AKCIT at SemEval-2026 Task 13: A Lightweight LightGBM Baseline for Cross-Language Detection of LLM-Generated Code

Rone C. Brandão Filho[1], Walcy S. R. Rios[1], Lucas L. Neves[1], José R. F. Oliveira[1], Diogo F. C. Silva[1], Arlindo R. Galvão Filho[1],
[1]Advanced Knowledge Center for Immersive Technologies (AKCIT)
Federal University of Goiás (UFG),
Correspondence: ronebrandao@discente.ufg.br

Abstract

The widespread use of LLMs in software development has made the detection of machine-generated code a pressing challenge, particularly when models must generalize across programming languages and domains. We present a lightweight, LLM-free pipeline that combines stylometric feature extraction with a LightGBM classifier and explicitly prioritizes structural generalization over deep semantic modeling. Despite its simplicity, the method achieves a Macro F1 of 0.70–0.72, more than doubling the CodeBERT baseline (0.30) in SemEval-2026 Task 13 Subtask A, while operating without GPUs or any fine-tuning.

1 Introduction

The rapid proliferation and integration of Large Language Models (LLMs) into software development has fundamentally transformed the landscape of source code production, with 82.1% of developers now using generative tools (Qodo, 2025). Although tools like GitHub Copilot and specialized models such as DeepSeek-Coder (Guo et al., 2024) have significantly improved developer productivity, their widespread adoption introduces critical challenges related to academic integrity, software provenance, and cybersecurity. In particular, the propagation of security vulnerabilities and insecure logic through AI-assisted development (Pearce et al., 2022), combined with the large-scale injection of synthetic code into public repositories—potentially triggering a feedback loop of degraded training data and model collapse in future neural systems (Shumailov et al., 2024)—has elevated the ability to distinguish between human-authored and machine-generated code to a pivotal research frontier in Natural Language Processing (NLP) and Software Engineering.

The SemEval-2026 Task 13 (Orel et al., 2026) addresses this challenge through the lens of robust generalization. Unlike traditional classification tasks where training and test distributions are often aligned, this task requires models to identify machine-generated code across unseen programming languages and distinct software domains (e.g., shifting from algorithmic competitive programming to research and production environments). This setting exposes a significant limitation in current State-of-the-Art (SOTA) approaches: deep neural architectures, while powerful, tend to overfit to the idiosyncratic semantic patterns of the dominant training language, failing to capture the underlying structural "fingerprint" of the generative process.

The dataset provided for Subtask A exemplifies these difficulties, exhibiting a severe class and language imbalance, with Python accounting for more than 91% of the training samples. Initial benchmarks using Transformer-based models, such as CodeBERT (Feng et al., 2020), yield a modest F1-score of approximately 0.30 on the test set, suggesting that these models struggle to generalize beyond the seen distributions. Furthermore, even sophisticated Out-of-Distribution (OOD) detection techniques, such as those treating human texts as outliers (Zeng et al., 2025), often incur prohibitive computational costs without providing proportional gains in cross-language robustness.

In this work, we propose a "Back to Basics" approach that prioritizes structural heuristics over deep semantic embeddings. We demonstrate that simple yet carefully engineered features extracted via Regular Expressions (Regex)—capturing stylistic markers such as operator spacing, naming conventions (e.g., *snake_case*), and comment density—provide a more stable signal for machine authorship than neural representations. Using a LightGBM classifier optimized through Bayesian search, our approach achieves an F1-score of 0.70–0.72, more than doubling the baseline performance.

Distinguishing human-authored from AI-generated code remains a challenge because

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3357–3362

source code is an inherently formal data type with low entropy. Existing text-based detectors often fail in this domain, as syntax rules and idiomatic constraints strictly limit the token variability that statistical models rely on. Effective detection requires revealing subtle "stylistic fingerprints"—such as mechanical uniformity and specific indentation patterns—that reflect machine regularity versus the idiosyncratic habits of human programmers.

Our participation in SemEval-2026 Task 13, specifically Subtask A, targets this detection challenge within a framework designed for a rigorous OOD evaluation. The task frames the detection as a supervised binary classification problem: identifying whether a code snippet is fully human-written (label 0) or fully machine-generated (label 1). The dataset provided by the organizers is notably unbalanced, with Python accounting for approximately 91% of the samples, which poses a significant risk of overfitting to language-specific artifacts. To address this, the task evaluates systems across two primary axes: programming languages (seen vs. unseen) and domains (algorithmic vs. research/production). This requires a methodology that prioritizes generalizable stylistic markers over language-specific semantics.

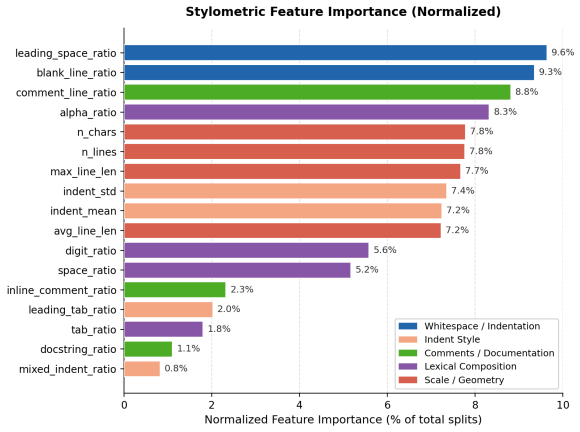

Figure 1: LightGBM feature importances for the 17 stylometric features, grouped by category. Whitespace and indentation signals dominate, confirming that formatting regularity is the primary discriminative cue for machine-generated code detection.

We propose an LLM-free, parameter-efficient pipeline leveraging deterministic Regular Expressions (REGEX) and a LightGBM classifier. Our approach positions strategic model refinement as an alternative to the prevailing trend of increasing parametric complexity. Using REGEX to extract features such as whitespace-to-code ratios and indentation consistency, our system captures discriminative stylistic markers with high transparency. This lightweight baseline enables rapid iteration and mitigates language-specific bias through strategic downsampling, ensuring robustness in out-of-distribution scenarios.

Our contributions are threefold:

1. We propose a parameter-efficient detection pipeline that takes advantage of deterministic REGEX features and a LightGBM classifier to distinguish human and machine authorship without the need for high-cost hardware or exhaustive model retraining.

2. We identify key REGEX-based stylistic markers, specifically those related to whitespace usage and indentation depth, which provide highly discriminative cues for identifying AI-generated code across multiple programming languages.

3. We demonstrate that a lightweight baseline can achieve competitive results with minimal computational overhead.

2 Background

2.1 Task Description

We participate in **SemEval-2026 Task 13** and focus on **Subtask A**, which frames *machine-generated code detection* as a supervised *binary classification* problem. Given a code snippet x, the system predicts $y \in \{0, 1\}$ that indicates whether x is **fully human-written** or **fully machine-generated**; hybrid categories are handled in other subtasks (Orel et al., 2026). The organizers report that Subtask A training data is drawn from an algorithmic/competitive-programming style domain and includes multiple languages (notably C++, Python and Java), while evaluation probes robustness under distribution shifts such as **unseen languages** and **unseen domains** (e.g., research/production), including their combination (Orel et al., 2026).

The systems are evaluated using **the macro-averaged F1 score** as the primary metric, as specified by the official scorer (Orel et al., 2026). Beyond the shared-task setting, the task is positioned within a broader research line on detecting AI-generated code across languages and domains.

2.2 Related Work

Detecting AI-generated code has been studied under increasingly realistic conditions where detectors must generalize across programming languages, generators, and domains. Orel et al. (2025a) introduce CoDet-M4, a benchmark for detecting machine-generated code in multi-lingual, multi-generator, and multi-domain settings, highlighting robustness challenges in domain transfer. Building on this line of work, Orel et al. (2026) propose a shared task that further evaluates detection under diverse programming languages, generators, and application scenarios. Orel et al. (2025b) present Droid, a resource suite for AI-generated code detection, and report that detector performance can degrade substantially under distribution shifts (e.g., unseen languages/domains) and under forms of "humanization" of model output, motivating more robust evaluation and training protocols.

A common baseline direction in code understanding tasks is to fine-tune transformer encoders pre-trained on code, such as CodeBERT (Feng et al., 2020), which learns representations from paired natural language and code. The SemEval Task 13 organizers explicitly allow the use of general-purpose or code-oriented pretrained models (including CodeBERT) and provide starter materials for participants (Orel et al., 2026).

In parallel, feature-engineered detectors remain competitive in some settings because they can combine sparse lexical signals with dense style and structural statistics. Gradient-boosted decision trees are a practical choice for such heterogeneous feature spaces; LightGBM is a widely used GBDT implementation designed for efficiency and scalability (Ke et al., 2017). Therefore, we position our LightGBM-based approach as a lightweight alternative to encoder fine-tuning, and empirically compare it under the shared-task evaluation protocol rather than assuming superiority (Orel et al., 2026).

3 Methods

Our approach shifts the focus from deep semantic modeling to a robust feature-engineering pipeline designed to mitigate the effects of extreme language imbalance and promote cross-domain generalization.

Given the severe Python dominance (>91%), we applied near-deduplication and stratified cluster-based sampling to reduce redundancy while preserving stylistic diversity across languages.

3.1 Stylometric Feature Engineering

As shown in Figure 1, the core of our detection strategy relies on the extraction of a 17-dimensional stylometric feature vector. Unlike neural architectures that operate on sub-token embeddings, our feature extractor captures the "mechanical" habits and formatting idiosyncrasies that differentiate human developers from LLM generative priors.

The features are categorized into four main structural pillars:

1. **Scale and Vertical Geometry:** We measure the total character count (*n_chars*), the line count (*n_lines*) and the line-length statistics (*avg_line_len*, *max_line_len*). These serve as proxies for code compactness and the complexity of horizontal structures (e.g., extensive method chaining or long string literals).

2. **Documentation and Comment Signalling:** We implement a heuristic-based detection (independent of language-specific parsers) to calculate ratios for full-line comments, inline comments, and docstring-like delimiters (e.g. triple quotes). These features identify the "explanatory density" of the code, a key differentiator since LLMs often exhibit more standardized and frequent commenting patterns compared to human-written algorithmic solutions.

3. **Indentation and Formatting Consistency:** We analyze the leading whitespace of each line to detect indentation styles. This includes the mean and standard deviation of space-based indentation (*indent_mean*, *indent_std*), as well as the presence of leading tabs, leading spaces, or *mixed indentation* patterns. The latter is a particularly strong signal, as LLMs rarely produce inconsistent indentation (mixed tabs and spaces) within a single snippet, a common trait in multi-author or legacy human code.

4. **Lexical and Character Composition:** We compute the density of digits, alphabetic characters, and general whitespace (including tabs and newlines) relative to the total character count. These ratios (*digit_ratio*, *alpha_ratio*, *space_ratio*) capture the underlying "texture" of the code, such as the density of numeric constants versus descriptive identifiers.

By normalizing these 17 features by the total number of lines or characters, we create a scale-invariant representation that remains robust even when the model encounters programming languages not seen during the training phase.

3.2 Classification and Optimization

We used LightGBM (Ke et al., 2017) for binary classification, using its efficiency in modeling nonlinear relationships within tabular data. Hyperparameters were optimized via Bayesian Search over hundreds of trials, targeting the Macro F1-score. The search space focused on structural constraints (*num_leaves*, *min_child_samples*), regularization (*reg_alpha*, *reg_lambda*), and stochastic sampling (*subsample*, *colsample_bytree*). To mitigate language imbalance, we implemented a *class_weight* strategy, ensuring that the model prioritizes generalizable stylistic signatures over language-specific noise.

3.3 Experimental Baselines

To benchmark the effectiveness of our stylometric approach, we implemented two primary baselines.

Neural Transformer Baseline: We utilized the official baseline competition, consisting of a CodeBERT model (Feng et al., 2020) coupled with a Multi-Layer Perceptron (MLP) classifier.

OOD Detection (DeepSVDD): We evaluated the approach proposed by Zeng et al. (2025), which treats machine-generated code detection as an Out-of-Distribution task. This method maps CodeBERT embeddings into a latent space defined by a Deep Support Vector Data Description (DeepSVDD) hypersphere. Code snippets that fall outside this boundary are classified as human outliers. Although theoretically sophisticated, our analysis in Section 4 demonstrates that this approach suffers from sensitivity to extreme distribution shifts of the SemEval-2026 dataset.

4 Results

This section details the empirical performance of our stylometric approach. We structure our analysis to demonstrate (i) the superiority of our model over current baselines, (ii) the impact of Bayesian optimization, and (iii) the model's robustness in cross-language generalization.

4.1 Comparative Performance

The performance gap between standard neural architectures and our heuristic-driven approach is substantial. As summarized in Table 1, the official competition baseline struggles with the distribution shifts of Task 13, while our LightGBM model more than doubles the Macro F1-score in most scenarios.

Table 1: Performance across models under Leave-One-Out (LOO) evaluation. Training sizes (50k and 75k) were selected via stratified subsampling to probe the data-efficiency regime and assess whether performance saturates beyond a mid-scale setting. "Opt." indicates Bayesian Hyperparameter Optimization.

Model	Holdout	Size	Opt.	Macro F1
Baselines				
CodeBERT (MLP)	-	Full	No	0.300
DeepSVDD (OOD)	Python	25k	No	0.637
Stylometric-LGBM				
	Python	75k	Yes	**0.709**
		75k	No	0.689
		50k	No	0.689
		Full	No	0.702
	C++	75k	No	0.645
		50k	No	**0.648**
		75k	Yes	0.558
	Java	75k	No	**0.473**
		75k	Yes	0.449
		50k	No	0.444

4.2 Analysis of the "Optimization Effort"

The data reveals a clear "plateau of simplicity." Increasing the training size from 50k to 75k samples yielded negligible gains in the C++ and Python holdouts. This suggests that our 17-dimensional feature space saturates quickly, capturing the essential "style" of machine authorship even with reduced data volumes.

However, the application of **Bayesian Optimization** proved critical for the Python holdout, pushing the F1-score from 0.6886 to 0.7092. Conversely, the optimization phase appeared to specialize the model toward Pythonic structures, as seen in the C++ holdout performance drop (0.6454 to 0.5575). This highlights a fundamental trade-off in machine code detection: optimization for a dominant language may inadvertently reduce the universality of stylistic heuristics.

4.3 Generalization and Outlier Detection

The DeepSVDD approach, although sophisticated, achieved an F1-score of 0.6371. While this is sig-

nificantly better than the CodeBERT baseline, it still falls short of our non-optimized LightGBM (0.6886). This confirms our central hypothesis: high-level structural markers (indentation variance, whitespace density, and character composition) provide a more stable signal for machine-generation than high-dimensional neural embeddings, which are prone to overfitting to specific algorithmic domains.

5 Discussion

Our results show a clear inversion between neural and feature-engineered approaches under distribution shift. CodeBERT performs well under aligned train-test distributions, but its representations appear to capture language-specific syntactic patterns rather than generation-process-invariant signals. Under severe training skew, such as Python comprising more than 91% of the training data, this specialization limits cross-language generalization.

Bayesian optimization further reveals a trade-off between validation performance and robustness. Tuning for Python-holdout validation improves performance in that setting but degrades performance on other language holdouts. This indicates that even language-agnostic features can produce language-specialized decision boundaries when the validation objective is skewed.

Stylometric features mitigate this issue by targeting shallow but stable structural signals. Indentation variance, whitespace density, and character composition capture mechanical regularities of LLM-generated code rather than programming-language semantics. This helps explain why the model remains competitive with reduced training data: performance is driven more by invariant feature design than by dataset scale alone.

Results from DeepSVDD further support this conclusion. Although outlier detection can reduce the impact of label imbalance, high-dimensional neural embeddings remain sensitive to severe domain shifts. In this setting, explicit structural inductive biases are more robust than representation-heavy alternatives. For AI-generated code detection, lightweight models can therefore outperform semantic embeddings when the strongest signals are formatting regularity and stylistic consistency.

6 Conclusion

This work demonstrates that structural stylometric features combined with gradient-boosted decision trees provide a competitive and efficient baseline for machine-generated code detection under extreme distribution shift. By prioritizing cross-language structural invariants over high-dimensional semantic representations, the proposed approach improves robustness under skewed training distributions.

These findings suggest that strong generalization in AI-generated code detection does not necessarily require large transformer-based detectors. Instead, simple features with the right inductive bias can offer a transparent, reproducible, and computationally lightweight alternative. Future work should evaluate adversarial robustness, expand generator coverage, and reassess these features as LLM coding styles evolve.

7 Limitations

Our 17-dimensional feature space captures only surface-level formatting signals and does not encode deeper semantic or control-flow structure. The system was not evaluated under adversarial formatting perturbations, and deterministic features may be easier to circumvent than neural representations. Additionally, results reflect the generators and domains present in the shared-task dataset; evolving LLM stylistic patterns or broader language coverage may require revalidation.

Acknowledgments

This work has been fully funded by the project Research and Development of Gênese Digital: Scaling Interactive and Culturally Adapted Digital Humans supported by Advanced Knowledge Center in Immersive Technologies (AKCIT), with financial resources from the PPI IoT of the MCTI grant number 057/2023, signed with EMBRAPII. The authors are also grateful to the Fundação de Amparo à Pesquisa do Estado de Goiás (FAPEG) for the financial support provided for this research (Grant 64448878/2024).

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, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi,

Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*.

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, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. Semeval-2026 task 13: Detecting machine-generated code with multiple programming languages, generators, and application scenarios. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

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 31263–31289, Suzhou, China. Association for Computational Linguistics.

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 *Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. IEEE.

Qodo. 2025. State of AI code quality in 2025. https://www.qodo.ai/reports/state-of-ai-code-quality/. Accessed: 2026.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. 2024. AI models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759.

Cong Zeng, Shengkun Tang, Yuanzhou Chen, Zhiqiang Shen, Wenchao Yu, Xujiang Zhao, Haifeng Chen, Wei Cheng, and Zhiqiang Xu. 2025. Human texts are outliers: Detecting llm-generated texts via out-of-distribution detection. *Preprint*, arXiv:2510.08602.