ICI Innolabs at SemEval-2026 Task 13: Sliding Windows Meet Code Transformers

Sebastian Balmuș **Bogdan Dura**
National Institute for Research & Development in Informatics – ICI Bucharest
{sebastian.balmus, bogdan.dura}@ici.ro

Abstract

We describe our system for SemEval-2026 Task 13, Subtask B, which focuses on multi-class authorship attribution for code: given a code snippet, the goal is to predict whether it is human-written or generated by one of ten LLM families. The task presents two central challenges: severe class imbalance and long input sequences that frequently exceed the context length of encoder-based Transformers. To address these issues, we adopt a window-based fine-tuning and inference framework. During training, we randomly sample 512-token windows from each snippet and optimize a class-weighted cross-entropy objective with label smoothing. At inference time, we apply a sliding-window strategy and aggregate window-level logits to obtain a snippet-level prediction. We fine-tune three pretrained code encoders (CodeBERT, UniXcoder, and StarEncoder) under this framework and combine their outputs via majority voting. On the official validation split, our best single model (StarEncoder) achieves 0.60 macro F1. On the final test set, the three-model ensemble reaches 0.41 macro F1, ranking 10th on the leaderboard. Our results demonstrate that window-based modeling combined with imbalance-aware optimization provides a robust and reproducible baseline for multi-class LLM attribution under distribution shift.

1 Introduction

Recent progress in large language models (LLMs) has made code generation widely accessible, enabling high-quality synthesis across programming languages and domains. At the same time, this capability introduces new challenges for software forensics, academic integrity, and security auditing: machine-generated code can be difficult to distinguish from human-written code, especially under distribution shifts in language, domain, or generator family.

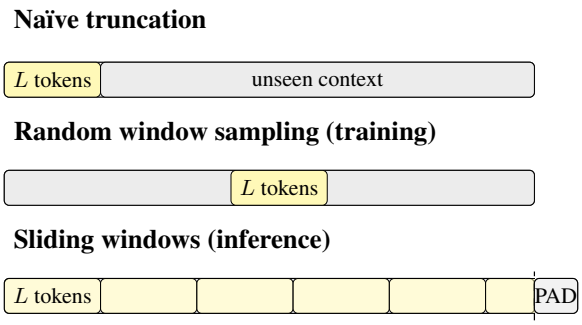

Figure 1: Motivation for our window-based approach. Naïve truncation keeps only the first L tokens, discarding unseen context. During training, we sample a random L-token window from long snippets. At inference time, we process the full snippet using fixed-length windows and pad the final window to length L.

SemEval-2026 Task 13 (Orel et al., 2026) addresses this problem by benchmarking systems for detecting and attributing machine-generated code under diverse and realistic conditions. Subtask B focuses on multi-class authorship attribution, requiring systems to determine whether a code snippet is human-written or generated by one of ten LLM families. Evaluation includes both seen-author conditions (generators observed during training) and unseen-author conditions (previously unseen generators from known families), and performance is measured using macro F1 to emphasize balanced behavior across classes. The training set contains 500K samples and exhibits substantial class imbalance, with human-written code forming the majority and individual LLM families represented by far fewer examples. Additionally, code snippets may exceed the maximum context length of standard encoder-based Transformers. These characteristics make long-context modeling, robustness under distribution shift, and minority-class calibration central challenges. Addressing these challenges requires balancing computational constraints with effective coverage of long inputs and

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3274–3279

stability under extreme class imbalance.

To address these issues, we propose a structured window-based fine-tuning framework combined with imbalance-aware optimization. During training, we randomly sample fixed-length token windows from long sequences to avoid truncation while maintaining bounded compute. We optimize a class-weighted cross-entropy objective with label smoothing to stabilize minority-class learning. At inference time, we apply a sliding-window strategy with logit aggregation to incorporate evidence from the entire snippet. We fine-tune three complementary pretrained code encoders under this framework and combine their predictions via ensembling to improve robustness under distribution shifts.

2 Related Work

Machine-generated content detection. Detection of synthetic content has been widely studied in natural language, including shared tasks such as SemEval-2024 Task 8 (Wang et al., 2024), which evaluated systems on machine-generated text detection and source attribution across domains and generators. Results from such benchmarks highlight the difficulty of generalizing across distribution shifts and the importance of robust modeling strategies. Prior work on LLM-generated text detection has shown that detectors often degrade under distribution or generator shifts due to stylistic overlap between human and machine outputs, emphasizing the challenge of generalized attribution (Wu et al., 2025; Przystalski et al., 2026). For code, recent work has proposed semantic feature–based models to distinguish human and generated code using pretrained models and statistical metrics (Pham et al., 2024).

Pretrained code representation models. Embedding-based and feature-driven models using pretrained code embeddings have both proven effective in distinguishing human vs model-generated code structures (Nirob et al., 2026). Transformer-based encoders pretrained on source code have demonstrated strong transfer performance on downstream classification tasks. CodeBERT (Feng et al., 2020) and UniXcoder (Guo et al., 2022) learn joint representations of natural language and programming languages, enabling effective fine-tuning for understanding tasks. Encoder-only models trained on large-scale code corpora, such as StarEncoder (Li et al., 2023), further provide lightweight yet expressive representations suitable for classification. Our approach builds on these pretrained encoders and adapts them to multi-class LLM-family attribution.

LLM authorship attribution for code. Recent studies show that LLM-generated code retains stylistic and structural fingerprints that allow attribution to specific models or model families. Bisztray et al. (2026) introduce a benchmark for LLM authorship identification in C programs and demonstrate that encoder-based classifiers can achieve high attribution accuracy, even among closely related generators.

3 System Overview

We approach Subtask B as a supervised multi-class classification problem over code snippets. Our design is driven by two central challenges: (i) many inputs exceed the maximum context length of encoder-based Transformers, and (ii) the class distribution is highly imbalanced across human and LLM-family labels. To address these issues, we propose a window-based training and inference framework with imbalance-aware optimization. An overview of the training and inference procedures is illustrated in Figures 2 and 3.

Our pipeline is applied uniformly to three pretrained code encoders: CodeBERT, UniXcoder, and StarEncoder. Each model is fine-tuned independently under the same protocol, and their outputs are combined through ensembling at test time.

3.1 Window-Based Training with Imbalance-Aware Optimization

Let a tokenized code sequence have length T, which may exceed the model's maximum input length L. Rather than truncating to a fixed prefix, we randomly sample a contiguous window of length L during each training epoch (padding when $T < L$), as shown in Figure 2. The sampled window is encoded by the pretrained backbone, followed by a linear classification layer.

Random window sampling allows the model to observe different regions of long files across epochs while maintaining bounded memory and computation. This strategy reduces reliance on arbitrary truncation and improves coverage of long-context information.

To mitigate severe class imbalance, we optimize a class-weighted cross-entropy loss, assigning larger weights to underrepresented classes. In addition, we apply label smoothing to reduce overcon-

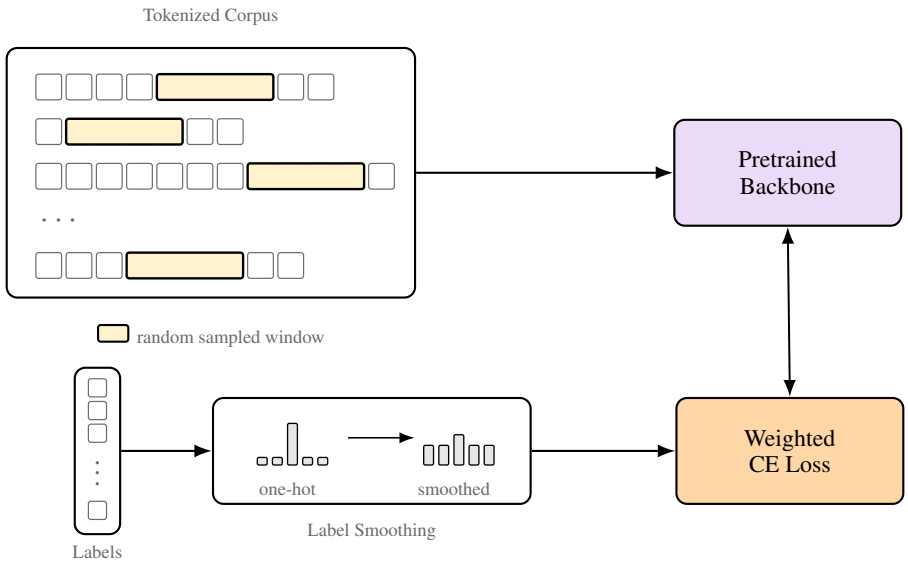

Figure 2: **Window-based training with class-balanced optimization.** Variable-length token sequences form a tokenized corpus; a fixed-length window is sampled (yellow) and fed into a pretrained backbone. Labels are smoothed before being used in the balanced loss; the loss provides the training signal for updating the backbone.

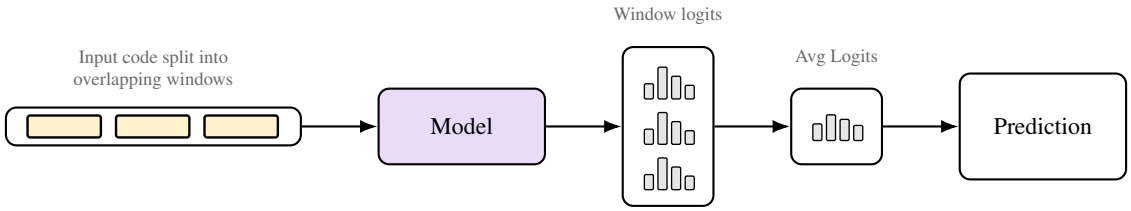

Figure 3: **Sliding-window inference with logit aggregation.** The input sequence is divided into overlapping fixed-length windows, each encoded by the pretrained backbone. Window-level logits are averaged to obtain aggregated logits, from which the final prediction is computed.

fidence and improve stability for minority labels. Together, these techniques improve robustness and macro-F1 performance on the development set.

3.2 Sliding-Window Inference and Logit Aggregation

At inference time, we ensure full coverage of each snippet using a deterministic sliding-window strategy (Figure 3). The token sequence is divided into overlapping windows of size L with stride $S < L$. Each window is independently encoded, producing window-level logits.

We aggregate evidence across the entire snippet by averaging logits over all windows and computing the final prediction via $\arg\max$ over the aggregated scores. This approach enables the model to integrate information from the complete code sample rather than relying on a single segment.

3.3 Model Ensembling

We fine-tune CodeBERT, UniXcoder, and StarEncoder independently using the same window-based training and inference framework. For each model, we obtain a single snippet-level prediction by applying sliding-window inference and averaging window-level logits, followed by an arg max decision. To produce the final system output, we combine the three snippet-level predictions using majority voting.

4 Experimental Setup

4.1 Data and Evaluation Protocol

We train and validate our models using the official Subtask B training and validation splits. The task consists of 11 classes (human-written code and 10 LLM families) and is evaluated using macro F1, which we use for model selection.

The dataset exhibits severe class imbalance. Human-written code accounts for 88.4% of the

training data, while individual LLM families range between 0.39% and 2.16%. The imbalance ratio between the largest and smallest classes exceeds 200×. This distribution motivates the use of class-weighted optimization and label smoothing to improve minority-class stability.

Code snippets frequently exceed the standard 512-token context limit of encoder-based models. While the median length is 361 tokens, the 75th percentile already reaches 752 tokens, and the 90th percentile exceeds 1,400 tokens. The distribution has a pronounced long tail, with the 99th percentile around 3,800 tokens and maximum lengths above 14,000 tokens. These statistics motivate our window-based modeling strategy.

4.2 Tokenization and Windowing

To handle long sequences, we operate on fixed-length windows of $L = 512$ tokens. Each snippet is first tokenized without truncation. Windows are then constructed at the token level before adding model-specific special tokens.

During training, we apply random window sampling: for each example and epoch, a contiguous span of length L is randomly selected from the full tokenized sequence (or the full sequence is used if shorter). This ensures that different parts of long files are observed across epochs.

For validation during training, we use a deterministic first-window strategy to ensure stable model selection and early stopping.

4.3 Training Configuration

Each backbone is fine-tuned with a linear classification head for up to 10 epochs. We use a learning rate of 2×10^{-5}, weight decay of 0.01, linear learning-rate scheduling with a warmup ratio of 0.05, gradient clipping with maximum norm 1.0, and mixed-precision training. The batch size is 32, and early stopping with patience 5 is applied based on validation macro F1. The best checkpoint on the validation set is retained.

To address extreme imbalance, we employ a class-weighted cross-entropy objective combined with label smoothing ($\epsilon = 0.05$). Class weights are computed from the training distribution using an inverse square-root frequency scheme and normalized to maintain stable optimization.

4.4 Inference and Logit Aggregation

At inference time, we apply a sliding-window strategy to ensure full coverage of each snippet. Inputs are split into overlapping windows of length $L = 512$ with stride $S = 256$, corresponding to 50% overlap. To control computational cost, the number of windows per sample is capped at 32.

Each window is encoded independently to produce window-level logits. We aggregate evidence across the entire snippet by averaging logits over all windows and compute the final prediction via $\arg\max$ over the aggregated scores. Windows are processed in micro-batches of 64 for efficiency.

5 Results

We evaluate our systems using macro F1, the official shared-task metric. Table 1 reports performance on the validation split and the final test set.

Model	Validation F1	Test F1
Official Baseline	0.33	0.22
CodeBERT	0.56	0.39
UniXcoder	0.59	0.38
StarEncoder	**0.60**	0.40
CodeBERT + UniXcoder	–	0.37
CodeBERT + StarEncoder	–	0.39
UniXcoder + StarEncoder	–	0.38
All Three (Ensemble)	–	**0.41**

Table 1: Macro F1 on the official validation split and final test set. The ensemble combines predictions from CodeBERT, UniXcoder, and StarEncoder using majority voting.

Overall performance. All three fine-tuned backbones substantially outperform the official baseline. The baseline achieves 0.22 macro F1 on the test set, whereas our best single model (StarEncoder) reaches 0.40 macro F1, corresponding to an absolute improvement of 18 points. These results suggest that window-based modeling combined with imbalance-aware optimization provides a strong baseline for multi-class LLM attribution.

Among single models, StarEncoder achieves the strongest validation performance (0.60 macro F1) and the highest single-model test score (0.40 macro F1). CodeBERT and UniXcoder follow closely, indicating that modern pretrained code encoders provide competitive representations for authorship attribution when adapted to long and imbalanced inputs.

Validation–test gap and distribution shift. We observe a substantial drop from validation to test performance across all models (approximately 0.20 macro F1). The consistency of this degradation across three distinct backbones suggests that the primary challenge is not model-specific overfitting,

but a systematic distribution shift between the two splits.

Given the extreme class imbalance (88.4% human-written code), even modest degradations on minority LLM families can disproportionately affect macro F1, which equally weights all classes. Under unseen-author and cross-domain conditions, stylistic cues learned during training may shift, leading to increased confusion for low-frequency generators.

The similar magnitude of the validation–test drop across CodeBERT, UniXcoder, and StarEncoder further indicates that robustness under distribution shift remains a structural challenge of the task rather than a limitation of a specific architecture.

Model complementarity and ensembling. To evaluate complementarity across encoders, we construct majority-vote ensembles. Pairwise combinations do not consistently surpass the strongest individual model, suggesting partial redundancy in learned representations. However, combining all three backbones yields the best overall test performance (0.41 macro F1), providing a modest but consistent improvement over the best single model. Although all ensemble members are encoder-based architectures, they differ in pretraining objectives and code corpora, which introduces moderate representational diversity. The modest gains from ensembling suggest partial complementarity despite architectural similarity. Under distribution shift, such diversity appears beneficial even when individual models already perform strongly.

Impact of class imbalance. Despite these improvements, performance remains constrained by the extreme imbalance of the dataset. Human-written code accounts for more than 88% of the training data, while several LLM families represent less than 1%. Because macro F1 equally weights all classes, errors on minority LLM families significantly affect the overall score. The improvements over the baseline nevertheless suggest that class-weighted optimization and label smoothing contribute positively to stabilizing minority-class predictions.

Overall, our results demonstrate that combining window-based modeling with imbalance-aware fine-tuning and multi-model ensembling provides a robust approach for long-context, multi-class code attribution under challenging distribution shifts.

6 Conclusions

In this paper, we presented our approach to Subtask B of SemEval-2026 Task 13 on multi-class attribution of machine-generated code. The task poses two central challenges: severe class imbalance and long input sequences that frequently exceed the standard context length of encoder-based models. To address these issues, we proposed a window-based fine-tuning and inference framework combined with imbalance-aware optimization. Random window sampling during training increases coverage of long files while maintaining bounded computation, and sliding-window inference aggregates evidence across the full snippet. In addition, class-weighted loss and label smoothing improve stability for minority LLM families. Compared with naïve truncation to the first 512 tokens, our window-based strategy provides broader coverage of long files and reduces sensitivity to missing contextual regions. This effect is particularly important for long-tail samples whose discriminative cues may appear outside the initial context window. Our experiments demonstrate that this combination substantially outperforms the official baseline and yields robust performance under distribution shift. While a notable validation–test gap highlights the difficulty of generalization to unseen conditions, ensembling diverse pretrained code encoders provides consistent improvements. Our best single model achieved 0.40 macro F1 on the official test set, while the three-model ensemble reached 0.41 macro F1 and ranked 10th overall. Our findings highlight that robust multi-class LLM attribution for code remains challenging under distribution shift, even with strong pretrained encoders. Future work should explore structure-aware representations, improved calibration under extreme imbalance, and training strategies explicitly designed for unseen-author generalization.

Limitations

Our approach relies on fixed-length encoder models and therefore processes long code snippets using independent windows. Although sliding-window aggregation improves coverage, it does not explicitly model long-range dependencies across distant parts of a file. We focused on class-weighted cross-entropy with label smoothing due to its stability under severe imbalance. Alternative objectives such as focal loss or contrastive learning remain promising directions for future work. Finally,

the noticeable validation–test performance gap indicates sensitivity to distribution shift. Additionally, our approach does not explicitly disentangle stylistic cues from content-level signals, which may limit generalization under domain shifts. Finally, majority-vote ensembling improves robustness but does not address calibration or minority-class uncertainty in a principled manner.

References

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*, AISec '25, pages 28–39. Association for Computing Machinery.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and 1 others. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the association for computational linguistics: EMNLP 2020*, pages 1536–1547.

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.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, and 48 others. 2023. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Syed Mehedi Hasan Nirob, Shamim Ehsan, Moqsadur Rahman, and Summit Haque. 2026. Whitespaces don't lie: Feature-driven and embedding-based approaches for detecting machine-generated code. *Preprint*, arXiv:2601.19264.

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.

Hung Pham, Huyen Ha, Van Tong, Dung Hoang, Duc Tran, and Tuyen Le. 2024. Magecode: Machine-generated code detection method using large language models. *IEEE Access*, PP:1–1.

Karol Przystalski, Jan K. Argasiński, Iwona Grabska-Gradzińska, and Jeremi K. Ochab. 2026. Stylometry recognizes human and llm-generated texts in short samples. *Expert Systems with Applications*, 296:129001.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, and Thomas Arnold. 2024. Semeval-2024 task 8: Multidomain, multimodel and multilingual machine-generated text detection. In *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 2057–2079.

Junchao Wu, Shu Yang, Runzhe Zhan, Yulin Yuan, Lidia Sam Chao, and Derek Fai Wong. 2025. A survey on llm-generated text detection: Necessity, methods, and future directions. *Computational Linguistics*, 51(1):275–338.