MedHastra at SemEval-2026 Task 13: Stylometric Ensembles and Transformer Fine-Tuning for Robust AI Code Detection, Attribution, and Adversarial Analysis

Shruti Chandrasekar and Vedajanaani R S and Dr. P. Vijayalakshmi
SSN College of Engineering, Chennai, India
{shruti2210139, vedajanaani2310594, vijayalakshmip}@ssn.edu.in

Abstract

This paper presents Team MedHastra's submission to SemEval-2026 Task 13, which investigates detection and attribution of AI-generated code under increasingly realistic and adversarial conditions. We participated in all three subtasks: (A) binary AI-code detection under out-of-distribution (OOD) settings, (B) multi-class authorship attribution across 10 large language model (LLM) families, and (C) classification of human, AI-generated, hybrid, and adversarial code. For Subtask A, we propose a stylometric ensemble combining handcrafted structural features with word- and character-level TF-IDF representations. For Subtasks B and C, we fine-tune CodeBERT for multi-class classification, incorporating downsampling and class-weighted loss to address imbalance. Our official macro F1 scores are 0.3222 (70/81), 0.31425 (24/34), and 0.53134 (24/32) for Subtasks A, B, and C respectively. Our analysis reveals that shallow stylometric signals struggle under strong distribution shift, while transformer-based representations capture generator-specific and adversarial characteristics more effectively. Our implementation is publicly available.[1]

1 Introduction

1.1 Subtask A: OOD AI-Code Detection

Subtask A of SemEval-2026 Task 13 (1) investigates the robustness of AI-generated code detection under strong distribution shift. Rather than evaluating detection in homogeneous settings, this subtask emphasizes generalization across programming languages and domains. Such robustness is crucial as AI-generated code is increasingly integrated into educational assignments, industrial workflows, and open-source projects. Detection systems that rely on superficial lexical cues risk failing in realistic, heterogeneous environments.

To address this challenge, we adopt a stylometric ensemble approach. Our system extracts structural formatting patterns, lexical statistics, and character- and word-level TF-IDF features, which are combined using Random Forest, Gradient Boosting, and Logistic Regression through weighted soft voting. The design prioritizes language-agnostic stylistic indicators to encourage cross-language robustness.

Our final macro F1 score of 0.3222 (70/81) highlights the inherent difficulty of OOD detection. Performance degradation on unseen languages suggests that handcrafted stylistic signals alone may not sufficiently capture deeper generative characteristics required for strong cross-domain generalization.

1.2 Subtask B: Fine-Grained LLM Attribution

Subtask B extends detection to multi-class authorship attribution across ten LLM families in addition to human-written code (2). Fine-grained attribution is particularly relevant for academic integrity, hiring validation, and dataset curation, where simply detecting AI generation is insufficient. The task requires distinguishing subtle stylistic and structural differences between related model families.

We fine-tune CodeBERT to leverage contextual representations learned from large-scale code corpora. Inputs consist of programming language tags concatenated with truncated code snippets. Class imbalance is mitigated through downsampling, and optimization is performed using AdamW with linear scheduling. Transformer-based contextual modeling allows the system to capture higher-level structural dependencies beyond surface-level stylometry.

Our system achieved a macro F1 score of

[1]Our implementations are publicly available: Subtask B code: GitHub Repository. Subtasks A and C code: GitHub Repository.

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2098–2103

0.31425 (24/34). Although contextual modeling improved discrimination compared to shallow features, confusion between architecturally similar LLM families underscores the complexity of generator-level attribution.

1.3 Subtask C: Hybrid and Adversarial Detection

Subtask C introduces hybrid and adversarial authorship settings, reflecting realistic human–AI collaborative programming workflows (3). Hybrid samples combine human-authored and AI-modified segments, while adversarial samples are designed to imitate human style and evade detection systems. This task moves beyond binary detection toward nuanced modeling of blended authorship.

We reuse the CodeBERT backbone and incorporate class-weighted cross-entropy to address imbalance among hybrid and adversarial categories. Contextual encoding enables modeling of stylistic inconsistencies and structural blending within code segments.

Subtask C yielded our strongest results, achieving a macro F1 score of 0.53134 (24/32). These results indicate that contextual representation learning is particularly effective for nuanced authorship scenarios, where shallow lexical features are insufficient.

2 Background

2.1 Subtask A: Task Formulation and Dataset

Subtask A is formulated as a binary classification problem in which each instance consists of a source code snippet and its programming language. The objective is to predict whether the snippet is human-written (0) or AI-generated (1). The dataset is provided in parquet format with fields including `code`, `language`, `generator`, and `label`, though only the binary label is used for evaluation.

The training set primarily contains C++, Python, and Java code from algorithmic problem-solving domains. The test set introduces unseen programming languages such as Go, PHP, C#, C, and JavaScript, as well as unseen domains including research prototypes and deployed production code. This creates a structured out-of-distribution evaluation scenario. Macro-averaged F1 score is used as the evaluation metric to balance class performance.

AI-generated content detection has been widely studied in natural language processing (4; 5). However, code differs due to rigid syntactic constraints and formatting regularities. Stylometric techniques traditionally used for authorship attribution (6; 7) have recently been adapted for code analysis. Subtask A extends this line of work by explicitly evaluating cross-language and cross-domain robustness.

2.2 Subtask B: Attribution Setting

Subtask B extends the binary setting to 11-class classification, where labels correspond to human-written code or one of ten LLM families. The dataset contains approximately 45 generators distributed across these families. The evaluation includes both generators seen during training and unseen generators within known families, thereby testing generalization beyond direct memorization.

Authorship attribution has a long history in computational linguistics (7). Recent work on model signature identification and watermarking (8) explores embedding signals during generation, but Subtask B assumes no such embedded markers. Instead, attribution must rely solely on observable stylistic and structural characteristics. Pretrained transformer models such as CodeBERT (9) provide contextual representations suitable for capturing these signals.

2.3 Subtask C: Hybrid and Adversarial Setting

Subtask C introduces four labels: human-written, fully AI-generated, hybrid, and adversarial. Hybrid samples reflect collaborative human–AI development, while adversarial samples are designed to evade detection by mimicking human stylistic patterns. The dataset structure mirrors the previous subtasks, with train, validation, and test splits and macro F1 evaluation.

Hybrid detection challenges the assumption of mutually exclusive authorship, requiring models to detect stylistic blending. Research on adversarial robustness and model behavior (10) demonstrates how generative models can adapt to evade detection mechanisms. Subtask C operationalizes these concerns within a controlled benchmark for evaluating robustness against blended and adversarial generation.

3 System Overview

In this section, we describe the modeling strategies adopted for each subtask. Although all three subtasks concern AI-generated code detection and attribution, the varying levels of difficulty and evaluation settings required different modeling decisions. We deliberately designed separate systems tailored to each subtask's objectives rather than enforcing a single unified architecture.

3.1 Subtask A: Stylometric Ensemble for OOD Detection

Subtask A presents a challenging out-of-distribution (OOD) binary classification problem. The primary difficulty arises from distribution shift across programming languages and domains. Because the test set includes unseen languages and unseen domains, we avoided reliance on deep pretrained models trained primarily on high-resource languages and instead adopted a stylometric and structural modeling approach.

Our feature extraction pipeline transforms raw source code into three complementary feature spaces: structural statistics, lexical indicators, and n-gram representations.

First, structural statistics capture formatting and syntactic organization. Given a code snippet c, we compute features such as total character length $L(c)$, number of lines $N_l(c)$, average line length $L(c)/N_l(c)$, whitespace ratio, indentation mean and variance, punctuation density, and bracket counts. These features aim to capture stylistic regularities independent of programming language semantics.

Second, lexical indicators capture usage patterns of language-agnostic keywords such as `if`, `for`, `return`, `class`, and `function`. Identifier statistics are computed by extracting all tokens matching the regular expression for valid identifiers. We calculate average identifier length, maximum identifier length, and the number of unique identifiers. These reflect naming diversity and verbosity differences between human and machine-generated code.

Third, we construct TF-IDF representations. Let V_w denote the word-level vocabulary extracted from training data with n-gram range $(1, 3)$ and V_c denote the character-level vocabulary with n-gram range $(3, 5)$. For each snippet c, the TF-IDF representation is computed as:

$$\text{TF-IDF}(t, c) = \text{tf}(t, c) \cdot \log \frac{N}{df(t)}$$

where t is a token, N is the total number of training documents, and $df(t)$ is the document frequency.

The final feature vector is formed by concatenation:

$$X(c) = [X_w(c) \mid\mid X_c(c) \mid\mid X_s(c)]$$

where X_w is the word-level TF-IDF vector, X_c is the character-level TF-IDF vector, and X_s is the stylometric feature vector.

We train three classifiers: Random Forest, Gradient Boosting, and Logistic Regression (11). Logistic Regression provides a linear decision boundary that often generalizes well under distribution shift. Random Forest and Gradient Boosting introduce nonlinear modeling capacity to capture feature interactions.

For each classifier i, we obtain predicted probabilities $P_i(y = 1|X)$. The final ensemble probability is computed as:

$$P(y = 1|X) = \sum_{i=1}^{3} w_i P_i(y = 1|X)$$

where weights w_i are empirically selected. A decision threshold τ is chosen to maximize macro F1 on validation data.

As an example, consider a Python function implementing breadth-first search. The pipeline extracts structural statistics (indentation depth and loop frequency), computes TF-IDF features for tokens such as `queue` and `while`, and produces a feature vector passed to the ensemble for final classification.

The primary challenge addressed here is language shift. Character n-grams provide partial robustness because indentation and punctuation patterns often transcend language syntax. However, we acknowledge that handcrafted features inherently struggle to capture deeper generative priors encoded in LLM outputs.

3.2 Subtask B: Transformer Fine-Tuning for Multi-Class Attribution

Subtask B requires fine-grained discrimination among 11 classes. Stylometric features are insufficient for this task because LLM families may share

similar surface-level formatting but differ in contextual and structural tendencies.

We fine-tune CodeBERT (9), a transformer pretrained on large-scale code corpora. Given programming language ℓ and code snippet c, we construct input sequence:

$$x = [\ell \,||\, c]$$

The sequence is tokenized and truncated to 256 tokens. CodeBERT produces contextual representations:

$$H = \text{CodeBERT}(x)$$

The classification head computes logits:

$$z = Wh_{CLS} + b$$

and predicted probabilities:

$$\hat{y} = \text{softmax}(z)$$

The loss function is standard cross-entropy:

$$\mathcal{L} = -\sum_{k=0}^{10} y_k \log \hat{y}_k$$

Because the human class dominates the dataset, we downsample human examples during training to reduce imbalance. Optimization is performed using AdamW with learning rate 2×10^{-5} and linear warmup scheduling.

The key modeling decision here is leveraging pretrained contextual embeddings. Transformer self-attention layers capture long-range dependencies and stylistic coherence patterns such as comment phrasing, function decomposition style, and error-handling patterns. These signals are difficult to represent using shallow n-gram features.

For example, if an OpenAI-family model frequently structures functions with detailed docstrings followed by segmented logic blocks, the contextual encoder may implicitly encode such structural regularities across tokens. This allows discrimination even when lexical overlap exists between families.

3.3 Subtask C: Weighted Transformer for Hybrid and Adversarial Detection

Subtask C introduces hybrid and adversarial classes, which require sensitivity to subtle stylistic blending. We reuse the CodeBERT backbone from Subtask B but modify the training objective to incorporate class weighting.

Let $C = \{0, 1, 2, 3\}$ denote the four classes. Because hybrid and adversarial samples are underrepresented, we compute class weights:

$$w_k = \frac{N}{|C| \cdot N_k}$$

where N_k is the number of samples in class k and N is the total number of samples.

The weighted cross-entropy loss becomes:

$$\mathcal{L} = -\sum_{k=0}^{3} w_k y_k \log \hat{y}_k$$

This ensures minority classes contribute proportionally larger gradients during optimization.

Hybrid detection requires recognizing stylistic discontinuities within a single snippet. Transformer self-attention enables modeling of token-level dependencies across distant code segments. For instance, a human-written function header followed by AI-style verbose implementation may produce conflicting contextual signals. The model learns to associate such blended representations with the hybrid class.

Adversarial samples, designed to mimic human style, challenge shallow feature detectors. However, contextual embeddings capture deeper generative signatures such as repetitive phrasing patterns or consistent indentation behaviors characteristic of LLM outputs.

Overall, Subtask C benefits most from transformer-based modeling, as it demands nuanced contextual reasoning beyond surface-level statistics.

4 Experimental Setup and Results

Experiments were conducted using the official train, validation, and test splits provided for SemEval-2026 Task 13 (1; 2; 3). The validation sets were used for hyperparameter tuning, and final evaluation was performed through leaderboard submissions on the hidden test set. Macro-averaged F1 served as the evaluation metric for all subtasks.

For Subtask A, features were extracted directly from raw code without language-specific parsing. Word-level TF-IDF used up to 3000 features with n-gram range $(1, 3)$, while character-level TF-IDF used 2000 features with n-gram range $(3, 5)$. Tokens with document frequency below 2 or above

0.95 were filtered. Random Forest (300 trees, max depth 30), Gradient Boosting (200 estimators, learning rate 0.1), and Logistic Regression with L_2 regularization were trained on the concatenated feature vectors. Ensemble weights and classification thresholds were selected using validation macro F1. Three official runs were submitted with minor threshold variations.

For Subtasks B and C, we fine-tuned `microsoft/codebert-base` using the HuggingFace Transformers library. Input sequences were truncated to 256 tokens and prefixed with the programming language tag. Training was conducted for three epochs using AdamW with learning rate 2×10^{-5} and linear warmup over 10% of training steps on NVIDIA Tesla T4 GPUs. Batch sizes were 8 (Subtask B) and 12 (Subtask C). Human-class downsampling was applied in Subtask B, while Subtask C used class-weighted cross-entropy.

Table 1 summarizes official test results.

Subtask	Macro F1	Rank	Total Teams
A	0.3222	70	81
B	0.31425	24	34
C	0.53134	24	32

Table 1: Official leaderboard performance.

Subtask A showed significant performance drop under unseen language and domain conditions, reflecting the challenge of OOD generalization. Subtask B achieved moderate performance but exhibited confusion among closely related LLM families. Subtask C obtained the strongest results, indicating that contextual transformer representations are particularly effective for hybrid and adversarial detection.

5 Error Analysis

For Subtask A, errors were predominantly concentrated on unseen programming languages and non-algorithmic domains. The stylometric ensemble struggled when formatting conventions, indentation styles, or comment structures differed substantially from training data. In particular, research-oriented code containing extensive documentation or unconventional structure was frequently misclassified as AI-generated. These patterns suggest that handcrafted structural features lack robustness under strong distribution shift.

In Subtask B, the primary source of error was confusion among closely related LLM families. Models trained on similar data distributions or sharing architectural similarities exhibited overlapping stylistic characteristics, leading to misattribution. While contextual embeddings captured high-level generation tendencies, subtle distinctions such as comment phrasing or variable naming patterns were not always sufficient for reliable separation.

In Subtask C, hybrid samples were the most challenging category. Code segments partially modified by LLMs were often predicted as fully AI-generated, indicating that AI-style segments dominate contextual representations. Adversarial samples occasionally mimicking concise human coding styles also reduced classification confidence. These findings suggest that modeling intra-snippet stylistic transitions may improve performance in blended authorship scenarios.

6 Conclusion

In this paper, we present the Team MedHastra's systems for SemEval-2026 Task 13, covering binary AI-code detection under out-of-distribution settings, multi-class LLM attribution, and hybrid/adversarial code classification. Our approaches combined stylometric feature engineering with ensemble learning for Subtask A and transformer-based fine-tuning of CodeBERT for Subtasks B and C.

The results highlight the varying difficulty across tasks. Stylometric modeling achieved limited robustness under strong distribution shift in Subtask A, underscoring the challenge of cross-language and cross-domain generalization. Transformer-based contextual modeling demonstrated improved performance for fine-grained attribution in Subtask B and yielded the strongest results for hybrid and adversarial detection in Subtask C. Error analysis revealed that unseen languages, closely related LLM families, and blended authorship patterns remain key challenges.

Overall, our findings emphasize the importance of contextual representation learning for nuanced AI-code detection and attribution. Future work may explore contrastive training for generator discrimination, domain-adaptive pretraining to improve OOD robustness, and token-level modeling to better capture stylistic transitions within hybrid code samples.

References

[1] D. Orel, D. Azizov, I. Paul, and Y. Wang. SemEval-2026 Task 13 Subtask A: Detecting AI-Generated Code in OOD Settings. Kaggle Competition, 2025. https://kaggle.com/competitions/sem-eval-2026-task-13-subtask-a.

[2] D. Orel, D. Azizov, I. Paul, and Y. Wang. SemEval-2026 Task 13 Subtask B: Code Authorship Identification. Kaggle Competition, 2025. https://kaggle.com/competitions/sem-eval-2026-task-13-subtask-b.

[3] D. Orel, D. Azizov, I. Paul, and Y. Wang. SemEval-2026 Task 13 Subtask C: Detection of AI-Generated, Hybrid and Adversarial Code. Kaggle Competition, 2025. https://kaggle.com/competitions/sem-eval-2026-task-13-subtask-c.

[4] S. Gehrmann, H. Strobelt, and A. M. Rush. GLTR: Statistical Detection and Visualization of Generated Text. In *Proceedings of ACL*, 2019. https://aclanthology.org/P19-3019/.

[5] E. Mitchell, Y. K. Lee, A. Khazatsky, C. D. Manning, and C. Finn. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. arXiv preprint arXiv:2301.11305, 2023. https://arxiv.org/abs/2301.11305.

[6] M. Koppel, J. Schler, and S. Argamon. Computational Methods in Authorship Attribution. *Journal of the American Society for Information Science and Technology*, 60(1):9–26, 2009.

[7] E. Stamatatos. A Survey of Modern Authorship Attribution Methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556, 2009.

[8] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein. A Watermark for Large Language Models. arXiv preprint arXiv:2301.10226, 2023. https://arxiv.org/abs/2301.10226.

[9] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, and T. Liu. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of EMNLP*, 2020. https://aclanthology.org/2020.findings-emnlp.139/.

[10] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. B. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting Training Data from Large Language Models. In *USENIX Security Symposium*, 2021. https://arxiv.org/abs/2012.07805.

[11] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant. *Applied Logistic Regression*, 3rd edition. Wiley, 2013.

[12] Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 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, San Diego, USA, June 2026.