Codexa at SemEval-2026 Task 13: Loss Engineering and Diverse Ensemble Strategies for Multi-Class Code Authorship Attribution

Anıl Dervişoğlu*
Department of AI and Data Engineering
Istanbul Technical University
Istanbul, Turkey
dervisoglua22@itu.edu.tr

Atakan Site
Department of AI and Data Engineering
Istanbul Technical University
Istanbul, Turkey
site21@itu.edu.tr

Abstract

We describe our system for SemEval-2026 Task 13, Subtask B: code classification into 11 categories (human-written or generated by one of 10 LLM families). The task presents extreme class imbalance and distribution shift across multiple generators provided in the dataset (31 in training, 59 in test, with 28 unseen). Our approach has two components: (1) UniXcoder as the encoder with Label-Distribution-Aware Margin (LDAM) loss for handling class imbalance, which provides about a +7 Macro-F1 point improvement over the cross-entropy baseline; and (2) a diverse ensemble of 12 models trained with different objectives and architectures, combined with hard majority voting. Our system achieves 41.28% Macro-F1 on the official test set, placing 9th in the competition. Through our experiments, we find that loss engineering and ensemble diversity matter more than domain generalization techniques such as V-REx, which degraded test performance.

1 Introduction

Large Language Models (LLMs) now generate code that is often indistinguishable from human-written code. This creates practical problems: students submit AI-generated assignments, and AI-written code with unreviewed vulnerabilities enters production systems. Reliably detecting machine-generated code, and identifying which model produced it have become important tasks but remain difficult given the growing number of code generators.

SemEval-2026 Task 13 (Orel et al., 2026b) formalizes this problem as three subtasks, and we focus on **Subtask B**: given a code snippet, classify it as human-written or generated by one of 10 LLM families (DeepSeek, Qwen, 01-ai, BigCode, Gemma, Phi, Meta-LLaMA, IBM-Granite, Mistral, OpenAI). The evaluation metric is Macro-F1, which weights all 11 classes equally regardless of frequency.

What makes this task difficult is the combination of two distribution shifts between the training and test sets: the number of distinct generators nearly doubles from 31 to 59, and the class balance flips (human code drops from 88% to 47% in the labeled test subset).

Our main findings are: (1) margin based loss engineering with LDAM (Label-Distribution-Aware Margin) is the most impactful single technique given our approach, improving the test Macro-F1 score by 7 points; (2) combining models with different inductive biases into a diverse ensemble works better than improving any single model; and (3) explicit domain generalization techniques such as Variance Risk Extrapolation (V-REx), Sharpness-Aware Minimization (SAM), and Stochastic Weight Averaging Densely (SWAD) borrowed from computer vision hurt performance on this task.

All the code is available on our GitHub[1].

2 Related Work

Most work on detecting AI-generated code frames it as a binary classification problem. Nguyen et al. (2024) fine-tuned CodeBERT to flag snippets written by ChatGPT, but their model does not generalize across generators or languages. Suh et al. (2025) surveyed existing detectors and found that none generalize reliably, with the best Abstract Syntax Tree (AST)-based approach reaching 82.55% F1 on their benchmark. Orel et al. (2025a) and Orel et al. (2025b) scaled the problem to multiple languages and generators, showing that cross-lingual and cross-generator transfer remains difficult. Our work focuses on a harder

*Corresponding author.

[1] https://github.com/anildervis/codexa-code-authorship

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3602–3607

variant: 11-class family attribution rather than binary detection.

Family-level attribution is also useful as a downstream tool for systems that depend on LLM-generated code as an intermediate artifact. For example, Site et al. (2025) answer questions over tabular data by prompting open-source LLMs to emit Pandas code, and report that the resulting code-generation behavior varies sharply across model families. A detector of the kind we describe can be plugged into such a pipeline as a post-hoc check: given the generated snippet, our classifier identifies the most likely producing family, which lets the pipeline route to a family-specific repair pass in the style of Jiang et al. (2023), gate execution on a per-family confidence threshold, or attribute failures back to the chosen LLM at evaluation time. The same applies more generally to retrieval-augmented coding assistants (Zhou et al., 2023), where an index drawn from mixed human and AI sources can be reweighted by predicted authorship, and to contamination audits of code-evaluation benchmarks (Jain et al., 2024), where the same classifier flags AI-written test items.

The technical challenges of this attribution task, namely cross-language transfer, learning from imbalanced label distribution, and the train-to-test distribution shift, correspond to three established lines of prior work that we either adopt or explicitly reject. For cross-language transfer, we employ a code encoder: CodeBERT (Feng et al., 2020) pre-trains jointly on code and natural language, and UniXcoder (Guo et al., 2022) extends this with an AST modality, which motivates our choice of encoder (Section 4). For the class imbalance, we adopt LDAM (Cao et al., 2019), which enforces minority-class margins, and we compare it against Focal Loss (Lin et al., 2017) and Class-Balanced loss (Cui et al., 2019).

The third challenge, distribution shift, is where our findings depart from the established methodology. Domain-Adversarial Neural Networks (Ganin et al., 2016) and V-REx (Krueger et al., 2021) are the canonical tools for this setting in computer vision: the former induces domain-invariant features through gradient reversal, while the latter penalizes loss variance across environments. As Section 4.4 reports, neither method improved test performance in our evaluation. The explanation for that is the authorship signals, in which the code are entangled with the very domain features these methods aim to suppress, and that this entanglement is intrinsic to the task rather than an artifact of hyperparameter tuning.

3 Dataset

The AICD Bench dataset (Orel et al., 2026a) provides 500K training samples, 100K validation samples, and 500K blind test samples across 8 programming languages (Java, Python, C#, JavaScript, C++, Go, PHP, C). Table 1 shows the class distribution in the training set and in the 1K labeled test subset provided by the organizers during the development phase. The "Shift" column reports the difference in percentage points between the test and training distributions. The training set is heavily imbalanced, with human code making up 88.42% of samples, while the smallest class (Gemma) accounts for only 0.39%. The test subset reveals a substantial label shift: human code drops to 47.40% and the number of distinct generators increases from 31 to 59, with 28 unseen during training.

Table 1: Class distribution in training and in the 1K labeled test subset provided by the organizers.

Family	Train	Test	Shift
Human	88.42%	47.40%	−41.0
OpenAI	2.16%	21.40%	+19.2
Qwen	1.80%	7.30%	+5.5
Meta-LLaMA	1.64%	6.10%	+4.5
IBM-Granite	1.63%	1.80%	+0.2
Phi	1.16%	5.40%	+4.2
Mistral	0.92%	1.80%	+0.9
DeepSeek	0.83%	2.10%	+1.3
01-ai	0.61%	2.10%	+1.5
BigCode	0.45%	1.00%	+0.6
Gemma	0.39%	3.60%	+3.2

4 System Overview

We build on UniXcoder as our base encoder, train it with margin-based loss to handle the class imbalance, and combine 12 such models into an ensemble via hard voting. Below we describe each component and the approaches that did not work.

4.1 Encoder Architecture

We use UniXcoder-base (Guo et al., 2022) (125M parameters) as the encoder, followed by a 2-layer MLP classifier ($768 \rightarrow 512 \rightarrow 11$) with ReLU and dropout 0.1 between layers, because its representations transfer across programming languages without language specific fine-tuning. Code is fed to the encoder as raw text with no comment

stripping or normalization, so that surface stylistic signal (whitespace, identifier choice, brace placement) is preserved. We tokenize with the UniXcoder BPE tokenizer, truncate to 512 tokens, and dynamically pad each batch.

We also evaluated GraphCodeBERT (Guo et al., 2021), which incorporates data flow graphs, and CodeSage (Zhang et al., 2024). GraphCodeBERT underperformed UniXcoder by 1.3 points on the test set, and CodeSage showed no improvement despite having 2.8× more parameters, suggesting that the bottleneck is generalization and not the model capacity.

4.2 Loss Engineering

LDAM loss enforces class-dependent margins during training:

$$\mathcal{L}_{\text{LDAM}} = -\log \frac{e^{z_y - \Delta_y}}{e^{z_y - \Delta_y} + \sum_{j \neq y} e^{z_j}} \quad (1)$$

where z_j are logits and $\Delta_j = C/n_j^{1/4}$ is the margin for class j with n_j training samples. We choose C such that $\max_j \Delta_j = 0.5$, giving margins from 0.13 (Human, n=442,098) to 0.50 (Gemma, n=1,946). In practice, minority classes get larger margins, which pushes the model to separate rare LLM families more clearly.

Switching from cross-entropy to LDAM improved test Macro-F1 from 32.02% to 39.07% (+7.05 points). Focal Loss reached 37.45% and Class-Balanced loss (with effective-number β=0.9999) 38.27%, both below LDAM.

The original LDAM paper also proposes Deferred Re-Weighting (DRW): training with standard weights for several epochs before switching to class-balanced weights. Our DRW models use LDAM-only for epochs 1–7 and LDAM with class-balanced weights for epochs 8–10. DRW improved validation F1 but overfitted to the training label distribution. Since the test subset shows a substantially different distribution (47% human vs. 88% in training), LDAM without DRW generally produced better test scores.

4.3 Diverse Ensemble via Hard Voting

Our final system combines 12 models through hard voting (majority prediction wins). The models span four categories of training strategies:

- **LDAM variants** (4): UniXcoder models with different loss configurations, DRW schedules, and checkpoint selections
- **Contrastive** (3): Supervised contrastive learning (Khosla et al., 2020) and generator-adversarial training via gradient reversal
- **Architectural** (3): Multi-head (binary + 11-class), multi-task learning, and GraphCodeBERT
- **Feature hybrid** (2): Neural embeddings combined with handcrafted stylometric features via CatBoost (Prokhorenkova et al., 2018), and a character-level CNN

Appendix A lists each model with its individual test score. Hard voting beats weighted voting at every ensemble size we tried (e.g., 41.08% vs. 41.03% Macro-F1 for 10 models), likely because weighted voting fits its weights to the validation distribution, which differs from test. Another pattern worth noting: diversity matters more than individual quality. The full 12-model ensemble outperforms all subsets (Table 3), and adding models with test F1 around 37% still helps the overall score.

4.4 Negative Results

Several techniques we tried looked promising on validation but did not help on the test set. A CharCNN branch trained to capture surface formatting patterns (whitespace, brace styles) generalized poorly to unseen generators when used as part of a single model, although it still contributes as a separate vote in the ensemble. We also tried V-REx combined with SAM (Foret et al., 2021) and Stochastic Weight Averaging Densely (Cha et al., 2021), which are domain generalization techniques from computer vision. This combination scored only 24.53% test Macro-F1, well below our 32.02% cross-entropy baseline, so stripping away domain information also strips away useful signal. Finally, a two-stage pipeline (binary human-vs-machine classification followed by 10-class family attribution) performed worse than direct 11-class prediction because misclassifications in the first stage were irrecoverable in the second stage.

5 Experimental Setup

All models use AdamW with learning rate 2×10^{-5}, weight decay 0.01, a OneCycleLR schedule with 10% warmup, and bfloat16 mixed precision. The maximum sequence length is 512 tokens. Models are trained for 10 epochs with early

stopping (patience 3) on validation Macro-F1, and we select the checkpoint with the highest validation Macro-F1 unless stated otherwise. Training was performed on Google Colab Pro+ with an NVIDIA A100 80GB GPU; a single UniXcoder run takes roughly 25 minutes per epoch. We report Macro-F1 on the official Kaggle leaderboard (500K blind test samples). Validation F1 is computed on the 100K validation split. Per-class scores in Table 4 and the confusion matrix in Figure 1 are computed on the 1K labeled test subset provided by the organizers.

6 Results

6.1 Main Results

Table 2 shows our main results. The biggest gain comes from switching to LDAM loss (+7.05 Macro-F1 points over the baseline), with ensembling adding another +2.2 points on top. V-REx, our attempt at domain generalization, dropped performance sharply.

Table 2: Main results. Test Macro-F1 is from the official Kaggle leaderboard.

System	Val F1	Test F1
UniXcoder + CE	50.11	32.02
+ LDAM	61.75	39.07
+ Focal Loss	59.82	37.45
+ SupCon + GRL	61.50	39.71
+ Multi-head	62.61	38.90
GraphCodeBERT + LDAM	60.09	37.81
V-REx + SAM + SWAD	48.21	24.53
Neural + CatBoost ensemble	64.25	40.12
12-model hard vote	**—**	**41.28**

Every approach we tested shows an 18–24 percentage point gap between validation and test F1. None of our techniques could close it.

6.2 Ensemble Ablation

Table 3 breaks down how ensemble size and voting strategy affect the score. More models help across the board, and hard voting beats weighted voting.

6.3 Per-Class Analysis

Table 4 shows that performance varies widely by class. Human (97.4%) and OpenAI (82.9%) are the only families above 56%, while DeepSeek sits at just 8.5%. We suspect that the OpenAI family scores well because its outputs have a relatively consistent style, whereas the three families DeepSeek, Qwen, 01-ai produce more similar code distributions, leading our classifier to frequently confuse them with one another.

Table 3: Ensemble ablation on the official test set. All scores are Kaggle Macro-F1.

Configuration	Test F1
12 models, hard voting	**41.28**
Top 10, hard voting	41.08
Top 3, hard voting	41.02
Top 10, weighted voting	41.03
Top 5, hard voting	40.80
Single best model	40.57

Table 4: Per-class F1 for the best ensemble on the 1K labeled test subset provided by the organizers during the development phase.

Family	Train %	F1
Human	88.42	0.974
OpenAI	2.16	0.829
Gemma	0.39	0.563
01-ai	0.61	0.545
Qwen	1.80	0.395
Phi	1.16	0.302
Mistral	0.92	0.286
Meta-LLaMA	1.64	0.260
BigCode	0.45	0.222
IBM-Granite	1.63	0.200
DeepSeek	0.83	0.085

6.4 Error Analysis

Figure 1 shows two failure modes. First, the larger LLM minority classes route a substantial fraction of their mass into other LLM classes rather than into Human: Phi (54 samples) splits into OpenAI (16.7%), Qwen (14.8%), and IBM-Granite (11.1%); Meta-LLaMA (61) splits into Qwen (18.0%), IBM-Granite (13.1%), and Mistral (13.1%); DeepSeek (21) routes 47.6% of its mass into 01-ai and Meta-LLaMA. Second, very small classes collapse asymmetrically: BigCode (10) is predicted Human in 50% of cases, while Mistral (18) routes 33.3% to IBM-Granite. Human (100% correct on this subset) and OpenAI (80.4%) anchor the diagonal; the remaining nine classes share the off-diagonal mass. Beyond these clusters, the 18–24 percentage point gap between validation and test Macro-F1 appears in every configuration we tried. We attribute it to two sources: *label shift*, where human code drops from 88% to 47% in the labeled test subset, and *concept shift*, where 28 unseen generators introduce coding patterns absent from training.

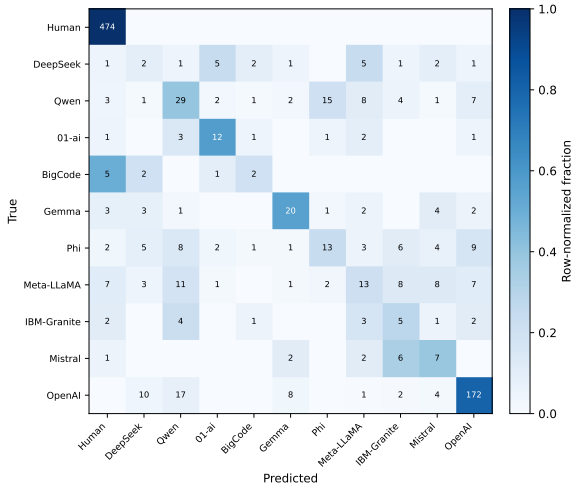

Figure 1: Row-normalized confusion matrix of our 12-model hard-vote ensemble on the 1K labeled test subset released by the organizers. Cell labels show raw counts; shading shows the row-normalized fraction. Class names follow Table 1.

7 Conclusion

Our system for SemEval-2026 Task 13, Subtask B reaches 41.28% Macro-F1 (9th place) by combining LDAM loss with a 12-model hard voting ensemble. Domain generalization methods from computer vision reduced performance, suggesting code authorship detection needs its own generalization strategies. The roughly 20-point gap between validation and test Macro-F1 scores remains unsolved. Going forward, we want to investigate test-time adaptation and pre-training corpora that better cover the diversity of LLM-generated code, since our experiments with CodeSage indicate that scaling parameters alone is unlikely to close the validation-to-test gap.

References

David Álvarez Fidalgo and Francisco Ortín Soler. 2025. CLAVE: A deep learning model for source code authorship verification with contrastive learning and transformer encoders. *Information Processing & Management*, 62(3):104005.

Kaidi Cao, Colin Wei, Adrien Gaidon, Nikos Arechiga, and Tengyu Ma. 2019. Learning imbalanced datasets with label-distribution-aware margin loss. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Junbum Cha, Sanghyuk Chun, Kyungjae Lee, Han-Cheol Cho, Seunghyun Park, Yunsung Lee, and Sungrae Park. 2021. SWAD: Domain generalization by seeking flat minima. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. Sharpness-aware minimization for efficiently improving generalization. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35.

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)*. Association for Computational Linguistics.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training code representations with data flow. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of code language models on automated program repair. In *Proceedings of the IEEE/ACM 45th International Conference on Software Engineering (ICSE)*.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2020. Supervised contrastive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Dinghuai Zhang, Remi Le Priol, and Aaron Courville. 2021. Out-of-distribution generalization via risk extrapolation (REx). In *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

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 (ICCV)*.

Phuong T. Nguyen, Juri Di Rocco, Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Massimiliano Di Penta. 2024. GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT. *Journal of Systems and Software*, 214:112059.

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*. 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*. Association for Computational Linguistics.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Atakan Site, Emre Erdemir, and Gülşen Eryiğit. 2025. ITUNLP at SemEval-2025 task 8: Question-answering over tabular data: A zero-shot approach using LLM-driven code generation. In *Proceedings of the 19th International Workshop on Semantic Evaluation (SemEval-2025)*, pages 1504–1514, Vienna, Austria. Association for Computational Linguistics.

Hyunjae Suh, Mahan Tafreshipour, Jiawei Li, Adithya Bhattiprolu, and Iftekhar Ahmed. 2025. An empirical study on automatically detecting AI-generated source code: How far are we? In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE)*.

Dejiao Zhang, Wasi Uddin Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. 2024. Code representation learning at scale. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Shuyan Zhou, Uri Alon, Frank F. Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2023. DocPrompting: Generating code by retrieving the docs. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*.

A Ensemble Model Details

Table 5 lists all 12 models in our final ensemble, grouped by the four training strategies described in Section 4. All models use UniXcoder-base as the encoder unless otherwise noted.

Table 5: The 12 models in our final ensemble. Test Macro-F1 is from the official Kaggle leaderboard.

Strategy	Description	F1
LDAM	LDAM loss, best checkpoint selection	40.57
LDAM	LDAM loss, alternative checkpoint	40.15
LDAM	LDAM loss with deferred reweighting	39.07
LDAM	LDAM loss, different random seed	36.57
Contrastive	SupCon loss + generator-adversarial gradient reversal	39.71
Contrastive	Generator-adversarial training with gradient reversal only	39.28
Contrastive	CLAVE (Álvarez Fidalgo and Ortín Soler, 2025): contrastive learning with augmented views	37.30
Architect.	Multi-head: shared encoder with binary + 11-class heads	38.90
Architect.	Multi-task: joint detection and family attribution	38.79
Architect.	GraphCodeBERT encoder + LDAM loss	37.81
Feature	CatBoost meta-classifier over UniXcoder embeddings + stylometric features	40.12
Feature	Character-level CNN capturing formatting patterns	38.50