CodeHunters at SemEval-2026 Task 13: Detecting Machine-Generated Code by Using Different Transformer Models

Daniel-Antoniu Dumitru[1], **Simina Lazăr**[1], **Nicoleta Danilă (Amargheoalei)**[1]
Daniela Gîfu[2,3], **Diana Trăndăbăț**[1]
[1] Faculty of Computer Science, Alexandru Ioan Cuza University of Iaşi, Romania
[2] Institute of Computer Science, Romanian Academy - Iasi Branch
[3] Academy of Romanian Scientist
{nicoleta.danila, daniel.dumitru, simina.lazar}@info.uaic.ro
diana.trandabat@info.uaic.ro, daniela.gifu@iit.academiaromana-is.ro

Abstract

Detecting machine-generated code has become increasingly relevant in software security, maintainability, and academic integrity, especially as large language models (LLMs) increasingly produce high-quality, multi-language source code. Building on prior research in authenticity detection and stylistic analysis of AI-generated content, SemEval-2026 Task 13 introduces a large-scale, multilingual, and multi-generator benchmark for distinguishing human-written from machine-generated code. Our system participates in both Subtask A (binary detection) and Subtask B (multi-class authorship attribution across LLM families). We evaluate three pretrained transformer models-UniXCoder, CodeBERT, and CodeT5-fine-tuned on the official datasets. UniXCoder consistently achieves the strongest performance, obtaining macro-F1 scores of 0.58724 (Subtask A) and 0.32469 (Subtask B) on the official test sets. These results align with recent findings in AI-generated content detection and highlight persistent challenges in generalization across programming languages, domains, and generator families [1].

1 Introduction

The rapid evolution of large language models (LLMs) such as Code LLaMA, GPT-family models, and other proprietary systems has fundamentally reshaped software development practices. These models can now generate syntactically correct and semantically coherent code across multiple programming languages, enabling new forms of automation but also raising concerns related to software security, intellectual property, academic integrity, and long-term maintainability. As a result, detecting whether a code snippet is human-written or machine-generated has become a critical research problem. SemEval-2026 Task 13 (Orel et al., 2026b) extends this line of research by providing a large-scale, multilingual, and multi-generator evaluation framework.

2 Background

Transformer-based architectures have become the foundation of modern approaches to code understanding and generation. Pretrained models such as CodeBERT (Zhangyin et al., 2020), UniXCoder (Yue et al., 2021) and codeT5 (Ming et al., 2021) learn joint representations of source code and natural language, enabling strong performance across tasks including code summarization, generation, and defect detection. However, these models are not explicitly optimized to capture stylistic or structural patterns that differentiate human-written code from LLM-generated code.

The detection of machine-generated code has recently emerged as a distinct research direction. CoDet-M4 (Orel et al., 2025a) demonstrated that detection performance drops significantly in cross-domain and cross-generator scenarios. SemEval-2026 Task 13 (Orel et al., 2026b) (Orel et al., 2026a) (Orel et al., 2025b) extends this line of work by evaluating systems across multiple programming languages, generator families, and application settings. Similar multilingual benchmarks, such as CodeMirage (Guo et al., 2025), further highlight the difficulty of generalizing across languages and paraphrasing strategies. These challenges parallel long-standing findings in computational linguistics regarding authenticity detection and stylistic analysis. Prior work has shown that linguistic, semantic, and structural cues can be leveraged to discriminate between human and

[1] https://github.com/Dumitru-Daniel-Antoniu/DMCGwMLPGaAS

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 270–276

machine-produced text, including systems for fake news detection (Gifu, 2023), emotion identification in online forums (Gifu and Cioca, 2014), offensive language detection in Romanian social media (Trandabat et al., 2022), and multi-dimensional analysis of political discourse (Gifu and Cristea, 2012). More recent studies demonstrate that neural models imprint detectable stylistic signatures in AI-generated fake news (Trandabat and Gifu, 2023) and that transformer-based architectures can effectively discriminate between human and AI-generated text (Gifu and Covaci, 2025). These insights align with observations in code-based detection, where formatting regularities, naming conventions, and structural patterns often reveal the generative origin of source code.

Recent work also emphasizes the importance of structural and stylistic cues beyond token-level modeling. Studies such as *Whitespaces Don't Lie* (Nirob et al., 2026) and *Between Lines of Code* (Shi et al., 2024) show that formatting and structural organization contribute significantly to distinguishing human from machine-generated code. Additional research on generator attribution (Oedingen et al., 2023); (Seongmin et al., 2025); (Norbert et al., 2025) demonstrates that LLMs leave identifiable stylistic fingerprints, suggesting that authorship detection may remain feasible even after code transformations. Prior SemEval systems (Alexandru et al., 2024) and (Dan-Ioan et al., 2025) show that combining pretrained representations with task-specific heuristics can be highly effective. These converging findings, across natural language, stylometry, and code analysis, have directly inspired the unified transformer-based approach explored in this study.

3 System Overview

This section presents the overall architecture and experimental setup used for both subtasks of SemEval-2026 Task 13 (Orel et al., 2026b). We first describe the datasets and their characteristics, then detail the unified model architecture based on UniXCoder, and finally outline the training strategy adopted for binary and multi-class code authorship detection.

3.1 Dataset

Subtask A addresses binary classification between human-written and LLM-generated code. The training set contains $500,000$ samples (approximately $238,000$ human, $262,000$ machine-generated), with an additional $100,000$ samples for validation. The dataset is dominated by Python (approx. 91%), followed by C++ (around 5%), with the remaining languages accounting for 4%. Evaluation includes combinations of seen and unseen programming languages (C++, Python, and Java versus Go, PHP, C#, C, and JavaScript) and domains (algorithmic versus research and production domains), enabling systematic assessment of cross-domain and cross-generator generalization.

Subtask B extends the task to multi-class authorship attribution across one human class and ten LLM families. The training set again includes $500,000$ samples, but with a highly imbalanced distribution: $442,000$ human-written samples and between $2,000$ - $10,000$ samples per LLM family. Programming languages are more evenly distributed (around 28% Java, 27% Python, and 45% other languages). Evaluation is performed under seen-author and unseen-author conditions, where test-time generators may belong to known families but not appear in training.

3.2 Model Architecture

We adopt an unified architecture, as illustrated in Figure 1, for both subtasks, centered on a pretrained UniXCoder encoder fine-tuned end-to-end. The pipeline consists of the following components:

- **Input processing**
  Each sample consists of a raw source-code snippet, optionally containing comments or docstrings. The code is tokenized using the UniXCoder tokenizer, which converts the input into subword units compatible with the pretrained model. To ensure computational efficiency and consistent batch processing, sequences are truncated to a maximum of 512 tokens for Subtask A and 192 tokens for Subtask B.

- **Contextual encoding**
  The tokenized sequence is passed through the UniXCoder encoder, a Transformer-based model pretrained on large-scale multimodal corpora for code understanding and generation. UniXCoder produces contextualized embeddings that capture syntactic structure, semantic relations, and cross-token dependencies within the code. During fine-tuning, all encoder parameters are updated jointly with the classification head.

- **Sequence pooling**
  To obtain a fixed-size representation for each code snippet, we apply UniXCoder's built-in pooling mechanism to the final hidden states. This pooled embedding serves as a compact summary of the entire input sequence and forms the shared representation used across both subtasks.

- **Task-specific classification heads**
  For Subtask A, the pooled representation is fed into a linear layer with two output units corresponding to human-written and LLM-generated code.

  For Subtask B, the classification head outputs eleven classes: one human class and ten LLM families.

  In both cases, a softmax layer produces normalized class probabilities, and the predicted label corresponds to the highest-scoring class.

- **Inference**
  The same forward pass is applied during validation and testing. Subtask A predictions are evaluated under combinations of seen/unseen languages and domains, while Subtask B predictions are evaluated under seen-author and unseen-author conditions.

4 Experimental Setup

For the experimental setup, multiple pretrained code representation models were explored to assess their suitability for the proposed detection tasks. In particular, UniXCoder, CodeT5, and CodeBERT were used as backbone models during training and validation. These models were selected due to their widespread adoption in code understanding tasks and their ability to capture semantic and syntactic properties of source code through large-scale pretraining.

Model performance was evaluated using macro-F1, the official metric of the competition. Macro-F1 equally weights all classes, making it appropriate for imbalanced datasets. It is computed as:

$$\text{Macro-F1} = \frac{1}{C} \sum_{c=1}^{C} \frac{2 \cdot P_c \cdot R_c}{P_c + R_c}$$

where P_c and R_c represent the precision and recall for a specific label, and C denotes the total number of labels (classes).

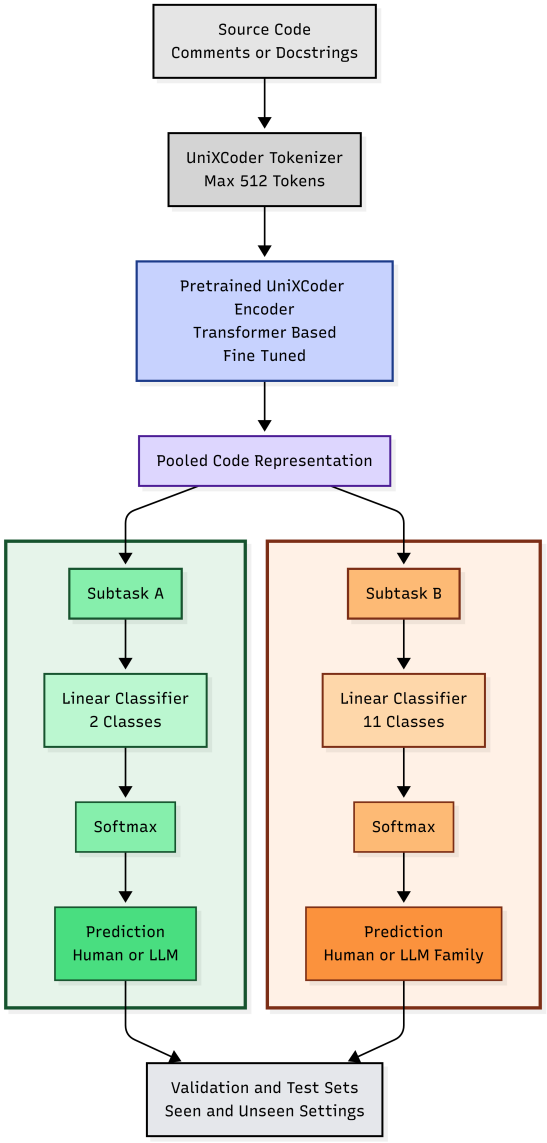

Figure 1: Overview of the UniXCoder-based architecture used for both subtasks.

4.1 Data Preprocessing

We applied minimal preprocessing to preserve structural and lexical cues known to be informative for distinguishing human-written from LLM-generated code. Comments and docstrings were removed, and indentation was normalized after comment removal. Although this improves consistency across languages, empirical validation showed that excessive preprocessing can remove stylistic and semantic signals that pretrained models rely on. As reflected in our results, preprocessing led to a performance decrease in Subtask B, indicating that lexical richness and natural-language tokens contribute to identifying LLM families.

4.2 Sample Training

Subtask A (binary classification)

Class imbalance is mild. To ensure equal contribution of both classes, we applied class weighting in the cross-entropy loss:

$$w_i = \frac{N}{2n_i}$$

where N is the total number of samples and n_i the number of samples in class i. This guarantees $n_i \cdot w_i = N/2$ for both classes.

Subtask B (multi-class authorship attribution)

The dataset is highly imbalanced, with the human class dominating the distribution. To mitigate this, we combined:

- **inverse square-root class weighting**

$$w_i = \frac{1}{\sqrt{n_i}}$$

  where n_i denotes the number of samples associated with class i. Using the square root of the class frequency reduces the dominance of the majority class while avoiding excessively large weights for minority classes.

- **balanced batch sampling** (approximately $140,000$ human, $60,000$ machine-generated, covering all LLM families)

- **end-to-end fine-tuning** of UniXCoder and the classification head.

This combination improved training stability and minority-class performance.

4.3 Training Details

Training for both subtasks follows the same general procedure and is based on fine-tuning a pretrained UniXCoder model using labeled code samples. All experiments were conducted using Kaggle notebooks with an T4 $\times$ 2 GPU configuration, which provided sufficient computational resources for training large Transformer-based models with long input sequences. For both subtasks, the input code is tokenized using the UniXCoder tokenizer and truncated to a maximum sequence length of 512 tokens (Subtask A) or 192 tokens (Subtask B). Tokenized samples are processed in batches and dynamically padded using a data collator to ensure efficient GPU utilization. The pretrained UniXCoder encoder is then fine-tuned end-to-end together with a lightweight task-specific classification head. Training is performed using the Hugging Face Trainer framework, which handles batching, evaluation, checkpointing, and metric computation. In Subtask A, optimization uses a cosine learning-rate scheduler with warmup. Early stopping is applied based on validation macro-F1, and the best-performing checkpoint is retained. Table 1 summarizes the main hyperparameters used for training subtask A.

The combination of a moderate learning rate, gradient accumulation, and mixed-precision training allows for stable optimization while keeping GPU memory usage within practical limits.

Hyperparameter	Value
Pretrained model	UniXCoder-base
Max sequence length	512
Learning rate	2×10^{-5}
Batch size (per device)	8
Gradient accumulation steps	2
Number of epochs	3
Weight decay	0.01
Warmup ratio	0.1

Table 1: Training hyperparameters for subtask A.

Subtask B follows the same training pipeline, model architecture, and optimization strategy. The main differences lie in the multi-class objective and the handling of severe label imbalance. Class weights are computed using the inverse square-root of class frequencies and incorporated into the loss function. Additionally, each training batch contains approximately 70% human-written samples and 30% LLM-generated samples, ensuring exposure to all generator families.

Hyperparameter	Value
Pretrained model	UniXCoder-base
Max sequence length	192
Learning rate	2×10^{-5}
Batch size (per device)	16 (train), 32 (eval)
Gradient accumulation steps	2
Number of epochs	4
Weight decay	0.02
Warmup ratio	0.1

Table 2: Training hyperparameters for subtask B.

Across both subtasks, model evaluation is performed at the end of each epoch on the validation set, and the checkpoint with the highest macro-F1 score is selected for final inference.

Table 2 reports the hyperparameters used for subtask B. Aside from the number of output classes and the class-weighting strategy, the training configuration closely mirrors that of subtask A, ensuring consistency across both tasks.

5 Results

For Subtask A, our system achieved a macro-F1 score of 0.58724, ranking 26th out of 81 participating teams. This represents a substantial improvement over the official baseline, which used CodeBERT and obtained a macro-F1 of 0.30530. The large margin between the baseline and our final score highlights the effectiveness of leveraging stronger pretrained code representations and task-specific training strategies. To identify the most suitable backbone model, we conducted comparative experiments on the sample test set. Table 3 reports the performance of UniXCoder, CodeBERT, and CodeT5 for Subtask A. While CodeBERT achieves slightly higher accuracy, UniXCoder obtains the highest macro-F1 score, which is the primary evaluation metric of the competition. This indicates that UniXCoder captures more discriminative structural and stylistic cues relevant for distinguishing human-written from LLM-generated code.

Model	Accuracy	Macro F1
CodeBERT-base	**0.5260**	0.3916
UniXCoder-base	0.5120	**0.3962**
CodeT5-base	0.4230	0.3758

Table 3: Sample test performance of pretrained models for subtask A.

For Subtask B, we evaluated the same three pretrained models under two conditions: (i) without additional preprocessing and (ii) after applying the preprocessing pipeline described in Section 4.1. The results are presented in Table 4 and Table 5, respectively.

As shown in Table 4, UniXCoder again achieves the strongest performance across all metrics (accuracy, macro-F1, precision, and recall) when no preprocessing is applied. This suggests that the pretrained representations already encode robust structural and lexical cues that are useful for distinguishing LLM families.

Model	Acc.	Macro F1	Prec.	Rec.
UniXCoder	**0.663**	**0.6373**	**0.6556**	**0.663**
CodeBERT	0.591	0.5862	0.6124	0.591
CodeT5	0.621	0.6011	0.6121	0.621

Table 4: Sample test performance for subtask B using the base configuration without additional preprocessing.

Model	Acc.	Macro F1	Prec.	Rec.
UniXCoder	**0.628**	**0.5868**	**0.5661**	**0.628**
CodeBERT	0.579	0.5462	0.5312	0.579
CodeT5	0.602	0.5498	0.5631	0.602

Table 5: Sample test performance for subtask B after applying preprocessing.

In contrast, applying preprocessing leads to a consistent performance drop across all models, as illustrated in Table 5. This confirms that lexical richness, comments, and natural-language tokens contribute meaningfully to multi-class authorship attribution. The degradation is particularly visible for UniXCoder, whose macro-F1 decreases from 0.6373 to 0.5868, a relative drop of approximately 8%. This suggests that removing comments and docstrings eliminates stylistic signals that help differentiate LLM families.

Based on these findings, UniXCoder without additional preprocessing was selected as the final model for both subtasks. This configuration balances strong empirical performance with architectural simplicity and avoids the risk of information loss introduced by aggressive preprocessing.

On the official leaderboard, our system achieved a macro-F1 score of 0.32469 for Subtask B, ranking 23rd out of 34 teams. The baseline system for this subtask, also based on CodeBERT, obtained a macro-F1 of 0.22858, confirming the benefits of using a more expressive pretrained encoder combined with tailored imbalance-handling strategies.

Although Subtask B is substantially more challenging due to extreme class imbalance and overlapping generation behaviors across LLM families, the combination of label sampling and cost-sensitive learning improved the model's ability to recognize minority classes. The achieved ranking reflects the difficulty of fine-grained authorship attribution while still demonstrating the robustness of the proposed architecture.

Overall, the results across both subtasks demonstrate that a unified UniXCoder-based architecture, combined with appropriate training strategies, provides a strong and competitive solution for code authorship detection. The consistent improvements over the CodeBERT baselines validate the design choices made in terms of model selection, loss formulation, and data sampling.

6 Discussions

The results across both subtasks reveal several consistent patterns regarding the behavior of pretrained code models and the nature of machine-generated code. First, UniXCoder systematically outperforms CodeBERT and CodeT5 in terms of macro-F1 across all evaluation settings (Table 3, Table 4 and Table 5). This stems from both architectural and pretraining differences. Unlike CodeBERT, whose token-level objectives favor local pattern recognition over global sequence-level discrimination, and CodeT5, whose encoder-decoder design introduces an inherent mismatch when repurposed for classification, UniXCoder's unified cross-modal pretraining yields richer contextual embeddings that better capture the structural and stylistic regularities relevant to authorship detection.

Second, the strong performance of UniXCoder in Subtask A indicates that binary detection relies heavily on global structural cues, such as formatting regularities, indentation patterns, and naming conventions, that remain relatively stable across programming languages and domains. The relatively small gap between accuracy and macro-F1 in Table 3 further suggests that both classes are reasonably separable when using a sufficiently expressive encoder.

In contrast, Subtask B proves substantially more challenging. The performance drop observed across all models when moving from binary to multi-class authorship attribution reflects the increased complexity of distinguishing between multiple LLM families with overlapping generation behaviors. The severe class imbalance amplifies this difficulty, as minority LLM families contribute disproportionately to the macro-F1 metric. The improvements obtained through inverse square-root class weighting and balanced batch sampling demonstrate that cost-sensitive learning is essential for stabilizing training under such conditions.

A notable finding is the negative impact of preprocessing on Subtask B performance (Table 5). Removing comments, docstrings, and natural-language tokens eliminates stylistic signals that appear to be highly informative for identifying specific LLM families. This aligns with prior work in text authenticity detection, where lexical richness and discourse markers often serve as strong indicators of generative origin. The fact that UniXCoder's performance decreases by approximately 8% macro-F1 after preprocessing suggests that authorship attribution relies on a combination of structural and lexical cues, not solely on code-level patterns.

Finally, the gap between sample test performance (Table 3, Table 4 and Table 5) and official leaderboard results highlights the difficulty of generalizing to unseen languages, domains, and generator families. This reinforces the central research question posed in the Introduction: pretrained transformer models capture meaningful stylistic and structural patterns, but their generalization remains limited when confronted with distribution shifts.

7 Conclusion

Across both subtasks of SemEval-2026 Task 13, UniXCoder consistently emerged as the strongest backbone model, outperforming CodeBERT and CodeT5 in terms of macro-F1. The results for Subtask A indicate that pretrained transformer models are capable of capturing structural regularities that transfer across programming languages and application domains. The substantial improvement over the official baseline further suggests that these representations encode discriminative patterns that remain stable even under distribution shifts.

The findings point to several promising directions for future work: improving robustness to distribution shifts, developing representations that better capture generator-specific stylistic signatures, and extending benchmarks to include emerging LLMs and more diverse programming languages. Advances along these lines will be essential to build reliable systems capable of identifying AI-generated code in increasingly heterogeneous real-world environments.

Acknowledgments

This work was carried out partially within the project "Tools for Processing Online Texts Specific to Cultural and Scientific Diplomacy", funded by the Academy of Romanian Scientists.

References

Mihai Alexandru, Cosmin Ciocoiu, Ionut Maniga, Oana Ungureanu, Daniela Gifu, and Diana Trandabat. 2024. Linguistech at semeval-2024 task 10: Emotion discovery and reasoning in conversation. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*.

Grigorita Dan-Ioan, Pricop Teodor-Cosmin, Suteu Stefan-Alexandru, Gifu Daniela, and Trandabat Diana. 2025. Fii the best at semeval-2025 task 2. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*.

Daniela Gifu. 2023. An intelligent system for detecting fake news. pages 1058–1065.

Daniela Gifu and Marius Cioca. 2014. Detecting emotions in comments on forums. *International Journal of Computers Communications and Control*, pages 694–702.

Daniela Gifu and Silviu-Vasile Covaci. 2025. Artificial intelligence vs. human: Decoding text authenticity with transformers. *Future Internet*.

Daniela Gifu and Dan Cristea. 2012. Multi-dimensional analysis of political language. pages 213–221. SPRINGER.

Hanxi Guo, Siyuan Cheng, Kaiyuan Zhang, Guangyu Shen, and Xiangyu Zhang. 2025. Codemirage: A multi-lingual benchmark for detecting ai-generated and paraphrased source code. *arXiv preprint arXiv:2506.11059*.

Chen Ming, Jiang Nan, Wang Yue, Gong Ming, Shou Linjun, Qin Bing, Liu Zhiyuan, and Ji Rongrong. 2021. Codet5: Pretrained encoder-decoder models for code understanding and generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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. *arXiv preprint arXiv:2601.19264*.

Tihanyi Norbert, Cherif Bilal, Dubniczky Rákos András, Ferrag Mohamed Amine, and Bisztray Tamás. 2025. The hidden DNA of LLM-generated javascript: Structural patterns enable high accuracy authorship attribution. *arXiv preprint*.

Marc Oedingen, Raphael C. Engelhardt, Robin Denz, Maximilian Hammer, and Wolfgang Konen. 2023. Chatgpt code detection: Techniques for uncovering the source of code. *AI*.

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 31251–31277.

Park Seongmin, Jin Hyunwoo, Cha Junhyung, and Han Young-Seob. 2025. Detection of LLM paraphrased code and identification of the responsible LLM using coding style features. *arXiv preprint*.

Yuling Shi, Hongyu Zhang, Chengcheng Wan, and Xiaodong Gu. 2024. Between lines of code: Unraveling the distinct patterns of machine and human programmers. *arXiv preprint arXiv:2401.06461*.

Diana Trandabat and Daniela Gifu. 2023. Discriminating ai-generated fake news. *Procedia Computer Science*, pages 3822–3831.

Diana Trandabat, Daniela Gifu, and Adrian Plesescu. 2022. Detecting offensive language in romanian social media. *Procedia Computer Science*, pages 2883––2890.

Wang Yue, Wang Weishi, Joty Shafiq R., Hoi Steven C. H., Li Xiaopeng, and Wang Liang. 2021. Unixcoder: Unified cross-modal pre-training for code understanding and generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Feng Zhangyin, Guo Daya, Tang Duyu, Duan Nan, Feng Xiaocheng, Gong Ming, Shou Linjun, Qin Bing, and Liu Ting. 2020. Codebert: A pre-trained model for programming and natural languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.