Test F1
TF-IDF only	0.44
Features only	0.51
Ensemble (equal weights)	0.48
Ensemble (0.4/0.6, thresh=0.5)	0.44
Ensemble (0.4/0.6, thresh=0.7)	**0.51**

Table 3: Ablation study showing the contribution of ensemble components and threshold tuning.

Language	Samples	Seen?	Test F1
Python	450	✓	0.49
Java	80	✓	0.25
C++	104	✓	0.41
Go	120		0.21
PHP	85		0.08
C#	68		0.27
JavaScript	58		0.26
C	35		0.29
Overall	1000	–	**0.53**

Table 4: Per-language performance breakdown. Unseen languages generally perform worse, with PHP being particularly challenging.

Among lexical approaches, TF-IDF (improved) with language tags and sample weighting achieves 0.44 F1, outperforming the basic configuration (0.33 F1) by 11 percentage points. The hand-crafted feature model alone achieves 0.51 F1, demonstrating that explicit stylometric signals are more robust than lexical patterns under distribution shift.

Our ensemble combines both signals, achieving 0.53 F1. This represents a 2-point improvement over features alone and a 9-point improvement over TF-IDF alone, confirming that the two representations provide complementary information.

6.2 Ablation Study

Table 3 shows the impact of key design choices.

Ensemble weights. Equal weighting (0.5/0.5) achieves 0.48 F1, while our optimized weights (0.4/0.6) reach 0.53 F1. The higher weight on features reflects their superior standalone performance.

Threshold tuning. Using the default threshold of 0.5 yields 0.44 F1, while our tuned threshold of 0.7 achieves 0.53 F1. This 9-point improvement demonstrates the importance of adapting to class distribution shift (78% human in test vs. 48% in training).

6.3 Cross-Language Analysis

Table 4 breaks down performance by programming language.

Performance varies substantially across languages. Among seen languages, Python achieves the highest F1 (0.49), likely due to its prevalence in the training set. Among unseen languages, performance is generally lower, with PHP (0.08) being particularly challenging. This suggests that while stylometric features provide some cross-language transfer, language-specific patterns remain important. Our initial experiments with transformer-based approaches (CodeBERT (Feng et al., 2020), UniXcoder(Guo et al., 2022)) observed severe overfitting where the model achieved near-perfect validation scores (0.99 Macro F1) yet poor test performance (0.24-0.27 Macro F1), which motivated our shift toward lightweight, more interpretable features that explicitly target stylometric differences rather than relying on semantic code understanding since transformers might have learned semantics of code rather than synactic elements.

7 Discussion

The poor test performance of transformer models (CodeBERT, UniXcoder) despite near-perfect validation performance suggests fundamental challenges in applying semantic understanding models to stylometric tasks.

CodeBERT is pre-trained for masked language modeling and code-text matching—tasks requiring semantic understanding of code functionality.

In contrast, AI code detection requires sensitivity to surface-level stylistic patterns (whitespace, formatting, documentation) that transformers are explicitly trained to ignore through tokenization and normalization. Transformers learned to exploit spurious correlations in the training data (e.g., Python-specific patterns) rather than generalizable authorship signals. When confronted with unseen languages and domains, these learned shortcuts failed completely. The severe class imbalance (78% human test vs. 48% training) further degraded decision boundaries. tokenizers normalize whitespace and strip formatting—precisely the features our analysis identified as most discriminative (tab vs. space usage, blank lines). This preprocessing removes the strongest authorship signals before the model sees the input.

These findings suggest that explicit feature engineering targeting stylometric properties may be more effective than end-to-end learning for authorship detection tasks.

8 Conclusion

In this paper, we presented our system for SemEval-2026 Task 13 Subtask A, to address binary classification of machine-generated code under cross-language and cross-domain distribution shift. Our analysis revealed consistent stylometric differences between human-written and machine-generated code, particularly in documentation density, whitespace usage, and verbosity, which remain stable across unseen programming languages and domains.

Our proposed system combining TF-IDF lexical representations with 23 hand-crafted stylometric features achieved 0.53 macro F1 on the test set, outperforming transformer-based baselines (CodeBERT, UniXcoder) in our experiments despite that near-perfect validation performance collapsed to 0.24–0.27 F1 under distribution shift. This confirms that explicit stylometric feature engineering is more robust than end-to-end semantic learning for authorship detection tasks.

For future work, we plan to explore AST-based structural features that remain language-agnostic while capturing deeper syntactic patterns beyond surface formatting, in addition to investigating domain adaptation techniques for transformer models such as calibration and adversarial training which could address brittleness under class imbalance and unseen languages. Extending the feature set with token-level entropy or perplexity signals from smaller language models is another promising direction.

References

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. *Preprint*, arXiv:2002.08155.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *Preprint*, arXiv:2203.03850.

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.

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

Daniil Orel and 1 others. 2026b. AICD bench: A challenging benchmark for AI-generated code detection. In *Proceedings of the 20th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2026)*.