MIUN BiasPatrol at SemEval-2026 Task 13: Why TF-IDF can Beat Transformers for OOD Code Detection

Loviza Sahlén[1], Thomas Springfeldt[1], Mehwish Fatima[2], Raja Khurram Shahzad[1,3]

[1]Department of Communication, Quality Management and Information Systems
Mid Sweden University, Ostersund Campus, Sweden.
{losa2300, thsp1900}@student.miun.se

[2]School of Electrical Engineering and Computer Science
National University of Sciences and Technology, Islamabad, Pakistan.
mehwish.fatima@seecs.edu.pk

[3]Department of Computer Science, Electrical and Space Engineering,
Lulea University of Technology, 97187 Luleå, Sweden.
raja-khurram.shahzad@miun.se

Abstract

The increasing use of AI-generated code underscores the need for effective detection systems. However, their performance often deteriorates when faced with distribution shifts. This paper presents our system for SemEval-2026 Task 13:A, which focuses on binary classification of human-written versus machine-generated code across various programming languages and domains. We systematically compare traditional classifiers, such as Random Forest and XGBoost, which utilize statistical and TF-IDF features, against pipelines that leverage frozen embeddings from advanced code transformers like UniXcoder and GraphCodeBERT. Our results reveal a notable trade-off, i.e., while transformer-based pipelines excel in in-distribution validation (reaching up to 0.89 Macro F1), they experience severe performance drops up to 77%; when applied to out-of-distribution languages and domains. In contrast, models employing TF-IDF with tree-based classifiers demonstrate significantly greater stability. We identify this fragility as a result of a bias toward superficial formatting, particularly whitespace, which is accentuated by transformers. By implementing simple space normalization, we enhance the out-of-distribution robustness of traditional models; however, this also highlights the ongoing dependence of embeddings on these non-semantic features. Our findings suggest that for creating generalizable code detection systems, straightforward, well-normalized lexical features may be more reliable than complex, unrefined embeddings.

1 Introduction

The distinction between human-written code and machine-generated code is becoming increasingly unclear. While AI assistants can enhance developer productivity, they also introduce new challenges related to academic integrity, software security, and code provenance. Detecting machine-generated code is a pressing concern, made more complex by the variety of real-world deployment scenarios. For example, detection systems must effectively tackle new programming languages, domains, and AI generators that may not have been encountered during their training (Orel et al., 2025a). This paper presents our submission to SemEval-2026 Task 13:A, a binary classification task specifically designed to evaluate generalization in out-of-distribution (OOD) settings. This task builds on previous research focused on detecting machine-generated code (Orel et al., 2025b,a).

Current approaches range from traditional machine learning (ML) methods that rely on handcrafted features (Bishop, 2006) to fine-tuned transformer models like CodeBERT (Feng et al., 2020) and UniXcoder (Guo et al., 2022). While these latter models typically perform well on in-distribution (ID) benchmarks, their performance often declines significantly when faced with even minor distributional shifts (Orel et al., 2025b). This decline is usually attributed to overfitting (Hastie et al., 2009) or the exploitation of dataset biases (Torralba and Efros, 2011), which can include issues like formatting artifacts.

We hypothesize that the representational power of large models can be a double-edged sword, capturing superficial patterns that may limit their effectiveness in new contexts. To investigate this, we conduct a controlled comparison by evaluating four classifiers: Logistic Regression (LogReg), Linear Support Vector Machine (SVM), Random Forest (RF), and XGBoost. We assess these classifiers across three feature categories: (1) handcrafted sta-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2469–2474

tistical features, (2) Term Frequency-Inverse Document Frequency (TF-IDF) lexical features, and (3) frozen embeddings from UniXcoder and GraphCodeBERT. Our primary contribution is a detailed analysis of these classifiers' OOD generalization on the SemEval task. Our results indicate that while transformer models perform exceptionally well in in-distribution settings, simpler TF-IDF features demonstrate significantly greater robustness. We identify whitespace patterns as a primary source of bias and demonstrate that, although space normalization improves performance, it does not completely eliminate the generalization gap for complex embeddings. Our findings suggest a focus on robust, language-agnostic features rather than relying solely on model capacity for real-world applications.

Split	Total	Human	AI
Training	500,000	238,475	261,525
Validation	100,000	47,695	52,305
Test sample	1,000	777	223

Table 1: Dataset splits and class distribution.

2 Related Work

Large Language Models are increasingly capable of generating source code, which raises several concerns. These include the potential for plagiarism evasion through paraphrasing-style transformations (Park et al., 2025; Lee et al., 2025), the memorization of copyrighted snippets (Karamolegkou et al., 2023), and a heightened risk of vulnerabilities in AI-generated programs (Pearce et al., 2022; Tihanyi et al., 2024). Human code also has its limitations; in complex scenarios, developers can make mistakes that may not be immediately evident (Elahi and Wang, 2024). This highlights the importance of provenance and accountability, regardless of whether the code is produced by humans or generated by models. As a result, there has been extensive work on source code author analysis and attribution. Two major challenges in this field are Out-Of-Distribution generalization and robustness against semantics-preserving adversarial edits. Several frameworks have demonstrated strong performance in addressing these issues, including language-oblivious identification (Abuhamad et al., 2025), adversarially motivated defenses such as RoPGen (Li et al., 2022), OOD-oriented detectors like CodeT5-Authorship (Bisztray et al., 2026) and CoDet-M4 (Orel et al., 2025a), large-scale evaluation resources such as Droid (Orel et al., 2025b), and contrastive stylometric verification (CLAVE) (Álvarez Fidalgo and Ortin, 2025). However, later studies have shown that even small perturbations, such as inserting unreachable code, can significantly disrupt author representations (Abuhamad et al., 2025). Moreover, more realistic definitions of OOD, like project, directory, or collaboration shifts, can drastically reduce accuracy (Bogomolov et al., 2021). Furthermore, many detectors rely on superficial cues, such as punctuation, identifier length, and empty lines, which can render them ineffective against style-mimic transformations (Orel et al., 2025b,a).

To contribute to solving the problem, in this work, we use controlled whitespace normalization as a targeted robustness probe and systematically compare statistical, TF-IDF, and frozen transformer-embedding pipelines. This approach aims to quantify how much of the apparent performance is due to formatting artifacts versus features that generalize to unseen languages and domains.

3 Task and Dataset

The dataset for SemEval-2026 Task 13: A[1] is designed to evaluate OOD generalization. The training and validation sets consist entirely of algorithmic code written in Python, Java, and C++. In contrast, the test set includes unseen programming languages (C#, JavaScript, Go, C, and PHP) as well as unseen domains (research and production code). This setup encourages models to rely on transferable signals rather than language-specific patterns.

Table 1 summarizes the dataset splits. The training set is balanced by class but heavily imbalanced by language, with Python accounting for over 91% of the samples. The test set features a different class distribution, comprising 77.7% human-generated code, and showcases a more diverse mix of languages, presenting a challenging out-of-distribution evaluation.

3.1 Data Curation

The initial exploratory analysis revealed inconsistencies in the samples we examined. Some contained code that did not match their assigned language labels, while others had no detectable code

[1]https://github.com/mbzuai-nlp/SemEval-2026-Task13

at all (e.g., they included only text). This inconsistency is critical for models that rely on language-specific parsers, such as Tree-Sitter[2], for feature extraction. To ensure reliable parsing for our statistical feature pipelines, we created a *clean* version of the training and validation sets by removing samples where the language label did not align with the code or where no code was detected. For models that utilize TF-IDF or embeddings and do not depend on a parser, we chose to retain these mismatched samples to introduce controlled noise and potentially enhance robustness.

Moreover, we removed samples that contained fewer than 200 characters or fewer than 6 lines, as well as those exceeding 6000 characters or 300 lines. This allowed us to focus on more substantive code snippets. The resulting curated datasets maintained a moderate class imbalance, with AI-generated samples comprising 55-57% of the total data.

Baseline	Val F1	Test F1
RF + Stat	76.55	35.60
SVM + Stat	71.42	47.13
XGBoost + Stat	77.90	31.02
RF + TF-IDF	68.72	**56.90**
SVM + TF-IDF	64.88	49.87
XGBoost + TF-IDF	68.81	54.40
SVM + UniXcoder	87.40	49.82
XGB + UniXcoder	84.99	28.36
RF + GraphCodeBERT	84.69	19.59
XGB + GraphCodeBERT	89.47	23.09

Table 2: Performance (Macro F1) on Validation (ID) and Test (OOD) sets after space normalization.

4 Methodology

Our approach systematically compares feature representations using standard classifiers. All the code and experimental results are fully reproducible. They can be accessed on GitHub[3] and are also available upon request.

4.1 Feature Representations

We evaluate three distinct families of features:

Statistical Features We extract nine handcrafted features from each code snippet, based on prior research (Orel et al., 2025a). These features include code length, line count, average line length, whitespace ratio, function count, import density, comment density, maximum nesting depth (calculated using Tree-Sitter), and average identifier length. To ensure consistency, we normalize these features to the range [0, 1] using a MinMaxScaler fitted to the training data.

TF-IDF Lexical Features We employ a TF-IDF vectorizer to determine the significance of lexical tokens. Notably, we replace the standard tokenization process with the GraphCodeBERT (Guo et al., 2021) tokenizer, as its subword vocabulary is better suited for capturing stylistic coding nuances (Čerňanský and Petrík, 2025). The vectorizer is first trained on the training set and then applied to the validation and test sets.

Transformer-based Embeddings To capture deep semantic structures, we utilize two pre-trained code transformers as frozen feature extractors, i.e., UniXcoder (Guo et al., 2022) and GraphCodeBERT (Guo et al., 2021). For each snippet, we extract the 768-dimensional [CLS] token embedding from the final layer after tokenization, ensuring that snippets are truncated or padded to 512 tokens. All embeddings are then L2-normalized per sample. This frozen approach allows us to assess the inherent generalizability of the pre-trained representations without any task specific fine-tuning.

Baseline	Accuracy	Macro F1
RF + Stat	85.56	85.47
XGBoost + Stat	90.78	90.70
RF + TF-IDF	69.38	65.03
SVM + UniXcoder	94.00	93.89
XGB + GraphCodeBERT	93.95	93.80

Table 3: Performance on validation set *before* space normalization.

4.2 Model Architectures and Training

For each feature set, we train four standard classifiers, i.e., LogReg, SVM, RF, and XGBoost. Hyperparameters are selected through a manual grid search on the validation set to maximize the Macro F1 score. The training pipeline, implemented in

[2]https://tree-sitter.github.io/tree-sitter/
[3]https://github.com/losa2300/dt166g-group5

Python, follows a modular registry-based architecture, ensuring consistent data loading, training, and evaluation across all combinations of models and features.

4.3 Zero-shot and Transformer Models

Our primary experiments use frozen embeddings with standard classifiers. However, we also explored zero-shot classification using large pre-trained transformers, such as DeBERTa-v3-large and GraphCodeBERT. These models use prompt-based inference to classify code authorship directly, without fine-tuning, using the Hugging Face Transformers library's pipelines (Wolf et al., 2020). While these zero-shot methods demonstrated reasonable performance on in-distribution data, they performed poorly on out-of-distribution samples, often resulting in performance comparable to random guessing. As a result, we have excluded these approaches from our main quantitative comparison. This outcome reinforces our central finding: without adaptation, even large pre-trained models struggle to generalize across different programming languages and domains when surface-level patterns change.

Baseline	Accuracy	Macro F1
RF + Stat	36.00	35.85
XGBoost + Stat	32.80	32.07
RF + TF-IDF	26.50	24.79
SVM + UniXcoder	48.70	48.15
XGB + GraphCodeBERT	40.80	40.80

Table 4: Performance on test set *before* space normalization.

4.4 Evaluation and Robustness Analysis

The primary performance metric we use is the Macro F1 score, which serves as the official score for the task and treats both classes equally. Moreover, we monitor accuracy, precision, recall, and AUC-ROC to provide a comprehensive overview of our results (Fawcett, 2006). Our primary analysis involves comparing performance on the ID validation set with that on OOD test samples.

A key aspect of our methodology is the targeted robustness analysis. Initial feature importance scores from our statistical models indicated that the whitespace ratio was the most predictive feature, which suggested a clear bias in the dataset. To quantify this bias and assess its impact on generalization, we conducted all experiments on a modified version of the dataset where we normalized the code by removing extra whitespace, reducing it to single spaces and applying standard indentation. This approach allows us to measure how much of the model's performance, as well as its degradation on OOD data, can be attributed to superficial formatting.

Baseline	Val F1	Test F1	Drop (%)
RF + Stat	76.55	35.60	53.5
RF + TF-IDF	68.72	56.90	17.2
SVM + UniXcoder	87.40	49.82	43.0
XGB + GraphCode-BERT	89.47	23.09	74.2

Table 5: Macro F1 degradation from validation to OOD test.

5 Results

We present our experimental results, focusing on the effects of feature types and space normalization on OOD generalization.

Table 2 illustrates a significant trade-off in model performance. Transformer-based pipelines achieve the highest ID validation scores, reaching 0.89. However, their performance suffers dramatically on the OOD test set. In contrast, simpler TF-IDF models, especially when combined with Random Forest, demonstrate the best OOD performance, scoring 0.57 in Macro F1, despite having comparatively modest ID scores.

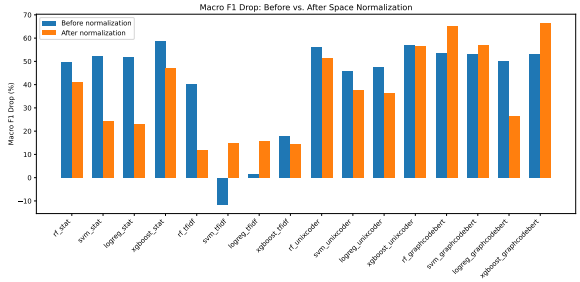

Figure 1: Macro F1 drop from validation to test, before and after space normalization.

Figure 1 illustrates the impact of our bias analysis. For traditional models (, such as RF+Stat and RF+TF-IDF, space normalization consistently reduces the decline in OOD performance. This

prompts these models to rely less on format-specific signals. However, the effect of normalization on transformer-based pipelines is more complex. Some models, like SVM+UniXcoder, demonstrate improved robustness after normalization, but their overall OOD F1 scores remain low. Notably, the degradation in XGBoost+GraphCodeBERT's performance after normalization indicates that its strong ID performance was largely dependent on learning spurious whitespace patterns. The full pre-normalization results, shown in Tables 3 and 4, confirm this trend. Specifically, XGB+GraphCodeBERT achieved a validation F1 score of 0.94 before normalization, which then dropped to 0.41 on the OOD test set.

6 Discussion

Our experiments yield three main findings. First, high performance on ID data does not guarantee robustness to OOD data. While transformer embeddings may perform well on a validation set, this does not necessarily indicate their effectiveness in real-world applications. Their capability might allow them to memorize characteristics specific to the training distribution, making them vulnerable to changes in the distribution.

Second, simpler lexical features can demonstrate greater generalizability. The better OOD performance of the TF-IDF + RF model suggests that normalized surface-level token distributions provide a more language-agnostic signal compared to deep semantic embeddings. This supports established concerns about overfitting and dataset bias (Hastie et al., 2009; Torralba and Efros, 2011), where complex models often learn irrelevant correlations that do not hold true in new domains.

Third, whitespace represents a widespread and significant bias. Our feature importance analysis and normalization experiments indicates that all models, particularly transformers, tend to rely on these superficial cues. While normalizing whitespace is a straightforward and effective solution for traditional models, it does not fully resolve the problem for embeddings, which may incorporate other, more subtle formatting biases. The significant decrease in GraphCodeBERT's performance after normalization indicates that its pre-training may unintentionally increase its sensitivity to non-semantic features.

These findings have practical implications. For the SemEval task, a well-tuned TF-IDF model serves as a robust baseline that should not be disregarded in favor of more complex approaches. More broadly, these results emphasize the critical need for evaluations and feature designs that focus on robustness. Advancements in code detection may rely less on developing larger models and more on understanding and mitigating the specific biases they exploit. While our research is limited by the use of frozen embeddings, fine-tuning could help transformers to eliminate some of these biases, though it also risks further overfitting. Future work should investigate advanced normalization techniques (such as identifier anonymization) and causal feature engineering to identify and eliminate sources of model brittleness.

7 Conclusion

We presented a comparative study for SemEval-2026 Task 13:A, which highlights a fundamental trade-off between in-distribution accuracy and out-of-distribution robustness in code authorship attribution. While transformer-based embeddings perform well on familiar datasets, they significantly struggle when faced with distribution shifts. This is primarily due to their over-reliance on superficial formatting elements, such as whitespace. In contrast, using simple TF-IDF features combined with tree-based classifiers and basic space normalization offers a much more stable and generalizable solution. Our findings caution against equating model complexity with real-world performance and underscore the lasting value of robust, language-agnostic feature design.

References

Mohammed Abuhamad, Changhun Jung, David Mohaisen, and DaeHun Nyang. 2025. SHIELD: Thwarting code authorship attribution. *IEEE Transactions on Dependable and Secure Computing*, 22(5):4753–4767.

Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer.

Tamas Bisztray, Bilel Cherif, Richard A. Dubniczky, Nils Gruschka, Bertalan Borsos, Mohamed Amine Ferrag, Attila Kovacs, Vasileios Mavroeidis, and Norbert Tihanyi. 2026. I know which llm wrote your code last summer: Llm generated code stylometry for authorship attribution. In *Proceedings of the 18th ACM Workshop on Artificial Intelligence and Security*, page 28–39. Association for Computing Machinery.

Egor Bogomolov, Vladimir Kovalenko, Yurii Rebryk, Alberto Bacchelli, and Timofey Bryksin. 2021. Authorship attribution of source code: a language-agnostic approach and applicability in software engineering. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, page 932–944. Association for Computing Machinery.

Haroon Elahi and Guojun Wang. 2024. Forward-porting and its limitations in fuzzer evaluation. *Information Sciences*, 662:120142.

Tom Fawcett. 2006. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Ming Feng, Ming Gong, Li Shou, Daxin Jiang, Ming Liu, 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. Association for Computational Linguistics.

Daya Guo, Zhangyin Feng, Lili Zhang, Xiaodong Wang, Yufeng Wu, Lei Chen, Dongmei Yu, and Ming Zhou. 2021. Graphcodebert: Pre-training code representations with data flow. In *Findings of the Association for Computational Linguistics: ACL 2021*, pages 3306–3317.

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. Association for Computational Linguistics.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. Springer.

Antonia Karamolegkou, Jiaang Li, Li Zhou, and Anders Søgaard. 2023. Copyright violations and large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7403–7412. Association for Computational Linguistics.

Jooyoung Lee, Toshini Agrawal, Adaku Uchendu, Thai Le, Jinghui Chen, and Dongwon Lee. 2025. PlagBench: Exploring the duality of large language models in plagiarism generation and detection. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7519–7534. Association for Computational Linguistics.

Zhen Li, Guenevere Qian Chen, Chen Chen, Yayi Zou, and Shouhuai Xu. 2022. Ropgen: Towards robust code authorship attribution via automatic coding style transformation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (*, pages 1906–1918.

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. 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.

Shinwoo Park, Hyundong Jin, Jeong-Won Cha, and Yo-Sub Han. 2025. Detecting code paraphrased by large language models using coding style features. *Engineering Applications of Artificial Intelligence*, 162:112454.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy*, pages 754–768.

Norbert Tihanyi, Tamas Bisztray, Mohamed Amine Ferrag, Ridhi Jain, and Lucas Cordeiro. 2024. How secure is ai-generated code: a large-scale comparison of large language models. *Empirical Software Engineering*, 30.

Antonio Torralba and Alexei Efros. 2011. Unbiased look at dataset bias. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, page 1521–1528. IEEE Computer Society.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, and Jamie Brew. 2020. Transformers: State-of-the-art natural language processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

David Álvarez Fidalgo and Francisco Ortin. 2025. CLAVE: A deep learning model for source code authorship verification with contrastive learning and transformer encoders. *Information Processing & Management*, 62:104005.

Bc. Richard Čerňanský and Ing. Juraj Petrík. 2025. Comparing machine learning and deep learning models with attention mechanisms for multiple source code authorship attribution. In *2025 International Conference on Computer Systems and Technologies*, pages 01–09.