YoungDSMLKZ at SemEval-2026 Task 13: MIL-UniXcoder with Meta-Stacking and Handcrafted Features for AI-Generated Code Detection

Yeraly Gainulla
Aqbobek Lyceum
eralygainulla@gmail.com

Agzam Shamsadinov
National School of Physics & Math
agzamshamsadinov@gmail.com

Abstract

We propose and validate a multi-view ensemble framework for 4-class AI-generated code detection (Human, AI, Hybrid, Adversarial) in realistic long-form repositories. Our system, Team YoungDSMLKZ, ranked **1st** out of 50+ teams in SemEval-2026 Task 13 Subtask C with a macro F1 of **0.7855** (+5.2 over runner-up). The framework combines: (i) a Dynamic Multiple Instance Learning (MIL) pipeline over UniXcoder chunks for $O(N)$-scalable long-context detection, (ii) transformer-based meta-stacking (UniXcoder and ModernBERT), and (iii) an XGBoost classifier on 200+ handcrafted stylometric features. Evidence localization analysis shows that 62.4% of decisive AI-detection signals reside beyond the standard 512-token window, validating the MIL design.

1 Introduction

The proliferation of advanced LLMs capable of writing functional code has introduced new challenges in software engineering, education, and security. Identifying whether a snippet of code is human-written, machine-generated, a hybrid of both, or intentionally obfuscated (adversarial) is an increasingly complex task. SemEval-2026 Task 13 (Orel et al., 2026b,a) provides a challenging benchmark for this exact problem across eight programming languages (Python, Java, JavaScript, C, C++, C#, PHP, and Go). Prior work on AI-generated code detection (Orel et al., 2025b,a) has established that multi-lingual and multi-generator settings pose substantial challenges for single-model approaches.

In this paper, we argue that AI-generated code detection in realistic repositories is fundamentally a *long-context* problem. Truncation makes standard detectors overconfident by processing only boilerplate (imports, class headers) while missing AI-generated logic further in the file. **Our key contribution** is a principled multi-view ensemble that partitions the detection task into three specialized diagnostic streams—stylometric, local-semantic, and global-evidence—fused via a class-source routing decision rule. Our code and extracted features are publicly available at `https://github.com/YoungDSMLKZ/YoungDSMLKZ-SemEval-13-c-solution`.

Related Work. Detecting machine-generated content has been studied for natural language (Wang et al., 2024; He et al., 2024) and code (Orel et al., 2025b,a). Recent benchmarks stringently emphasize robustness to obfuscation, paraphrasing, and low False Positive Rate (FPR) constraints (Yang et al., 2024). Authorship attribution methods (Uchendu et al., 2020) offer a complementary view by modelling statistical writing style. Multiple Instance Learning (MIL) (Ilse et al., 2018) has been applied to long-document classification in NLP, but to the best of our knowledge has not been used for AI-generated code detection across multi-kilotoken files. Our work closes this gap.

2 System Overview

Our final solution comprises a multi-stream architecture, the predictions of which are fused via a class-source routing decision rule. Figure 1 illustrates the overall pipeline.

2.1 Meta-Stacking Pipeline

The first module leverages a multi-stage meta-stacking architecture. We process the input code independently through fine-tuned UniXcoder (`YoungDSMLKZ/semeval-13c-unixcoder` on HuggingFace) and MazgaBERT (our fine-tuned ModernBERT (Answer.AI and LightOn, 2024), weights: `YoungDSMLKZ/MazgaBERT` on HuggingFace) pipelines. For each transformer, we extract class probabilities and train Level-1 CatBoost (Prokhorenkova et al., 2018) models—**CB-UniX** on UniXcoder probabilities and **CB-Maz** on MazgaBERT probabilities—using logarithmic proba-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 994–999

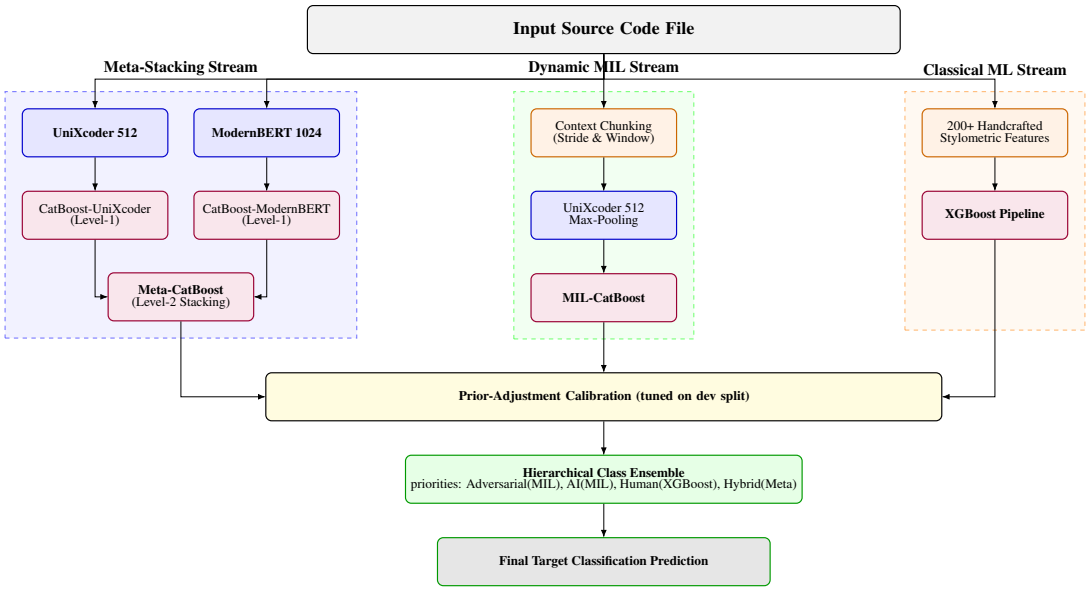

Figure 1: Overview of our multi-stream ensemble architecture.

bilities as baselines. To strictly prevent data leakage, we generate 5-fold stratified out-of-fold (OOF) predictions, where Level-1 models are trained exclusively on folds $k \neq i$ to infer on fold i. Finally, we concatenate the pristine OOF features and the blended logarithmic baselines to train a Level-2 Meta CatBoost model, which outputs the final stacked predictions.

2.2 Dynamic MIL Pipeline

Standard transformer-based models are constrained by a 512-token limit, leading to the "truncation trap" where models miss AI signals located deeper in a file. To address this efficiently, we implement a Dynamic MIL pipeline (Ilse et al., 2018). Rather than processing 4096 tokens through a computationally expensive long-context transformer ($O(N^2)$ attention), we decompose each file into N overlapping 512-token chunks with a stride of 256, giving $O(N)$ scaling relative to file length.

Each chunk is encoded independently by UniXcoder (model weights: `YoungDSMLKZ/UnixCoder-MIL` on HuggingFace), producing N hidden representations. We aggregate them via **max-pooling** followed by a CatBoost head. To validate this design, we conducted an aggregation ablation comparing max-pooling against a learned gated attention aggregator (Ilse et al., 2018) and a Dual-Stream aggregator (DSMIL) (Li et al., 2021) (Table 3). With identical linear classifier heads, attention-based aggregation slightly outperforms max-pooling (+1.68 F1), and DSMIL achieves the best linear performance (+2.70 F1). This suggests that learned weighting and dual-stream contextualization capture useful inter-chunk relationships. However, the dominant factor is the **downstream CatBoost head with class weights**, which provides a +4.25 F1 gain over the max-pooling linear baseline by handling class imbalance effectively. The aggregated logits are therefore passed to a MIL-CatBoost head, which yields a +10.9 F1 improvement over the XGBoost baseline, primarily by recovering errors on long-form code.

2.3 Classical ML Pipeline with XGBoost

Transformers can sometimes overlook deterministic, heuristic markers left by LLMs. To complement the deep learning models, we extract 200+ features spanning (i) formatting (blank/comment ratios, indentation consistency), (ii) lexical statistics (character entropy, average identifier length, coefficient of variation of line lengths), (iii) AST structure (node counts, tree depth), and (iv) regex markers of LLM-style patterns (e.g., conversational artifacts such as "Sure, here is the code"). We release the full feature list and extraction code in our repository. These features form the input to an XGBoost (Chen and Guestrin, 2016) classifier that predicts the class based entirely on structural and stylometric cues.

2.4 Final Ensemble: Class-Routing Decision Rule

Rather than averaging predictions, our fusion strategy applies a *class-source routing* rule. Each class

is assigned the subsystem that achieved highest per-class recall on dev-eval: Human and Hybrid are routed to the XGBoost and meta-stacking models respectively; AI and Adversarial are routed to the MIL-CatBoost pipeline. At inference time, we evaluate the designated models in priority order: Adversarial → AI → Human → Hybrid. The first model to predict its designated class determines the final label. If no priority conditions are met, we fall back to the base MIL-CatBoost prediction. This asymmetric prioritization ensures we explicitly maximize recall for the challenging Adversarial and AI classes before considering easier classes.

2.5 Language-Specific Feature Analysis

To study feature importance by language, we explored a language-routing LightGBM (Ke et al., 2017) baseline with TF-IDF + handcrafted features (macro-F1 0.7067 on dev-eval; **not part of the final system**). This allowed us to analyze per-language feature importance (Figure 2). Formatting traits such as `blank_ratio` and `comment_ratio` are universally important across languages, while `special_char_ratio` and `uppercase_ratio` peak for PHP and C#, confirming that tree-based models can implicitly partition the decision space by language.

3 Experimental Setup

Data. We used the official AICD Bench dataset provided by the task organizers (Orel et al., 2026a), covering eight programming languages: Python, Java, JavaScript, C, C++, C#, PHP, and Go. We train all models on the official training split (∼500K samples). Subtask C is characterized by long-form repository files (up to 2,400+ words), making simple truncation strategies highly suboptimal. For model selection and routing threshold tuning, we create a **stratified 80/20 split of the official validation set** (160K dev-train / 40K dev-eval). All offline metrics reported in this paper are on dev-eval unless stated otherwise.

Transformer Training. All transformer-based models were fine-tuned on NVIDIA H100 GPUs using AdamW optimizer with linear warmup and cosine decay, mixed-precision (bf16) training, MultiClass cross-entropy loss, and fixed random seed 42. Early stopping was based on dev macro-F1. UniXcoder was fine-tuned for 36K steps with a maximum sequence length of 512 tokens and a batch size of 128. The MIL-UniXcoder variant operates on a maximum input length of 4096 tokens, chunked into 512-token windows with a stride of 256 and up to 16 chunks per sample, with a batch size of 64. MazgaBERT (our fine-tuned ModernBERT-base (Answer.AI and LightOn, 2024)) was fine-tuned with a maximum length of 1024 tokens and a batch size of 64.

Gradient Boosting. The meta-stacking CatBoost models were trained with 5-fold stratified cross-validation using 2500 iterations, learning rate 0.05, depth 4, and L2 regularization of 3.0 with GPU acceleration. The MIL-CatBoost head used 500 iterations, depth 6, $\ell_2 = 5$, and class weights $[1.0, 1.0, 22.0, 1.0]$ to boost minority-class recall. The XGBoost classifier was trained with 1000 estimators, learning rate 0.05, max depth 10, subsample 0.8, and early stopping after 100 rounds of no improvement.

Evaluation. The official metric for this task is macro-averaged F1-score across the four classes.

4 Results and Analysis

4.1 Leaderboard Results

Our system ranked **1st** on the official Subtask C leaderboard. Table 1 shows the top-5 teams.

Rank	Team	Macro F1
1	**YoungDSMLKZ (ours)**	**0.7855**
2	TeleAI	0.7333
3	Yuvan Ramesh	0.7140
4	Brazil AI	0.6977
5	mcdok	0.6864

Table 1: Top-5 teams on the official SemEval-2026 Task 13 Subtask C leaderboard.

4.2 Subsystem Contributions

Table 2 and Figure 3 present the **official submission scores** (Kaggle leaderboard) of each individual subsystem and the final ensemble.

Model / Configuration	Test F1 (Macro)
Lang-Routing LightGBM	0.4930
Classical ML (XGBoost only)	0.5243
Dynamic MIL-MaxPool + CatBoost	0.6331
Meta-Stacking only	0.6995
Full Ensemble	**0.7855**

Table 2: Subsystem ablation on the official test leaderboard.

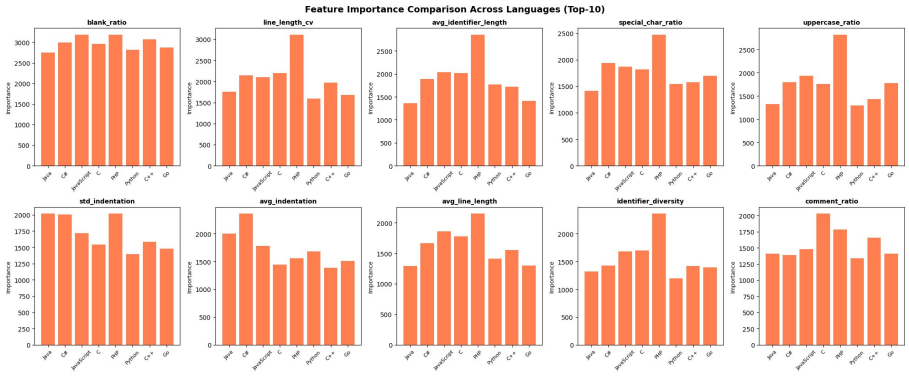

Figure 2: Feature importance across languages (Top-10 features). Formatting traits like `blank_ratio` are universally discriminative, while others vary by language family.

Aggregation	Head	Test F1
Max-Pooling	Linear	0.5906
Attention (Ilse et al.)	Linear	0.6074
Dual-Stream (DSMIL)	Linear	0.6176
Max-Pooling	CatBoost	0.6092
Max-Pooling	CatBoost + weights	**0.6331**

Table 3: MIL aggregation ablation. With an identical linear head, attention outperforms max-pooling (+1.68 F1), while Dual-Stream MIL yields the highest linear baseline (+2.70 F1 over max-pooling). However, Max-Pooling with CatBoost and class weights dominates all linear configurations.

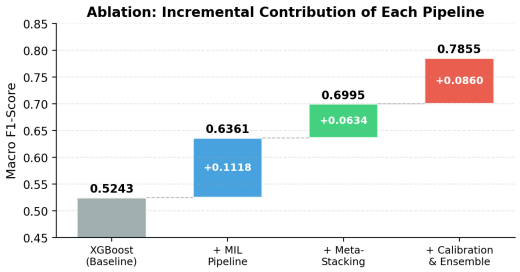

Figure 3: Incremental contribution of each pipeline component. Each block shows the gain over the previous best, building from the XGBoost baseline (0.5243) to the final ensemble (0.7855).

Class	Prec.	Rec.	F1
Human	0.979	0.984	0.982
AI	0.871	0.878	0.874
Hybrid	0.840	0.856	0.848
Adversarial	0.878	0.834	0.855
Macro Avg	**0.892**	**0.888**	**0.890**

Table 4: Per-class metrics (validation).

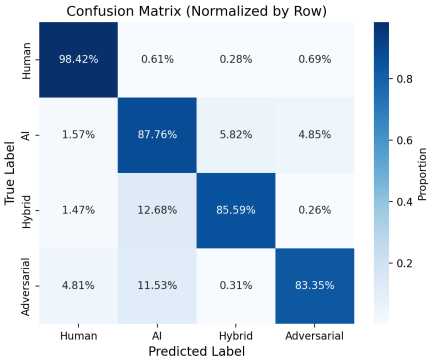

Figure 4: Normalized confusion matrix on validation set.

4.3 Per-Class Error Analysis

Table 4 reports per-class precision, recall, and F1 on the validation set. Figure 4 shows the corresponding confusion matrix.

The *Human* class is the easiest to detect (F1=0.982), likely due to high stylistic consistency. The hardest classes are *Hybrid* (F1=0.848) and *Adversarial* (F1=0.855). Adversarial samples are primarily confused with Human code. Manual inspection of misclassified adversarial samples reveals a recurring pattern: obfuscation tools (e.g., variable renaming with random suffixes, dead code injection) disrupt the **global semantic structure** that transformer-based encoders rely on, causing confident misclassification. However, these tools preserve the statistical distribution of whitespace, comment ratios, and character entropy—the exact features captured by our XGBoost stylometric stream—explaining why the multi-view ensemble recovers many cases the individual transformer streams miss. The Hybrid-AI confusion stems from contiguous AI-generated blocks that are locally indistinguishable from pure-AI code at the chunk level.

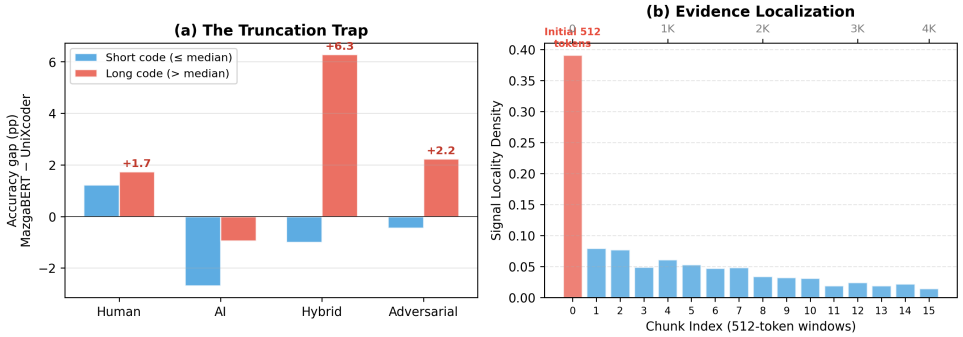

Figure 5: Long-context analysis: (a) Gap between 1024-token and 512-token models by length; (b) AI-signal locality density.

4.4 Long-Context Analysis

Our core hypothesis is that truncation causes confident misclassification on long code: a 512-token model may label a file as *Human* because the imports look standard, even when AI-generated logic resides deeper. Figure 5 summarizes our analysis of this effect.

First, we compare the per-class accuracy gap between MazgaBERT (1024 tokens) and UniXcoder (512 tokens). As shown in Figure 5a, the advantage is negligible on short code but grows dramatically on long files: **+6.3pp** for Hybrid and **+2.2pp** for Adversarial. Second, Figure 5b shows that in 62.4% of correctly identified long-context AI samples, the decisive max-pooled chunk was located beyond the initial 512 tokens, directly validating the MIL design. A controlled ablation of max- vs. mean-pooling strategies would require full model retraining and remains future work; however, the noisy-OR rationale (Section 2) and the evidence localization result together constitute strong indirect support.

5 Limitations and Broader Impacts

While effective on Subtask C, stylometric features remain vulnerable to formatting normalization and simple countermeasures (e.g., automated identifier renaming). Furthermore, real-world deployment requires strict False Positive Rate (FPR) controls (e.g., TPR@1% FPR (Yang et al., 2024)) to prevent false accusations against developers, a societal consideration absent from purely metric-driven optimizations. Future research should evaluate this architecture on external out-of-distribution (OOD) splits such as AICD Bench Task 3 or cross-model subsets, replace the heuristic class-routing rule with learned gating, and conduct window-occlusion causal tests to definitively isolate long-context evidence drivers.

6 Conclusion

We presented the YoungDSMLKZ system for SemEval-2026 Task 13 Subtask C, which ranked 1st with a macro F1 of 0.7855. Our multi-level ensemble demonstrates that complementary modeling paradigms yield substantial gains over any single approach (+8.6 F1). We highlight three generalizable lessons: (1) long-context evidence aggregation (via MIL chunk-and-pool) is essential for repository-scale code, where truncation causes confident but incorrect predictions; (2) stylometric and structural features capture stable signals across languages and generators, complementing neural detectors under distributional drift; and (3) class-source routing—assigning each class to its most accurate subsystem—effectively reduces the asymmetric cost of missing rare positives. Error analysis shows that Hybrid and Adversarial classes remain the most challenging, suggesting that future work should focus on better modeling partial and obfuscated code generation.

Acknowledgments

We thank the SemEval-2026 Task 13 organizers for providing the AICD Bench dataset and the evaluation framework.

References

Answer.AI and LightOn. 2024. Modernbert. https://huggingface.co/answerdotai/ModernBERT-base.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, pages 785–794.

Xinlei He, Xinyue Shen, Zeyuan Chen, Michael Backes, and Yang Zhang. 2024. MGTBench: Benchmarking machine-generated text detection. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*.

Maximilian Ilse, Jakub M. Tomczak, and Max Welling. 2018. Attention-based deep multiple instance learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2127–2136.

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*, volume 30.

Bin Li, Yin Li, and Kevin W Eliceiri. 2021. Dual-stream multiple instance learning network for whole slide image classification with self-supervised contrastive learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14318–14328.

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, Vienna, Austria. Association for Computational Linguistics.

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)*, 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)*, 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.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, volume 31.

Adaku Uchendu, Thai Le, Kai Shu, and Dongwon Lee. 2020. Authorship attribution for neural text generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 8384–8395.

Yuxia Wang, Jonibek Manber, Artem Shelmanov, Ekaterina Artemova, Giovanni Puccetti, Dmitry Ustalov, Jinyan Guo, Ryuto Koike, Naula Khusainova, and Preslav Nakov. 2024. M4: Multi-generator, multi-domain, and multi-lingual black-box machine-generated text detection. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1369–1380.

Ziqian Yang and 1 others. 2024. Codemirage: A multilingual benchmark for detecting ai-generated and paraphrased source code from production-level llms. *arXiv preprint*.