Stylometry at SemEval-2026 Task 13: Clustered Stylometric Modeling for Machine-Generated Code Detection

Sruthi Santhanam, Parthib Sarkar and **Yashvardhan Sharma**
Department of Computer Science and Information Systems
BITS Pilani
{f20220837,f20220111,yash}@pilani.bits-pilani.ac.in

Abstract

Machine-generated code detection is examined under out-of-distribution conditions where robust generalization is required. A hybrid feature representation is used in which code snippets are encoded through character-level TF–IDF patterns together with explicit structural indicators capturing properties such as verbosity and formatting behavior. Variability across generators is handled through clustering-based expert specialization, and predictions are produced using an ensemble of logistic regression and Naïve Bayes models with calibrated thresholds. Experimental results show that the proposed approach achieves a macro F1-score of **0.6125** on the official test set, performing competitively despite relying on simple linear classifiers. The findings suggest that persistent structural patterns in code provide reliable cross-domain signals for identifying machine-generated programs.

1 Introduction

The prevalent adoption of large language models reshapes software development by enabling automated code generation at scale. Systems capable of producing syntactically correct and functionally plausible programs are now widely accessible, and generated code is increasingly integrated into real-world workflows. While this shift accelerates development, it also introduces challenges related to authorship attribution, academic integrity, and software provenance. The ability to distinguish human-written code from machine-generated code is therefore established as an important problem at the intersection of natural language processing and programming language analysis.

Detecting machine-generated content remains difficult because modern generators are trained to imitate human style. Many detectors rely on distributional artifacts tied to training data with specific patterns, and performance degrades under distribution shift. This limitation is especially pronounced in code, where stylistic conventions vary across languages and domains. Robust detection must therefore capture structural regularities that persist across contexts rather than relying on superficial lexical cues.

SemEval-2026 Task 13 Subtask A evaluates this challenge under controlled out-of-distribution conditions. Training data are restricted to limited languages and domains, while evaluation introduces unseen settings, emphasizing stylistic generalization. In this work, a framework is proposed in which hybrid lexical and structural features are combined with clustering-based expert specialization. Competitive performance is achieved without reliance on large neural models, and the analysis demonstrates that structural stylometry provides strong cross-domain signals for machine-generated code detection.The code for our system is available at: https://github.com/parthib-s/semeval-2026-task13-stylometry.

2 Background

Machine-generated content detection has become an important research problem with the rapid adoption of large language models (Radford et al., 2019). Early approaches focus on identifying statistical irregularities in generated text using entropy-based signals, token distributions, and surface-level writing style features (Gehrmann et al., 2019; Jawahar et al., 2020). However, many detectors overfit to training distributions and degrade under domain shift, and even zero-shot methods show limited robustness (Mitchell et al., 2023). These limitations motivate approaches that capture generation-specific structure rather than dataset artifacts.

Compared to natural language, code presents additional challenges. Although programming languages impose syntactic constraints, substantial variation arises from formatting, commenting behavior, and structural organization. Recent benchmarks demonstrate that machine-generated code

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 1319–1325

exhibits consistent verbosity patterns and template-like structures that differ from human-written programs (Orel et al., 2026a). Cross-language evaluations further highlight the need for language-agnostic detection strategies. Pretrained code models such as CodeBERT and CodeT5 have shown strong performance on code understanding tasks (Feng et al., 2020; Wang et al., 2021), but often rely on semantic representations that may not generalize well under distribution shift.

Stylometric analysis provides a natural foundation for this setting (Stamatatos, 2009). Character-level patterns and punctuation statistics have been successfully applied in authorship attribution and programmer identification (Caliskan-Islam et al., 2015; Koppel et al., 2009), as they capture structural regularities largely independent of semantics.

To model heterogeneity in writing style across generators, mixture-of-experts frameworks offer a suitable approach. By partitioning the input space into coherent regions and training specialized classifiers, expert models can better capture diverse stylistic regimes (Jacobs et al., 1991). The proposed system follows this intuition by combining stylometric features with clustering-based expert specialization for cross-domain machine-generated code detection.

3 Task, Dataset, Exploratory Data Analysis

3.1 Task

SemEval-2026 Task 13 Subtask A (Orel et al., 2026b) formulates machine-generated code detection as a binary classification problem under out-of-distribution conditions. Models are trained on a limited set of programming languages and a single algorithmic domain, while evaluation introduces unseen languages and domains. The task therefore emphasizes stylistic generalization rather than memorization of language-specific patterns. Performance is measured using macro F1-score.

3.2 Dataset Description

The combined training and validation sets are approximately class-balanced. Python constitutes roughly 91% of the data, with C++ and Java forming the remainder, and language proportions are similar across classes. Machine-generated samples originate from 34 different generators, resulting in substantial stylistic heterogeneity within the AI class. This diversity motivates specialized modeling rather than a single global classifier.

3.3 Exploratory Data Analysis

Exploratory analysis reveals consistent structural differences between human-written and machine-generated code.

AI-generated snippets are substantially longer. Human-written code averages approximately 600 characters (median 319), whereas AI-generated snippets average 1052 characters (median 727). Line counts show a similar trend (33 vs. 40 lines on average), reflecting increased verbosity in generated outputs.

Comment usage exhibits strong separation: the mean comment ratio is approximately 0.03 for human-written code and 0.105 for AI-generated code, indicating significantly heavier documentation in generated snippets.

Punctuation density is higher in human-written code (0.217 vs. 0.157), suggesting more compact syntax, while machine-generated snippets display greater lexical diversity (49 vs. 39 unique characters on average).

Overall, AI-generated code tends to be longer, more heavily commented, and more diverse, whereas human-written code is more compact and punctuation-dense. These structural differences are largely language-agnostic and motivate the hybrid stylometric representation used in our system.This separation is further illustrated in Figure 1, where the joint distribution of length and comment features reveals distinct regions for the two classes.

4 Methodology

The system follows a staged pipeline: (1) hybrid stylometric feature extraction, (2) clustering-based specialization, (3) ensemble classification, and (4) threshold calibration. The design prioritizes structural robustness and computational simplicity.

4.1 Feature Representation

Each snippet is encoded using a hybrid representation combining character-level TF-IDF features (Ramos, 2003) and explicit structural descriptors.

Character n-grams (3-5) capture fine-grained stylistic signals such as indentation, punctuation density, symbol frequency, and formatting patterns. Because they operate below the token level, character features remain largely language-agnostic and model stylistic compression versus verbosity.

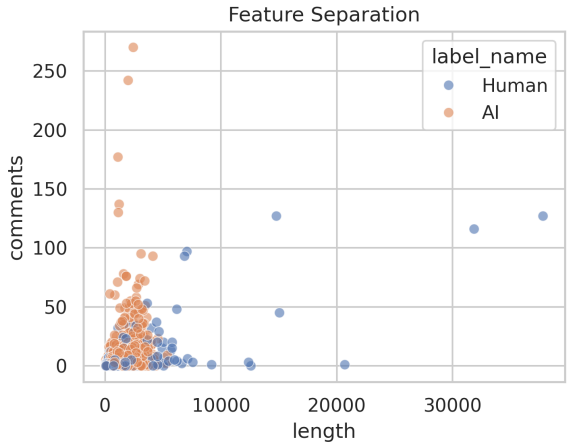

Figure 1: Two-dimensional feature space defined by code length and comment count. The plot shows a clear separation between human-written and machine-generated samples, indicating that combining structural features provides strong discriminative power.

Structural features include total length, line count, comment ratio, digit ratio, punctuation ratio, and lexical diversity (unique character count). These descriptors encode macro-level differences identified during exploratory analysis, particularly verbosity and documentation behavior.

The final feature vector is formed by concatenating TF-IDF representations with standardized structural features.

4.2 Clustering-Based Specialization

To address stylistic heterogeneity introduced by multiple generators and programming contexts, training data are partitioned using K-means clustering (Lloyd, 1982) in TF-IDF space. Each cluster represents a localized stylistic regime.

Cluster-specific classifiers are trained instead of a single global model. During inference, a snippet is assigned to its nearest centroid and evaluated by the corresponding expert. This lightweight mixture-of-experts strategy enables localized decision boundaries without increasing model complexity.

Mixture-of-Experts Interpretation. The proposed framework can be interpreted as a lightweight mixture-of-experts (MoE) model. In this formulation, the clustering module acts as a gating mechanism that partitions the feature space into stylistically coherent regions. Each cluster is associated with a dedicated classifier, which functions as an expert specialized for that region.

During inference, each input sample is assigned to its nearest cluster centroid based on feature similarity, and the corresponding expert model is used to generate predictions. This design enables specialization across heterogeneous coding styles while avoiding the computational overhead of traditional neural MoE architectures.

As illustrated in Figure 2, the system integrates feature extraction, clustering-based routing, and expert-level classification into a unified pipeline.

4.3 Expert Models and Ensemble

Within each cluster, logistic regression and multinomial Naïve Bayes are trained. Logistic regression provides stable linear decision boundaries in high-dimensional space, while Naïve Bayes performs well under sparse lexical distributions (McCallum and Nigam, 1998).

Predicted probabilities from both models are averaged to produce cluster-level outputs. This ensemble reduces variance and stabilizes predictions.

4.4 Threshold Calibration

A final calibration step selects a decision threshold that maximizes the macro F1-score on validation data. Because probability distributions are not naturally aligned with the evaluation metric, calibration prevents bias toward dominant code styles and ensures balanced performance.

5 Experimental Setup

5.1 Data and Evaluation

Experiments follow the official training, validation, and test splits. Model parameters are learned on the training set, hyperparameters and thresholds are tuned on validation data, and the test set remains unseen during development. Performance is evaluated using the macro F1-score.

Due to memory constraints, a stratified subset of 80,000 training samples is used while preserving class balance. This choice reflects a practical trade-off between computational cost and dataset coverage. While this subset was sufficient for stable training in our setup, it is not necessarily optimal, and performance may vary with different subset sizes.

5.2 Preprocessing and Features

Code snippets are treated as raw character sequences without language-specific tokenization or normalization. Empty entries are removed prior to processing.

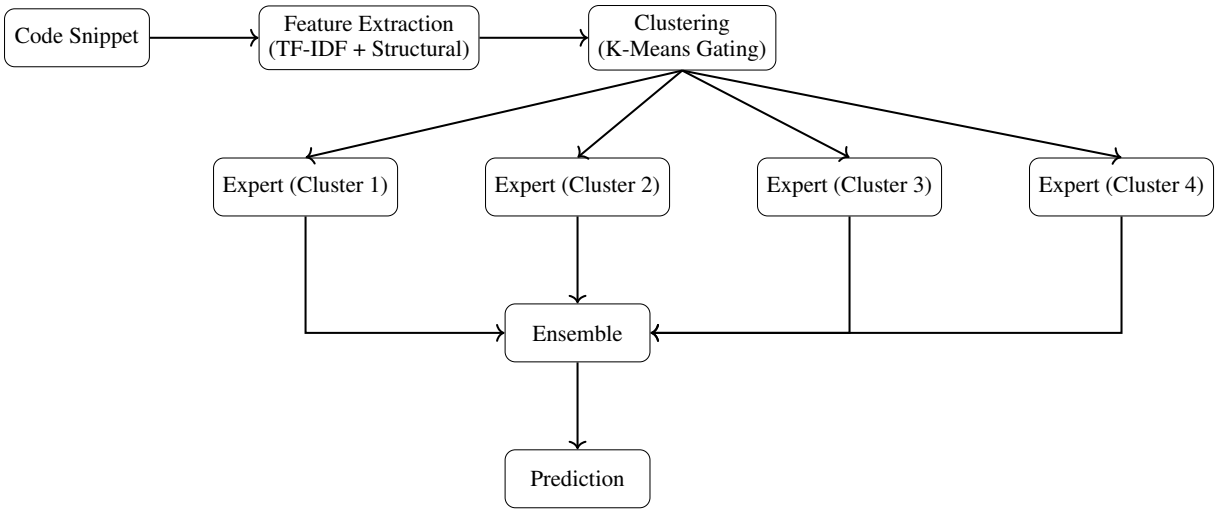

Figure 2: Overview of the proposed mixture-of-experts pipeline. Code snippets are transformed into stylometric features, routed via clustering (acting as a gating mechanism), and processed by cluster-specific experts before ensemble aggregation produces the final prediction.

Character-level TF-IDF features are extracted using 3-5 gram ranges with a maximum vocabulary of 120,000 and minimum document frequency of 5. Structural features (length, line count, comment ratio, digit ratio, punctuation ratio, lexical diversity) are standardized and concatenated with TF-IDF vectors.

5.3 Model Configuration

MiniBatch K-means clustering is applied with $K = 4$ in TF-IDF space. Within each cluster, logistic regression (L2 regularization, max 3000 iterations) and multinomial Naïve Bayes ($\alpha = 0.5$) are trained. Cluster-level probabilities are averaged for ensemble prediction. Random seeds are fixed for reproducibility.

5.4 Threshold Calibration

Decision thresholds are selected on the validation set by scanning the range $[0.70, 0.99]$ and choosing the value that maximizes the macro F1-score. The selected threshold is applied unchanged during test inference.

5.5 Implementation

All experiments are conducted in Python using `scikit-learn` on a Google Colab environment with a T4 GPU. MiniBatch clustering and batched TF-IDF processing are used to operate under limited memory constraints.

6 Results

6.1 Main Performance

The proposed pipeline achieves competitive performance on SemEval-2026 Task 13 Subtask A. On the official test set, our model achieves a macro F1-score of **0.6125**, demonstrating strong generalization under out-of-distribution conditions.

The system is optimized using the validation split, while the test set remains strictly unseen during development. Because the task emphasizes cross-domain robustness, design decisions prioritize stable structural signals over highly dataset-specific patterns.

Despite relying on lightweight stylometric features and linear classifiers, the model remains competitive with more complex approaches, highlighting the effectiveness of structural representations for machine-generated code detection.

6.2 Ablation Study

To assess the contribution of individual components, an ablation study is performed on the validation split, modifying one component at a time while keeping others fixed.

Removing clustering drops performance from 0.639 to 0.619, indicating that a single global classifier fails to capture stylistic heterogeneity. Expert specialization improves separation across diverse code styles.

Excluding structural features slightly increases validation performance (0.645), suggesting that lexical signals dominate on this split. However, this

Variant	Macro F1
Full Model (MoE + Structural + Ensemble + Calibration)	0.639
w/o Clustering	0.619
w/o Structural Features	0.645
Logistic Regression Only	0.640
w/o Calibration	0.338

Table 1: Ablation study on the validation split showing the contribution of each component in the proposed pipeline.

improvement is likely due to mild overfitting to the validation distribution. Structural features capture more stable patterns that generalize better under distribution shift, which is central to the SemEval evaluation setting.

Replacing the ensemble with a single logistic regression model yields comparable performance (0.640 vs 0.639), indicating that linear decision boundaries over TF–IDF features already capture the primary separability between classes. The Naive Bayes component provides limited additional gain, as both models operate on similar feature representations. Its contribution is mainly in smoothing probability estimates across clusters, leading to more stable thresholding rather than significant improvements in raw classification performance.

The largest drop occurs without threshold calibration (0.338), highlighting its critical role in handling class imbalance and aligning decision thresholds with the evaluation metric.

Overall, lexical features provide the core signal, clustering handles heterogeneity, and calibration is essential for stable performance.

While validation performance is higher, the drop on the test set reflects the difficulty of out-of-distribution generalization, which is the primary focus of the SemEval task.

6.3 Error Analysis

Error analysis is performed on the validation split to understand failure modes. Out of 1000 samples, 241 (24.1%) are misclassified, with 110 false positives and 131 false negatives.

Length plays a key role. False positives are much longer on average (2146.7) than both false negatives (1399.1) and correct samples (1252.7), indicating that long, well-structured human code is often mistaken for generated code. Short snippets (<200 chars) form only a small fraction of errors ($\approx$ 3-5%) , suggesting that errors are driven more by structure than lack of context.

Punctuation shows a consistent pattern. False negatives have slightly lower punctuation density (0.1373 vs 0.1410), while false positives are even lower (0.1318), pointing to confusion between simple generated code and clean, template-like human code. Comment ratios are marginally higher in errors (0.0022 vs 0.0018), suggesting that heavy annotation adds ambiguity. Qualitatively, false positives tend to be long and regular, while false negatives are short and less distinctive. Notably, errors have higher confidence (0.386 vs 0.354), indicating overconfident predictions rather than uncertainty.

Overall, errors are systematic and stem from structural similarity and overlapping stylistic cues, highlighting the limits of purely stylometric signals.

Positioning with Respect to Neural Baselines. Pretrained models such as CodeBERT and CodeT5 have shown strong results on related tasks, but they typically require significantly more compute and training effort.

In contrast, our approach uses simple stylometric features with linear models, making it more efficient and easier to interpret. Although we do not include a direct comparison here, the results suggest that these lightweight signals are already quite effective for this task. A more detailed comparison with neural baselines is left for future work.

7 Limitations

The proposed system relies primarily on structural signals and therefore depends on sufficiently strong structural cues in the input. Very short or minimal code snippets provide limited stylistic information, reducing classification reliability. Performance may also degrade when human-written and machine-generated code converge stylistically, as modern generators increasingly imitate compact formatting and human conventions.

The clustering-based mixture-of-experts framework assumes approximate separability of code patterns. Samples near cluster boundaries may receive inconsistent predictions when feature distributions overlap. In addition, the system does not incorporate semantic program understanding and operates exclusively on surface structure. While this design improves efficiency and cross-language robustness, it limits detection of generators that deliberately manipulate statistical signals. Future work may

benefit from integrating lightweight semantic features to complement stylometric modeling.

In addition, several structural features used in this work, such as code length and line count, may be influenced by dataset-specific characteristics. Differences between human-written and machine-generated samples can partly reflect variations in task complexity or dataset construction rather than purely intrinsic stylistic properties. As a result, these signals may not generalize consistently across datasets with different distributions. While combining them with lexical features (TF-IDF) helps reduce this reliance, some dependence on dataset-specific patterns remains.

8 Conclusion

We present a lightweight stylometric system for machine-generated code detection under out-of-distribution conditions in SemEval-2026 Task 13 Subtask A. Without relying on pretrained language models or neural architectures, our approach ranks 23rd using hybrid lexical-structural features, clustering-based specialization, ensemble fusion, and calibrated thresholds. Our findings demonstrate that persistent structural properties, such as verbosity, formatting behavior, and documentation patterns, provide robust cross-domain signals for AI-generated code detection. The results further highlight the importance of modeling stylistic heterogeneity and carefully calibrating decision thresholds when optimizing macro F1 under distribution shift. Future work may explore integrating lightweight semantic representations with stylometric modeling to further improve robustness against stylistic convergence.

9 Acknowledgments

We are grateful for the guidance of the SemEval-2026 Task 13 organisers. Authors are also thankful to the authorities of Birla Institute of Technology and Science, Pilani, to provide basic infrastructure facilities during the preparation of the paper.

References

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

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, and Ting Liu. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of EMNLP*.

Sebastian Gehrmann, Hendrik Strobelt, and Alexander Rush. 2019. Gltr: Statistical detection and visualization of generated text. *Proceedings of ACL System Demonstrations*.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Ganesh Jawahar, Muhammad Abdul-Mageed, and Laks V.S. Lakshmanan. 2020. Automatic detection of machine generated text: A critical survey. In *Proceedings of COLING*.

Moshe Koppel, Jonathan Schler, and Shlomo Argamon. 2009. Computational methods in authorship attribution. *Journal of the American Society for Information Science and Technology*, 60(1):9–26.

Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137.

Andrew McCallum and Kamal Nigam. 1998. A comparison of event models for naive bayes text classification. In *AAAI Workshop*.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. DetectGPT: Zero-shot machine-generated text detection using probability curvature. In *Proceedings of the 40th International Conference on Machine Learning (ICML 2023)*, volume 202 of *Proceedings of Machine Learning Research*, pages 24950–24962. PMLR.

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)*, pages 6913–6938, 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)*. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. In *OpenAI Technical Report*.

Juan Ramos. 2003. Using tf-idf to determine word relevance in document queries. In *ML Conference*.

Efstathios Stamatatos. 2009. A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP*.