SYSUpporter at SemEval-2026 Task 13: Leveraging Stylistic Signals and Language-Aware Truncation for Machine-Generated Code Detection

Longfeng Chen[1*†], **Zheng Xiao**[2*]
[1]South China University of Technology, Guangzhou, China
[2]Peking University, Beijing, China
ftclf_dh@mail.scut.edu.cn, zhengxiao@stu.pku.edu.cn

Abstract

This paper describes our system for SemEval-2026 Task 13 Subtask B, which requires attributing source code to either a human author or one of 10 LLM families. Guided by dataset analysis, we identify three practical challenges: formatting fingerprints discarded by tokenizers, heterogeneous code lengths, and extreme class imbalance. We build on `unixcoder-base` with *Explicit Stylistic Prompting*, *Language-Aware Truncation*, and imbalance-aware training (Focal Loss, GeM pooling, multi-sample dropout, and bucket batching). Our system achieves 0.434 Macro F1 on the official hidden test set, ranking 4th out of 34 teams with only 125M parameters. Controlled 5-fold cross-validation confirms that each component contributes to the final system, and a formatting-normalization study quantifies the model's reliance on formatting cues.

1 Introduction

The proliferation of Large Language Models (LLMs) in software development (Chen et al., 2021) has introduced unprecedented challenges in code attribution, copyright enforcement, and academic integrity. SemEval-2026 Task 13 Subtask B (Orel et al., 2026) addresses this by formulating a fine-grained, 11-way multi-class attribution task: distinguishing human-written code from code generated by 10 distinct LLM families across 8 programming languages.

Our approach is fundamentally driven by a systematic analysis of the dataset. We identify three critical challenges that standard fine-tuning pipelines fail to address:

Invisible typographic fingerprints. Different LLMs leave distinct formatting habits (e.g., trailing whitespace, tab vs. space indentation) that are silently discarded by standard subword tokenizers, wasting highly discriminative signals.

Heterogeneous code length distributions. Code lengths vary dramatically across generators and languages, and different languages place critical structures (e.g., imports vs. entry points) at different positions, so a fixed truncation strategy systematically discards informative regions.

Extreme class imbalance. The dataset exhibits a 224.6:1 imbalance ratio between the largest and smallest classes, while Macro F1 weights all 11 classes equally, making minority-class performance disproportionately important.

To address these challenges, we build upon `unixcoder-base` (Guo et al., 2022) and propose a data-analysis-driven system. For Challenge 1, we introduce *Explicit Stylistic Prompting* that converts invisible formatting habits into discriminative special tokens. For Challenge 2, we design *Language-Aware Truncation* with dynamic head-to-tail ratios and newline boundary alignment. For Challenge 3, we adopt imbalance-aware training that combines Focal Loss with class weights, Generalized Mean (GeM) pooling, and multi-sample dropout, together with bucket batching for efficiency. Our system achieved a Macro F1 of 0.434, securing 4th place among 34 teams.

2 Related Work

Machine-Generated Text Detection. Detecting machine-generated text has been studied through zero-shot statistical methods (Mitchell et al., 2023), supervised classifiers, and shared tasks such as SemEval-2024 Task 8 (Wang et al., 2024). However, most work targets natural language; extending detection to source code, with its unique syntactic structure and formatting conventions, remains underexplored.

Code Authorship Attribution. Code stylometry has been used to de-anonymize human program-

[*]These authors contributed equally.
[†]Corresponding author.

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2903–2909

mers, and deep learning methods have scaled authorship identification to large, language-oblivious settings (Abuhamad et al., 2018). Our task differs fundamentally: we discriminate between human code and code from distinct LLM families, where intra-family stylistic variation is much smaller.

Pre-trained Models for Code. CodeBERT (Feng et al., 2020) pioneered pre-trained Transformers for programming languages, followed by GraphCodeBERT (Guo et al., 2020) with data flow information. UniXcoder (Guo et al., 2022) unified cross-modal pre-training across code, ASTs, and comments. More recently, decoder-only models such as StarCoder (Li et al., 2023) and DeepSeek-Coder (Guo et al., 2024) have demonstrated strong code generation capabilities. We adopt UniXcoder for its cross-language representation ability and computational efficiency.

3 Dataset and Task Overview

SemEval-2026 Task 13 Subtask B formulates code attribution as an 11-way classification problem: given a code snippet with its programming language label, the system must predict whether it was written by a human or generated by one of 10 LLM families (DeepSeek-AI, Qwen (Hui et al., 2024), 01-ai, BigCode, Gemma, Phi (Abdin et al., 2024), Meta-LLaMA (Touvron et al., 2023), IBM-Granite, Mistral, OpenAI).

The dataset comprises 500K training, 100K validation, and 500K test samples spanning 8 languages. The 31 concrete generator identifiers provided in the data are grouped into 10 LLM families plus the Human class, yielding the 11-way label space used in the task. Our data analysis reveals properties that directly motivate our design:

Extreme Class Imbalance. Within the 500K training split, human-written code dominates at 88.42% (442,096 samples), while AI-generated classes range from 0.39% (Gemma: 1,968) to 2.16% (OpenAI: 10,810), yielding a maximum imbalance ratio of 224.6:1. Even among AI classes, the internal imbalance is 5.5:1. Under Macro F1 evaluation, each class contributes equally regardless of size, making minority class performance critical.

Heterogeneous Code Lengths. Code lengths vary dramatically: the overall mean is 1,410 characters with std=1,662, ranging from 24 to 34,346. Critically, length distributions differ substantially *across generators*: Gemma produces the longest code (mean=2,694, median=2,393), while BigCode produces the shortest (mean=842, median=469), representing a 3.2× difference. Length also varies *across languages*: PHP and Java exhibit the widest distributions.

Label	Trail.WS	Tab	Space4	Empty
Human	0.037	0.191	0.564	0.073
DeepSeek-AI	0.052	0.036	0.591	0.141
Qwen	0.035	0.019	0.583	0.174
01-ai	0.032	0.013	0.568	0.177
BigCode	0.035	0.092	0.407	0.163
Gemma	0.008	0.038	0.571	0.173
Phi	0.035	0.013	0.592	0.207
Meta-LLaMA	0.024	0.019	0.561	0.142
IBM-Granite	0.006	0.033	0.614	0.153
Mistral	0.000	0.031	0.569	0.159
OpenAI	0.024	0.029	0.575	0.198

Table 1: Formatting feature ratios per label (full 500K training set). Trail.WS = trailing whitespace ratio, Tab = tab indentation ratio, Space4 = 4-space indentation ratio, Empty = empty line ratio. Values are averaged over all samples per class.

Distinctive Stylistic Fingerprints. Our analysis of formatting features across labels reveals strong discriminative patterns that are invisible to standard tokenizers. Table 1 summarizes key findings from the full 500K training set.

Notable observations include: (1) Human code has a distinctively high tab indentation ratio (0.191), far exceeding all AI generators (0.013–0.092); (2) DeepSeek-AI exhibits the highest trailing whitespace ratio (0.052) among all labels, while Mistral has exactly zero; (3) IBM-Granite leads in 4-space indentation (0.614), while BigCode has the lowest Space4 ratio (0.407) but the highest tab ratio among AI classes (0.092); (4) Phi shows the highest empty line ratio (0.207).

Language Coverage Gaps. Four label-language combinations have zero training samples (01-ai, BigCode, Phi, OpenAI × PHP), and the test set exhibits higher median code length (1,115 vs. 812) than the training set, suggesting a distributional shift.

4 Methodology

Our system design is directly guided by the data analysis in Section 3. Each component addresses a specific data characteristic. Figure 1 illustrates the overall pipeline.

4.1 Base Model Architecture

We adopt `unixcoder-base` as our encoder. Unlike text-only models such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019), UniXcoder is pre-trained on multimodal data comprising source code, ASTs, and comments, making it well-suited for cross-language code representation.

Under our compute budget (sequence length 1024, effective batch size 512), jointly updating the full embedding matrix together with the top encoder layers, GeM pooling, and multi-sample dropout exceeded our available GPU memory. We therefore freeze the embedding layer and the first 8 encoder layers, fine-tuning only the top 4 layers, and enable gradient checkpointing. We extend the vocabulary with 11 new special tokens (7 stylistic tokens described in Section 4.2 and 4 functional tokens: `[LANG]`, `[CODE]`, `[TRUNC]`, `[WINDOW]`); because the embedding layer is frozen, their vectors are randomly initialized and remain fixed, and the fine-tuned upper layers learn to project them into task-relevant representations via self-attention. Learning these new-token embeddings via parameter-efficient tuning is left as future work (see Limitations).

4.2 Stylistic Feature Engineering

As shown in Table 1, different LLM families exhibit distinct formatting habits that serve as reliable attribution signals, yet these whitespace characters are typically absorbed during subword tokenization. We select indentation, line endings, and trailing whitespace a priori, based on four properties: (i) they originate at the *decoder* level, so they are shaped by each family's pre-training corpus and post-training templates; (ii) they are *orthogonal* to code semantics, making them complementary to UniXcoder's AST- and comment-aware representations; (iii) they fall into *tokenizer blind spots*, since BPE merges consecutive whitespace and drops trailing spaces; and (iv) they are *language-agnostic* and extractable by character-level rules. Candidates such as identifier naming, comment density, and bracket style were excluded as language-specific or semantically entangled. Section 5.6 provides a per-class a posteriori analysis.

Concretely, we extract formatting features organized into three mutually exclusive groups *before* any text normalization:

(1) **Line ending**: `[CRLF]` (Windows-style) or `[CR]` (legacy Mac-style);

(2) **Indentation**: `[TAB_INDENT]` (pure tab), `[MIXED_INDENT]` (tabs mixed with spaces), `[SPACE4]` (consistent 4-space), or `[SPACE_OTHER]` (non-standard spacing);

(3) **Trailing whitespace**: `[TRAILING_WS]` (any line with trailing spaces).

Within each group, at most one token is selected, so a given sample receives between zero and three stylistic tokens. These are registered as special tokens in the vocabulary and prepended to the input sequence, forming an explicit stylistic prompt:

`[LANG] Python [SPACE4] [TRAILING_WS] [CODE] def ...`

This design enables the self-attention mechanism to directly condition its code representations on prior stylistic information, without relying on the tokenizer to preserve formatting details. We therefore use these fingerprints as compact prompts rather than as a replacement for lexical or structural code representations.

4.3 Language-Aware Truncation

Data motivation. UniXcoder's maximum sequence length is 1024 tokens, yet our analysis shows that mean code length is 1,410 characters (with Gemma reaching 2,694), meaning a significant fraction of samples require truncation. Moreover, the syntactic structure varies by language: Python concentrates imports and class definitions at the head, while Java and C++ place critical `main` methods at the tail.

We implement a three-component truncation strategy:

- **Dynamic head-to-tail ratios** per language: Python retains 80% head / 20% tail; Java and C++ use 70%/30% to preserve tail-end entry points; other languages default to 75%/25%.
- **Newline boundary alignment**: at both head and tail truncation points, we search within a 20-token window for the nearest newline token and shift the cut point accordingly, preventing mid-statement truncation.
- **Dynamic window sampling** (training only): with 10% probability, we replace the standard head+tail truncation with a randomly positioned contiguous window, marked with a `[WINDOW]` token. This data augmentation forces the model to learn from middle-section code patterns that would otherwise be permanently discarded for long samples (e.g.,

Gemma, where 40–60% of the code exceeds the token limit).

4.4 Imbalance-Aware Training

Data motivation. The 224.6:1 class imbalance, the small absolute sample count of minority classes (as few as 1,968 for Gemma), and the Macro F1 metric jointly make minority-class performance the bottleneck. We therefore combine three complementary techniques that act at the *loss*, *pooling*, and *regularization* levels respectively, each targeting a distinct failure mode.

Focal Loss with Class Weights (loss level). To address the 224.6:1 imbalance, we use Focal Loss (Lin et al., 2017) ($\gamma = 2.0$) with frequency-smoothed class weights $w_c = 1/\sqrt{n_c}$, normalized across classes. The focal term $(1 - p_t)^\gamma$ automatically downweights well-classified samples (predominantly Human), concentrating learning on hard, underrepresented classes.

GeM Pooling (pooling level). We replace standard [CLS]-token or mean pooling with Generalized Mean (GeM) pooling using a learnable parameter p (initialized to 3):

$$\mathrm{GeM}(\mathbf{x}) = \left(\frac{1}{\sum m_i} \sum (x_i \cdot m_i)^p \right)^{\frac{1}{p}} \quad (1)$$

where m_i is the attention mask. When $p > 1$, GeM amplifies high-activation positions, effectively upweighting the most discriminative tokens (including our injected stylistic tokens) while suppressing padding regions.

Multi-Sample Dropout (regularization level). Given that the smallest class (Gemma) has only 1,968 samples, overfitting on minority classes is a serious concern. Standard dropout addresses this by randomly zeroing activations during training. We extend this with 5 parallel dropout layers with rates from 0.1 to 0.18, averaging their logits. This acts as an implicit ensemble that reduces prediction variance, which is critical when per-fold training samples for rare classes number fewer than 1,600.

4.5 Efficient Batching

Bucket Batch Sampler. The extreme length variance (std=1,662, min=24, max=34,346) makes random batching highly inefficient: a batch mixing 50-token and 1024-token samples wastes over 50% computation on padding. We sort samples by length, partition into batches of adjacent samples, and shuffle only at the batch level, reducing padding waste to below 10%.

Input: Raw code + language label
↓ *Extract formatting features*
↓ *Language-aware truncation*
↓ *Prepend stylistic tokens*
`[LANG] Py [SPACE4] [CODE] def ...`
↓ *UniXcoder encoder (top 4 layers fine-tuned)*
↓ *GeM pooling*
↓ *Multi-sample dropout (5×)*
↓ *Linear classifier* → 11 classes

Figure 1: System pipeline overview.

5 Experimental Setup and Results

5.1 Implementation Details

We use 5-fold stratified cross-validation on the combined training and validation sets (600K samples). The model is trained with AdamW optimizer (lr=1×10^{-4}, weight decay=0.01), OneCycleLR scheduler (pct_start=0.1), and gradient accumulation (n=2, effective batch size=512). We train for 10 epochs per fold with AMP (fp16) and gradient clipping (max norm=20). During training, head-to-tail ratios are randomly sampled within language-specific ranges for data augmentation.

5.2 Evaluation Protocol

We report the final submission on the official hidden test set, whose labels are unavailable, and use 5-fold stratified CV on labeled data for all ablations, normalization tests, and per-class analyses.

We observe a substantial absolute gap between 5-fold CV Macro F1 (0.640) and hidden-test Macro F1 (0.434). Two dataset properties documented in Section 3 plausibly explain this gap: (i) the test set has a 37% higher median code length than the training set (1,115 vs. 812 characters), so truncation-induced information loss is heavier at test time, particularly for long-code classes such as Gemma; (ii) four label–language combinations (01-ai, BigCode, Phi, OpenAI × PHP) have zero training samples, forcing the model to extrapolate on these cells. Despite the absolute offset, the single-model leaderboard progression in Table 2 (CodeBERT 0.363 → GraphCodeBERT 0.364 → UniXcoder 0.394 → Full System 0.434) shows that relative improvements observed during our CV-based development consistently transfer to the hidden test. We therefore use CV results for relative component analysis

System	Macro F1
CodeBERT	0.363
GraphCodeBERT	0.364
UniXcoder	0.394
UniXcoder + Full System (Ours)	**0.434**
UniXcoder + LightGBM ensemble	0.402

Table 2: System development progression on the official hidden-test leaderboard. The LightGBM ensemble was submitted *after* the full system, resulting in a score decrease.

rather than hidden-test estimation in Sections 5.4–5.6.

5.3 Main Results

Table 2 presents the progression of our system development on the official leaderboard.

CodeBERT and GraphCodeBERT (0.363/0.364; head-only truncation, 512 tokens, cross-entropy) serve as baselines; their near-identical scores suggest that data-flow information alone provides limited benefit for stylistic attribution. Moving to UniXcoder (0.394) with a 1024-token context gives a +0.030 gain. Adding our full system—stylistic prompting, language-aware truncation, GeM pooling, multi-sample dropout, Focal Loss, and bucket batching—yields 0.434, a further +0.040 improvement. The post-hoc LightGBM stack (0.402) underperforms the single model and is analyzed in Section 5.7.

5.4 Controlled Ablation Study

Table 3 reports controlled 5-fold CV ablations. Removing Focal Loss and class weights causes the largest degradation (0.037 Macro F1), showing that imbalance-aware optimization is the dominant contributor under Macro F1. Language-aware truncation and stylistic prompting provide smaller but consistent gains (0.012 and 0.011), while GeM pooling contributes an additional 0.008. The stylistic-tokens-only setting reaches 0.587 Macro F1, indicating that stylistic prompts are useful but insufficient without truncation, imbalance handling, and pooling.

5.5 Robustness to Formatting Normalization

We re-evaluate the full-system fold models after code normalization to test whether the model relies exclusively on formatting artifacts. Light normalization removes trailing whitespace and unifies line endings; strong normalization further normalizes indentation and redundant blank lines in a formatter-style manner. Table 4 shows that light normalization causes only a small drop (0.006), whereas strong normalization reduces Macro F1 by 0.038. This suggests that formatting features are informative but not the sole signal: the system retains non-trivial performance after aggressive normalization, while its sensitivity to strong formatting changes remains a robustness limitation.

Variant	CV Macro F1	Drop / Gap
Full system	0.640	–
w/o Focal Loss / class weights	0.603	-0.037
w/o language-aware truncation	0.628	-0.012
w/o stylistic tokens	0.629	-0.011
w/o GeM pooling	0.632	-0.008
Stylistic tokens only	0.587	-0.053

Table 3: 5-fold CV ablations. *w/o GeM pooling* uses mean pooling. The green row (*stylistic tokens only*) keeps stylistic prompting while removing language-aware truncation, Focal Loss / class weights, GeM pooling, and multi-sample dropout (i.e., cross-entropy with mean pooling and standard head-only truncation).

Input setting	CV Macro F1	Drop
Original	0.640 ± 0.006	–
Light normalization	0.634 ± 0.007	-0.006
Strong normalization	0.602 ± 0.011	-0.038

Table 4: 5-fold CV under formatting normalization. Light normalization removes trailing whitespace and unifies line endings; strong normalization additionally simulates formatter-style tools such as `black`, `clang-format`, and `prettier`.

5.6 Per-Class Effect of Stylistic Tokens

Table 5 compares per-class F1 with and without stylistic tokens. Stylistic prompting mainly benefits Mistral (+0.031), DeepSeek-AI (+0.024), BigCode (+0.023), and IBM-Granite (+0.018), while Gemma and Meta-LLaMA receive negligible or slightly negative changes. The beneficiaries align with the distinctive formatting statistics in Table 1 (e.g., Mistral's near-zero trailing whitespace and BigCode's relatively high tab ratio among AI classes), supporting our a priori feature selection; the null effects on Gemma and Meta-LLaMA indicate that stylistic prompting is an auxiliary cue rather than a universal class separator.

5.7 Post-hoc LightGBM Ensemble Analysis

After the full single-model system achieved 0.434, a second-stage LightGBM (Ke et al., 2017) stack

Class	w/o Style	w/ Style	Δ
Human	0.943	0.946	+0.003
DeepSeek-AI	0.618	0.642	+0.024
Qwen	0.600	0.608	+0.008
01-ai	0.540	0.547	+0.007
BigCode	0.608	0.631	+0.023
Gemma	0.506	0.503	-0.003
Phi	0.573	0.578	+0.005
Meta-LLaMA	0.603	0.601	-0.002
IBM-Granite	0.628	0.646	+0.018
Mistral	0.589	0.620	+0.031
OpenAI	0.711	0.717	+0.006
Macro F1	0.629	0.640	+0.011

Table 5: Per-class F1 with and without stylistic tokens.

Rank	Team	Macro F1
1	TeleAI	0.508
2	TocToc	0.456
3	Yuvan	0.449
4	SYSUpporter (Ours)	0.434
5	WWTC	0.430

Table 6: Official leaderboard for Subtask B (top 5 of 34 teams).

using UniXcoder OOF predictions plus handcrafted and TF-IDF features dropped to 0.402. This suggests feature overlap with the fine-tuned encoder and validation overfitting rather than complementary signal.

5.8 Discussion: Efficiency–Performance Trade-off

To contextualize our results, we present the official leaderboard standings and compare our system with top-ranked approaches across multiple dimensions.

Official Leaderboard. Table 6 shows the top 5 teams out of 34 participants. Our system ranks 4th with a Macro F1 of 0.434, only 0.015 behind the 3rd-place team and 0.074 behind the winning solution.

System Comparison. Table 7 compares the computational requirements of top systems based on their system descriptions.

Extreme parameter efficiency. With only 125M parameters, our system is 240× smaller than TeleAI's Qwen3-30B-A3B and 12× smaller than Yuvan's Qwen2.5-Coder-1.5B, yet remains competitive. This highlights the value of data-driven feature engineering under limited model capacity.

	TeleAI (Rank 1)	Yuvan (Rank 3)	WWTC (Rank 5)	Ours (Rank 4)
Params	30B (3B active)	∼1.5B	∼395M	∼125M
Max Len	≥2K	≥2K	2048	1024
Training	Full FT	LoRA	Full FT	Partial FT‡

Table 7: Comparison of computational requirements across top systems. TeleAI uses Qwen3-30B-A3B (Qwen Team, 2025), Yuvan uses Qwen2.5-Coder-1.5B, WWTC uses ModernBERT (Warner et al., 2025), and ours uses UniXcoder. FT = fine-tuning. ‡Only the top 4 encoder layers are fine-tuned (see Section 4.1).

6 Conclusion

We presented a data-analysis-driven system for SemEval-2026 Task 13 Subtask B, achieving 0.434 Macro F1 and ranking 4th/34 teams. The system combines Explicit Stylistic Prompting, Language-Aware Truncation, and imbalance-aware training to address typographic fingerprints, length heterogeneity, and class imbalance. With only 125M parameters, it remains competitive with much larger systems, showing that targeted feature engineering can partially compensate for limited model capacity.

Limitations

Our system has several limitations. First, although formatting features are useful, the normalization experiment shows that strong formatter-style normalization still reduces performance. This indicates that the model is not fully robust to aggressive formatting removal and should be complemented with deeper semantic or execution-aware signals in future work. Second, the embedding layer is frozen due to memory constraints rather than as an empirically validated design choice. Learning the embeddings of the newly added stylistic tokens, for example via parameter-efficient adaptation, may yield additional gains. Third, our language-aware truncation ratios are manually configured rather than learned; replacing them with a learned or differentiable truncation policy is a promising direction. Finally, we did not explore larger decoder-only classifiers due to resource constraints, and four label-language combinations with zero training samples are not explicitly addressed.

Acknowledgments

The authors sincerely appreciate the event organizers for maintaining the shared task and releasing the benchmark resources.

References

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Beber, and 1 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Mohammed Abuhamad, Tamer AbuHmed, Aziz Mohaisen, and DaeHun Nyang. 2018. Large-scale and language-oblivious code authorship identification. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 101–114.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

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.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, and 1 others. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, and 1 others. 2024. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

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*.

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. *Advances in neural information processing systems*, 30.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

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.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, and 1 others. 2025. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2526–2547.