CodeDet-NITS at SemEval-2026 Task 13: AI Code Authorship Detection Beyond Truncation

Sai Teja Lekkala[1], Annepaka Yadagiri[1], Kshitij Patiyal[1], Sangam Sai Anish[1], Partha Pakray[1]
[1]National Institute of Technology Silchar, Assam, India
{lekkalad_ug_22, annepaka22_rs, partha}@cse.nits.ac.in

Abstract

Automatically determining whether source code is human written or produced by a specific family of large language models is becoming essential for reliable assessment, provenance tracking, and dataset curation. We present a lightweight yet competitive system for SemEval 2026 Task 13 Subtask B, which requires attributing each snippet to one of eleven classes: human or one of ten LLM families. Our method repurposes code oriented instruction tuned backbones from the Qwen2.5 Coder series as sequence classifiers and adapts them using QLoRA, combining frozen low precision weights with low rank trainable adapters to reduce memory and compute overhead. The core design choice addresses long snippets without losing evidence. Instead of truncating to a fixed context, we apply an overlapping sliding window strategy that expands long examples into multiple fixed length windows during training, all sharing the same label. For validation and test, windows are generated on the fly and their evidence is aggregated by averaging logits to yield a single prediction per snippet, enabling token complete use of the input while keeping inference stable. Our final submission ranked **8th** on the official Subtask B test set leaderboard.

1 Introduction

Detecting machine generated code has moved from a curiosity to a practical requirement. Modern code generators can produce fluent solutions across many languages, which complicates academic integrity checks, hiring assessments, dataset curation, and incident response where provenance matters. Subtask B of SemEval 2026 Task 13 (Orel et al., 2026) frames this challenge as multi class authorship detection: given a code snippet, the system must decide whether it is human written or produced by one of ten model families, and it is evaluated using macro F1 to emphasize balanced performance under strong label imbalance. Two aspects make the setting demanding. First, the classes correspond to generator families rather than a single model, which encourages features that generalize across related generators. Second, code snippets can be long. A straightforward classifier that truncates to a small context risks discarding critical cues that may appear anywhere in the snippet, such as library usage patterns, error handling style, naming conventions, or comment formatting.

Our system follows a pragmatic philosophy: begin with a strong code pretrained backbone, keep the adaptation lightweight, and handle long inputs without throwing away evidence. We choose Qwen2.5 Coder models (Hui et al., 2024) as backbones because they are trained specifically for code understanding and generation. We convert the problem into standard sequence classification by prepending a short language cue and attaching a classification head. To make training feasible on modest hardware, we apply QLoRA (Dettmers et al., 2023), which builds on LoRA (Hu et al., 2022) by backpropagating through a frozen quantized model and learning only low rank adapters and a small set of task parameters. The novelty in our pipeline is not a new architecture but a robust long context strategy tailored to authorship detection: train time window expansion combined with streaming window inference and logit pooling. This design keeps the context budget fixed while still considering the entire snippet.

2 Related Work

Code authorship attribution. Authorship attribution for source code has traditionally been studied through code stylometry, where consistent choices in formatting, lexical preferences, and structural patterns act as signatures. Even before modern LLMs, work in this area showed that such signals can persist across projects and withstand basic obfuscation, making attribution feasible in re-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3165–3168
July 3-4, 2026 ©2026 Association for Computational Linguistics

alistic settings (Caliskan-Islam et al., 2015). More recent neural approaches use pretrained code representations such as CodeBERT to replace manual feature design with learned features for downstream classification (Feng et al., 2020). Our work follows this supervised classification direction, but focuses on distinguishing human code from multiple LLM-family sources in a shared-task setting.

Machine-generated content and code detection. The broader literature on machine-generated content detection has explored supervised detectors and model-based scoring methods, with increasing attention to robustness under distribution shift (Mitchell et al., 2023). Compared with binary generated-text detection, AI code authorship detection is more fine-grained because the model must distinguish among multiple generator families whose outputs can share similar style, formatting, and library-use patterns.

Efficient adaptation and long-context handling. Parameter-efficient fine-tuning has become a practical way to adapt large transformer models. LoRA enables low-rank updates, while QLoRA extends this idea to quantized backbones to reduce memory requirements (Hu et al., 2022; Dettmers et al., 2023). Long inputs are commonly handled either by modifying the attention mechanism or by chunking the input and aggregating predictions; the latter is attractive because it works with off-the-shelf pretrained models (Beltagy et al., 2020). Our system follows this chunking-based direction by combining sliding-window training with streaming inference and logit pooling.

3 Methodology

3.1 Dataset Description

We use the official Subtask B data release, which provides a training split of 500,000 samples and a validation split of 100,000 samples and the label count in each splits are shown in Figure 1, and Figure 2. Each sample includes the program text, the programming language, and a label ID where 0 denotes human and 1 through 10 correspond to the ten LLM families. The class distribution is highly skewed toward human code, with hundreds of thousands of human examples and a few thousand per generator family, which motivates macro averaged evaluation and careful monitoring of minority class behavior. The test split used for submission contains an id field and requires predictions in a two

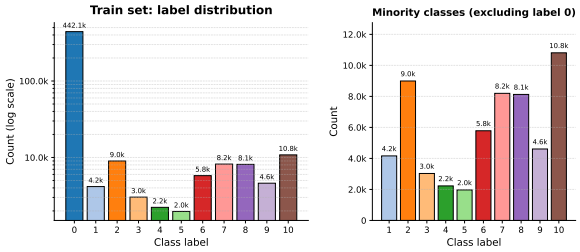

Figure 1: Label distribution in the training set, shown on a logarithmic scale (left) and for minority classes excluding label 0 on a linear scale (right).

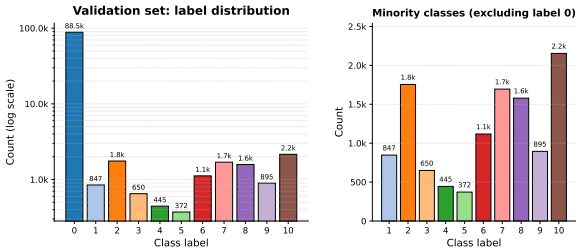

Figure 2: Label distribution in the validation set, shown on a logarithmic scale (left) and for minority classes excluding label 0 on a linear scale (right).

column CSV file with id and label.

3.2 Experimental Setup

Input representation. For each example we construct a single text sequence

$$x = \text{“Language: } \ell\text{”} \parallel \text{“\textbackslash n\textbackslash n”} \parallel c, \qquad (1)$$

where ℓ is the language string and c is the code snippet. This gives the model a minimal cue about syntax and idioms without hand engineered features.

Backbone and fine tuning. We use Qwen2.5 Coder instruction tuned checkpoints as encoder style classifiers by loading them with a sequence classification head. Training uses QLoRA, with four bit quantized weights and NF4 style quantization, and low rank adapters injected into quantized linear layers. Only adapter parameters and the classification head are updated. Optimization uses a paged AdamW variant supported by bitsandbytes, a constant with warmup learning rate schedule, and mixed precision where available. Implementation relies on the Transformers library and Accelerate for device and distributed handling. The entire pipeline architecture is shown in the Figure 3.

Token complete long context via windows. Let the tokenizer produce a token sequence of length

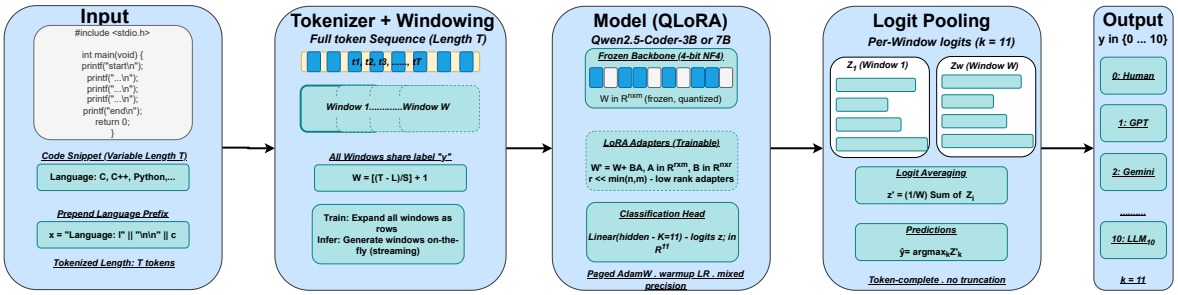

Figure 3: QLoRA-based code classification pipeline: from tokenization and windowing to pooled logits and final model prediction.

T. We fix a window length L and an overlap controlled by stride S with $S < L$. We generate windows starting at positions $0, S, 2S, \ldots$ until the end of the sequence is covered. Each window is independently classified, producing logits $z_i \in \mathbb{R}^K$ for $K = 11$ classes. At inference time we aggregate by logit averaging:

$$\bar{z} = \frac{1}{W} \sum_{i=1}^{W} z_i, \qquad \hat{y} = \arg \max_{k \in \{1,\ldots,K\}} \bar{z}_k, \quad (2)$$

where W is the number of windows for the snippet. This pooling uses all windows equally and empirically avoids instability that can occur when a single window dominates. The key practical detail is that validation and test windows are generated on the fly, so extremely long snippets do not need to be materialized as thousands of dataset rows.

Evaluation metric. The official metric is macro F1. For class k, precision and recall are computed from the confusion counts, and

$$F_{1,k} = \frac{2P_k R_k}{P_k + R_k}, \qquad F_{1,\text{macro}} = \frac{1}{K} \sum_{k=1}^{K} F_{1,k}. \quad (3)$$

We select the best checkpoint by validation macro F1, aligning model selection with the leaderboard target.

4 Results and Discussion

4.1 Official Results

We train two main model sizes, Qwen2.5-Coder 3B and 7B, keeping the token budget fixed at $L = 512$ and using a stride of 128 tokens, corresponding to an overlap of 384 tokens between adjacent windows. Training uses the official training split and evaluates on the validation split after each epoch. The submission file is produced by running streaming window inference on the test split and writing the predicted label IDs alongside sample ids.

Table 1 summarizes the official Subtask B leaderboard for the submission track. Our system, submitted under the team name *CodeDet-NITS*, ranks **8th** with an official macro F1-score of **0.41570**. The gap between the top entries and the middle of the table suggests that multi-class family attribution remains challenging, especially under severe class imbalance and overlapping stylistic signals across model families. In our own development runs, we observed that improvements on the validation split typically came from reducing errors on minority generator classes rather than further increasing the already high accuracy driven by the dominant human class. For this reason, we treated macro F1 as the primary selection criterion throughout, and we recommend reporting per-class F1 alongside macro F1 when diagnosing failure modes. In qualitative inspection, the most frequent confusions tended to occur among closely related families whose outputs share similar surface formatting and library usage conventions, reinforcing the need_for robust evidence aggregation when discriminative cues are sparse or localized.

4.2 Ablation studies

Our ablation results on the validation split (Table 2) underline that most of the gains come from how we handle long snippets and how we aggregate evidence across them. When we replace our windowed pipeline with a simple truncation strategy that keeps only the first L tokens, performance drops sharply to 54.87 macro F1 with 85.19% accuracy, which supports the intuition that attribution cues are not reliably concentrated at the beginning of a snippet. Removing the explicit language prefix also hurts, reducing macro F1 to 56.14, suggesting that a minimal language cue helps the classifier interpret syntax and idioms consistently across languages. Among aggregation strategies, major-

Rank	Team	F1-Score
1	TeleAI	0.508
2	TocToc	0.455
3	Yuvan Ramesh	0.448
4	SYSUPporter	0.433
5	WWTC@UniA	0.429
6	Wooo2002	0.429
7	Jany-Gabriel Ispas	0.422
8	CodeDet-NITS	0.415
31	adobosz	0.254
32	CodeBERT-Baseline	0.228

Table 1: Official leaderboard ranking for the submission track, reporting each team's final F1-score (higher is better) and our team is highlighted in *Green*.

System variant	Macro F1 (%)	Accuracy (%)
Qwen2.5-Coder-7B (baseline)	62.45	93.10
Truncation baseline	54.87	85.19
No language prefix	56.14	86.02
Majority vote pooling	58.03	87.46
Stride 256	57.29	86.91
Qwen2.5-Coder-3B-QL-win*	*60.32*	*89.23*
Qwen2.5-Coder-7B-QL-win*	*64.61*	*95.41*

*Uses windowing (sliding-window).

Table 2: Ablation results on the Validation set. QL indicates *QLoRA*. Our results were highlighted in ***Orange***.

ity vote pooling underperforms logit averaging at 58.03 macro F1, indicating that soft aggregation preserves useful uncertainty across windows and is less brittle when only a few windows contain strong signals. We also observe a trade off between compute and coverage: increasing stride to 256 lowers macro F1 to 57.29, consistent with fewer overlapping views of the same example. Finally, both capacity and parameter efficient adaptation contribute positively, with the QLoRA windowed variants performing best overall; the 3B configuration reaches 60.32 macro F1 while the 7B QLoRA windowed model achieves the strongest validation performance at 64.61 macro F1 and 95.41% accuracy.

5 Conclusion

We presented our submission to SemEval 2026 Task 13 Subtask B, addressing multi class code authorship detection across human code and ten LLM families. Our system fine tunes Qwen2.5 Coder models using QLoRA and relies on a token complete long context strategy that replaces truncation with overlapping sliding windows, combined with streaming window inference and logit averaging to produce a single prediction per snippet. This design is simple to reproduce, scales to very long inputs, and aligns model selection with the official macro F1 metric. On the official test set leaderboard, our submission ranked **8th** (**CodeDet-NITS**, macro F1 = 0.41570), demonstrating that parameter efficient fine tuning with windowed evidence aggregation is an effective and practical approach for fine grained AI code attribution in the shared task setting.

References

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In *24th USENIX security symposium (USENIX Security 15)*, pages 255–270.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D Manning, and Chelsea Finn. 2023. Detectgpt: Zero-shot machine-generated text detection using probability curvature. In *International conference on machine learning*, pages 24950–24962. PMLR.

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.