Farhan Nafis Rayhan at SemEval-2026 Task 13: Supervised Contrastive Learning Approach with Gated Multiclass Decomposition Ensemble Architecture for Code Authorship Identification

Farhan Nafis Rayhan and Fariska Zakhralativa Ruskanda
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
farhannafis281004@gmail.com, fariska.zr@staff.stei.itb.ac.id

Abstract

This paper present our submission for SemEval-2026 Task 13 Subtask B, which requires the multi-class attribution of code snippets across 10 distinct AI generator families and a human baseline. Our proposed system utilizes a three-stage ensemble architecture specifically designed to navigate extreme class imbalance and capture subtle stylometric fingerprints. Initially, we employ Supervised Contrastive Learning to fine-tune a UniXcoder and ModernBERT backbone. Resulting embeddings are then processed by five heterogeneous shallow experts, each utilizing a multiclass decomposition to master specific generator lineages through specialized architectures. A Human Shield acts as a hierarchical safety auditor as an aggressive binary layer of human vs machine. Finally, a Context-Aware Gated Meta-Learner dynamically aggregates these expert opinions into a final predictions. Our experiments reveal that streamlining the system to a pure UniXcoder backbone fine-tuned with supervised contrastive learning improves performance, outclassing the official CodeBERT baseline with a final Macro-F1 score of 0.31389, ranking 26th overall.

1 Introduction

In the past few years, we had witnessed the rapid growth of Large Language Models (LLM), which transitioned from a research prototype into a publicly accessible text generation model. These state of the art models shown an incredible ability at high quality source code generation, it only takes a small matter of time until LLMs are integrated as software development tool. A widespread adaptation of code generating machines essentially offered many proposed values for a software developer's productivity, significantly reducing time taken to solve a variety of programming tasks (Noy and Zhang, 2023). In actuality, the probabilistic nature of LLMs implies that machine generated code might bear an unrealized bug, compromising the safety of said code with high-risk security vulnerabilities (Pearce et al., 2022). Need for differentiating between a human written code with a machine generated one is crucial for a future with safe and original software ecosystems.

Generalization is still the main complexity of attributing source code's authorship. (Li et al., 2023). The development of LLMs is currently more rapid than researches on machine generation identification strategies. This leads into an arms race, where a detector reliable at flagging an AI model would soon be irrelevant once a more sophisticated model is released. Another problem arrives from variety of programming languages out there. Detectors only trained on a single programming language might perform very well, yet failing completely once given a snippet of other languages with completely different structure and syntaxes. With the advancement of machine's ability to imitate human's stylometry, building a robust code authorship system has only proven to be a grueling task.

Task 13 at SemEval 2026 is formed to bridge the gap of these complexities (Orel et al., 2026). This shared task aims at building a system to detect machine generated code with an emphasis of generalization on cross-language, cross-generation, and cross-domains. To be precise, this task is structured as 3 different subtasks (subtask A, subtask B, and subtask C). We decided to focus on subtask B only, which not only to identify human written code snippets, but a multi-class prediction of which family generated it. Specifically the objective is to predict which of these 10 different LLM families the model was a part of, including DeepSeek-AI, Qwen, 01-ai, BigCode, Gemma, Phi, Meta-LLaMA, IBM-Granite, Mistral, and OpenAI. It stresses the importance of a deeper understanding of each LLM families consistent stylometric fingerprints, such it is possible to detect codes from unseen models of the same families.

This task uses a robust framework based on the

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 3485–3494
July 3-4, 2026

task proposer's previous work (notably Orel et al., 2025b & Orel et al., 2025a). For subtask B, the dataset consists of code snippets in 8 different languages (Python, C, C++, C#, PHP, Go, Java, Javascript) across 3 domains (Algorithmic, Production, and Research). It is given as splits of 500.000 rows train data and 100.000 rows of validation data. Each code snippet is provided with it's programming language and generator model's name. As for evaluation, 500.000 rows of test data is given, without the language feature supplied. The training dataset is heavily imbalanced with 88.42% of the whole dataset are labeled Human, while the least frequent class is Gemma with only 0.39% of instances. Macro-F1 score is used as the official evaluation metric to battle this imbalance.

In this paper, we aimed for a competitive performance under the tight constraint of low costs computing resources. Our proposed solution is a three-stage ensemble consisting of 8 different AI models. The system includes 2 LLMs trained with supervised contrastive learning, 5 heterogenous shallow learning models, and a meta learner. This approach minimizes the amount of LLM involved, which are a substantial bottleneck with respect to training time and computing load.

Every fragment of the code that could be considered as human-like languages (such as comments, string values, and identifiers), are formatted as a long space separated string and fed into ModernBERT (Warner et al., 2024). On the other hand, the code structure, sanitized from the human-like languages, were fed into UniXcoder (Guo et al., 2022). Both encoders were fine-tuned using Supervised Contrastive Learning (SupCon, from the work of Khosla et al., 2021).

The resulting embeddings are processed and partitioned by shallow models to drive each model becoming an expert on a distinct class decomposition. These decompositions are based on families known to have similar stylometric patterns. Among the 4 shallow models used are Residual Multi-Layer Perceptron, Attentive Zero-Shot Prototypical Head, Cat Boost Classifier, and Attentive Few-Shot Prototypical Head. (He et al., 2015; Snell et al., 2017; Prokhorenkova et al., 2019; Gao et al., 2019). The last shallow model is a binary shielding mechanism specifically to flag human written codes, implemented with XGBoost (Chen and Guestrin, 2016).

Final stage of the ensemble is a Context-Aware Gated Meta-Learner (Jacobs et al., 1991) that performs dynamic opinion aggregation through a multi-layered decision process. Beginning with an enforcement of human labeling for samples with high confidence on the binary shielding. Continued with a neural gating network to weigh each of the 4 shallow models, and ends with a learnable temperature scaling calibration (Arad and Rosset, 2025). Aggregating the whole ensemble into a probability of distribution where the prediction would be inferred.

The following are key observations based on our experiments:

1. Transitioning to Contrastive Learning from Cross-Entropy Loss gave a significant improvement in Macro F1-Score.

2. Adding a small amount of extra machine samples of every class into each shallow model's dataset did yield a small increase in performance.

3. ModernBERT didn't learn very well from the human-like language feature served in the proposed format and it acted as noise instead.

4. Our proposed system beats the given CodeBERT baseline but less robust compared to other teams.

2 Related Works

Over time, methodologies for detecting machine-generated code have evolved significantly. Recent research has shifted from simple *Cross-Entropy*-based classification toward representation learning via *Contrastive Learning*. Unlike traditional methods that train models to map text to discrete labels, *Contrastive Learning* compels the model to learn an *embedding* space where samples from the same generator are pulled closer together, while those from different generators are pushed apart. By learning the relative distances between samples, models become more robust in identifying new generators unseen during training. Consequently, this approach can improve attribution accuracy by up to 6.8% compared to standard *cross-entropy*-based fine-tuning.

The *CodeGPTSensor* methodology proposes the use of the *Contrastive Learning* paradigm as a primary strategy to map the subtle differences between a machine's *statistical fingerprint* and unique human writing styles (Xu et al., 2025). By leveraging *UniXcoder* as a *semantic encoder*, this methodology captures deep structural and contextual rela-

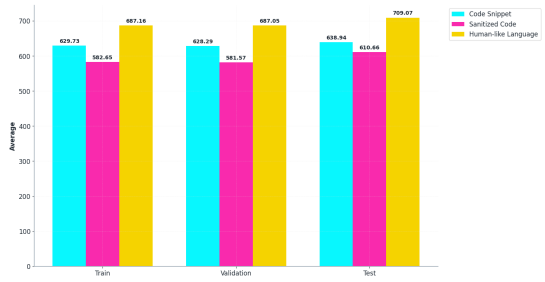

Figure 1: Average Token Count Before and After Feature Engineering

[COMMENTS]
< c1 >
< c2 >
⋮
[STRINGS]
< s1 >
< s2 >
⋮
[IDENTIFIERS]
< id1 > < id2 > < id3 > . . .
[SEP]

Figure 2: Human-like language representation format.

tionships within the source code, thereby surpassing simple lexical analysis. The system was validated on *HMCorp*, a massive corpus comprising 550,000 Python and Java code pairs, providing a robust foundation for precisely identifying generative patterns left by artificial intelligence.

Empirically, this research reveals that machine-generated code tends toward simpler, more repetitive structures and frequently leaves artifacts such as unused constructs or *hardcoded* debugging remnants. In contrast, human-written code exhibits significantly higher structural complexity and stylometric variation, reflecting the more dynamic reasoning of developers. The integration of *Contrastive Learning* and semantic *encoders* has proven highly effective in capturing these generative *stylometric fingerprints*. As a result, this method provides a more robust solution for maintaining the integrity of originality within the modern software development ecosystem.

3 System Overview

3.1 Feature Engineering

We devised 2 new features for each code snippet in purpose of increasing the amount of relevant tokens able to be fed into each LLM. Initially, the average token count of training data are 629.73, which is longer than 512 the Sequence Length possible for both LLMs' base version to train on (under computing power constraints). This prompts the need for truncating the input such that it abides the maximum sequence length, which risks the loss of important stylometric artifacts located later inside the code snippet. Thus, by splitting the whole code snippet into 2 different features we avoided less relevant tokens to the model, making it learn more from tokens it's supposed to learn from.

We define human-like language parts of the code snippet to be those that are not syntactic and would realistically be varying if written by humans. These category includes comments, documentation, string literals, and identifiers, which are then formatted as a long string, and became our first feature. This format allows the model to know what each token is supposed to represent, especially if one part is missing from the code snippet. Identifiers are normalized by removing it's case specific characters (e.g. underlines on snake case) such it's easier to be learned by general LLMs such as ModernBERT.

For the second feature, we removed every instances of human-like language tokens, leaving only a sanitized code full of syntactic tokens and structure. The syntax for comments and string literals (such as '#' in python) are kept, followed by special tokens which are <COMMENT> and <STRING> respectively. This is done to preserve the syntactic validity of the original code snippet.

3.2 SupCon on LLMs

Previously, our experiment using Weighted Cross-Entropy didn't work out too well. A naive implementation of several code based encoders (including UniXcoder) only yields a Macro-F1 of 0.08 on the public test set. This, including the heavy class imbalance prompted us to transition into Contrastive Learning approach. The sanitized code feature is fed into the UniXcoder-base model, while the human-like language feature is fed into the ModernBERT-base model. The main output for both model is a vector embedding of size 768, which are then projected into a contrastive space with dimension 128 by a 2 layer MLP, called the projection head. This head is what we optimize using SupCon loss. Once both model encodes the code snippet into two 768-D embedding vectors in the contrastive space, they are concatenated as a 1536-D vector, the output of this stage of our

ensemble system.

3.3 Multiclass Decomposition

Inspired from leading team on SemEval 2024 Task 8 for Subtask B (Gu and Meng, 2024), Our approach is to create a One-vs-Some Decomposition, unlike most standard decompositions such as One-vs-All or One-vs-One. Here on the second stage of our ensemble architecture, we decompose the 10 machine classes into 4 different groups, where each group consists of generator families with similar code stylometry based mostly on our own judgments. Then for each group, we pick a shallow model that serves as a local specialist, trained specifically to be an expert at distinguishing its assigned generator families from the human-authored baseline. By focusing on a subset of classes, each model can more effectively capture the fine-grained stylometric profile unique to it's specific AI family groups. To be precise, here are the groups we formed:

3.3.1 Group 1: OpenAI & Meta-LLama

The code of these models exhibits high polish characterized by extensive, college-level explanatory comments and highly descriptive, standardized variable naming (Jaashan and Bin-Hady, 2025). Although OpenAI-generated code often features longer average sentence lengths in documentation, Llama follows prompt instructions with a rigid, predictable syntax that results in a uniform stylistic signature. These models are trained on massive, curated datasets, leading to a perfected coding style that avoids idiosyncratic human-like messiness. **Residual Multi-Layer Perceptron** is used since it excels at modeling the higher capacity feature space where these 2 models reside, effectively disentangling semantic meaning from the subtle stylistic polish. Choosing 4 layers helps learning where the boundaries between these models and high-quality human code are quite thin. The Residual Blocks are key here, preventing the stylistic signal from washing out across deeper layers.

3.3.2 Group 2: Qwen, IBM-Granite, 01-ai

This group is distinguished by unique characteristics in variable naming and comment structures, heavily influenced by their training on vast, diverse multilingual corpora. They demonstrate a higher density of varied naming conventions and localized comment patterns compared to Western-centric models. Their stylometry is defined by this cross-linguistic diversity, making their outputs more varied in how they describe logic and document code blocks. **Attentive Zero-Shot Prototypical Head** is more robust to the cloud-like distribution of these multilingual model, where code stylometry varies wildly depending on the linguistic influence. By fitting a memory buffer, instead of the usual learning examples, with real samples via PCA-lowrank initialization, the model creates a geometric anchor for each multilingual cluster. The Multi-Head Attention allows the model to dynamically attend to the most relevant prototypes in memory, essentially filtering out the linguistic noise and focusing on the core structural similarities shared by these multilingual models.

3.3.3 Group 3: Deepseek & BigCode

These models act as an algorithmic specialists, with DeepSeek specifically showing a high efficiency in complex recursive functions and structural artifacts within recursive logic. Meanwhile, BigCode models exhibit high fidelity to existing GitHub patterns, frequently incorporating specific boilerplate code and standard open-source naming conventions. Their tokenization is optimized for syntax trees, allowing them to maintain strict variable and type consistency across long, logic-heavy contexts. **CatBoost** is chosen here due to it's ability at finding hard structural rules. The embeddings from the first stage can be treated as a tabular-like data, where GDBT excels on. This model tends to be better at identifying precise, rule-based artifacts that a deep learner might over-smooth into a general semantic vector.

3.3.4 Group 4: Gemma, Phi, Mistral

Smaller models tend to produce minimalist, highly functional code with significantly lower stylometric variance than their larger counterparts. Their profile often manifests as thin probability distributions on control structures, reflecting a more constrained and direct path to solving a coding task. Because they focus on core reasoning and code generation with less fluff, their outputs often have the highest overlap with human-authored code, making them the most challenging for standard detectors (Aydin et al., 2025). The **Attentive Few-Shot Prototypical Head** maximizes the detection sensitivity for low-variance models, ensuring that the final ensemble can attribute even the most quiet stylometric signals. Trained on episodic batches, the model learns to sharpen its focus on the few samples available,

effectively heightening the Minority Precision.

3.4 Human Shield

This component is technically a One-vs-All decomposition using shallow model just like the second stage of our ensemble, but it also provides hierarchical override to the next stage. Human Shield mainly acts as binary shielding mechanism, a final safety auditor that protects the dense manifold of human written code from being misattributed as machine's, agresively reducing False-positive rate. Utilizing an **XGBoost** model trained on the full human distribution, contrasted with a small sample of machines, to calculate a humanity score based on idiosyncratic fingerprints like messy logic and inconsistent indentation. If this score exceeds a defined threshold, the aggregator triggers a hard bypass that forces the final output to human class. Human data in a SupCon embedding space usually forms a massive, dense core, in which GBDTs will excel at carving out the complex, jagged boundaries of this dense human core.

3.5 Meta-Classifier

We used **Context-Aware Gated Meta-Learner**, which features a dynamic Mixture of Experts (MoE) framework designed to handle the expert disagreement in the heterogeneous ensemble. It operates through a hierarchical two-level decision logic: the first level is a Binary Shielding mechanism driven the human shield. Enforcing the class to be human if the confidence is high enough. For instances that pass the human shield, the Gating Network performs a sophisticated weighted fusion by simultaneously evaluating expert opinions and global code context. Compress the 1536-D embeddings input into a 128-D context vector, it's then concatenated with refined meta-features, including both class probabilities and Shannon entropy (H), from all five experts. This gating mechanism calculates dynamic trust scores (α) that allow the meta-learner to shift its reliance, weighing each five experts differently based on context (Zhang et al., 2025).

To ensure the final output is statistically reliable, the system concludes with a Calibration layer that applies learnable Temperature Scaling (T), optimized via L-BFGS to sharpen or soften the distribution based on a held-out calibration set. NA-FIR addresses a fundamental flaw in traditional isotonic regression, which is the inability to naturally produce probabilities that sum to one in multiclass settings. By incorporating normalization directly into the optimization process, NA-FIR consistently improves both negative log-likelihood (NLL) and calibration error across diverse datasets. This strategy is particularly effective for our ensemble since it accounts unnormalized outputs of diverse base learners.

4 Experimental Setup

4.1 Feature Engineering Pipeline

We implemented the lexical and structural extraction pipeline using **ANTLR4** (Parr, 2013), which provides parsers for all programming languages in the dataset. For each language, we specified token types of interests, allowing the extraction logic to operate uniformly across languages while still respecting language-specific token definitions. Some languages (e.g. Python, JavaScript, Go) place comments in hidden lexer, meaning we must explicitly check tokens from the relevant hidden channels during extraction. String literals are extracted after removing surrounding delimiters, while Identifiers are collected and subsequently split into smaller components. Simultaneously, we construct a sanitized version of the source code. Each detected comment or string literal is replaced with a placeholder that preserves the original syntactic delimiters. For the full test dataset, where programming languages feature wasn't included, we created another pipeline for language labeling. The full dataset with these engineered features are publicly available at (Rayhan, 2026)

4.2 SupCon Training

Supervised contrastive learning requires multiple samples from the same class within a batch in order to construct positive pairs. To guarantee this property, we implement a custom batch sampler that ensures each batch contains same number of samples for each class. To reduce the number of trainable parameters while maintaining adaptation capacity, we apply LoRa using the PEFT library (Mangrulkar et al., 2022), injecting modules into the attention projection matrices.

The projection head maps the 768-dimensional pooled representation into a 128-D contrastive embedding space using $768 \rightarrow 2048 \rightarrow 128$ architecture with Batch Normalization and ReLU activation applied between the two linear layers. The resulting embeddings are L2-normalized so that all representations lie on a unit hypersphere, which

Parameter	UniXcoder	ModernBERT
Learning Rate	$5e^{-5}$	$3e^{-4}$
Epochs	8	20
Batch Size	352	616
Lora Rank	32	28
Lora Alpha	64	56

Table 1: SupCon Fine-tuning Hyperparameters used

stabilizes similarity comparisons during contrastive learning. We trained this head using SupCon loss of temperature 0.1.

We evaluate the learned representations using a k-nearest neighbor classifier. After each training epoch, we construct a reference feature bank consisting of balanced embeddings sampled from the training set. For each validation sample, cosine similarity is computed between its embedding and all reference embeddings. The predicted label is determined via majority voting among the $k = 20$ nearest neighbors.

Our UniXcoder achieved validation performance Macro-F1 of **0.3971**, on the other hand ModernBERT converges on a staggeringly low Macro-F1 of **0.1164**. Projection head is removed after fine-tuning, such both models output the 768-D encoding of the code snippet, concatenated as the input for the next stage.

4.3 Shallow Model Trainings

Each of the 4 models are trained with a different $\frac{1}{4}$ split of human classes combined with every machine class it's focusing to learn. This is done for the sake of reducing fused embedding data size fed to each model (approximately 25% each), while enough human classes as contrasts in each split.

The Residual MLP consists of Residual Blocks architecture ($1024 \to 768 \to 512 \to 256$) with LeakyReLU and Batch Normalization to prevent stylistic signal washout. Key hyperparameters include an AdamW optimizer with a low learning rate (1×10^{-5}) and Focal Loss ($\gamma = 2.3$) with specific class weighting. Training progressed steadily, reaching a peak Macro-F1 of 0.8485 at epoch 15 before early stopping with patience of 5 epochs triggered as the validation loss stabilized at **0.04**.

Our Zero-shot Attentive Prototypical head replaced the usual static Euclidean centroids with a Memory Buffer ($k = 100$ per class). We initialize the projection layer using GPU-accelerated PCA-lowrank to align the geometry of the embedding space before running 10 refinement epochs. The model uses a Loss function calculated as $0.3 \times$ Distance-Scaled Contrastive Loss $+ 0.7 \times$ contrastive loss. We also use a learnable temperature parameter (T), used to scale squared euclidean distances, which went from 0.0709 down to approximately 0.0518 during training. The progression was highly efficient, achieving a final Accuracy of 94.48% and a Macro-F1 of **0.7280**.

Our CatBoost utilized the MultiClassOneVsAll loss function to effectively isolate the rigid, recursive artifacts characteristic of the machine families. Key hyperparameters involve a Depth of 8, a learning rate of 0.05, and a heavy L2-leaf regularization (8.0) to maintain generalization in the logic-heavy feature space. Training was conducted on the GPU, monitoring TotalF1 as the evaluation metric to balance precision and recall for minority machine classes. The model converged rapidly, reaching its best iteration at 515 with a BestTest TotalF1 of **0.9713**.

The Attentive Few-Shot Prototypical Head that uses an Episodic Training deviates from the foundational architecture by implementing Dynamic Episodic Sampling (varying 3–7 shots) and Support Jittering ($\sigma = 0.001$) to simulate the style variance found in human-adjacent code. We utilize a 4-head attention mechanism and an RAdam optimizer (learning rate: 7×10^{-5}) to minimize the simplicity bias"that often leads detectors to miss minimal AI patterns. The training progression followed a meta-learning curve, where performance was validated every 500 episodes using Macro-F1. The model peaked at a Macro-F1 of **0.8206** at episode 1500, demonstrating that episodic refinement is essential for catching models with low stylometric variance.

4.4 Human Shield Training

We took the fused embedding data of every human class and combine them with 1500 instances from each classes, resulting in a data size of approximately 455K rows. Our XGBoost deviates from standard binary classifiers by using the DART booster (Dropouts meet Multiple Additive Regression Trees) and a massive scale_pos_weight of 7.3 to protect the 88% human class. We configured the model with a Max Depth of 9, a 0.05 learning rate, and a high early stopping threshold of 800 rounds to ensure it masters the dense manifold of human variability. The evaluation was based on a custom XGBoost-Macro-F1 metric, where it reached

Hyperparameter	Gating Network	Calibration Layer
Optimizer	AdamW (lr: 5e-4)	L-BFGS (lr: 0.01)
Regularization	Expert Dropout (p: 0.05)	T-clamp (0.01 - 5.0)
Objective	Focal Loss ($\gamma = 2.0$)	NLL Minimization
Result	Trust Weights (α)	Scaled Probabilities

Table 2: Hyperparameters for Third Stage of Ensemble Architecture

0.9384 by iteration 700.

4.5 Meta-Learner Training

Gated Meta-Classifier and Calibrated Aggregator are trained last, stacking from every shallow model and human shield. The training data used are the whole validation set splitted into 2, half for the gating network training and the other for calibration set.

Our Gating Network utilizes a Residual Compressor featuring a linear bottleneck, Batch Normalization, and a GELU activation to project the 1536-D fused embeddings into a compact 128-D context vector, using architecture $(1536 \to 512 \to 128)$. Simultaneously, an Expert Refiner processes a 60-D meta-feature vector, comprising class probabilities and Shannon entropy (H) from all five shallow models, through a linear layer and Layer Normalization to extract inter-expert correlations as a vector of 256-D. These two streams are concatenated and fed into the final Decision Gate, a multilayer perceptron of architecture $(384 \to 128 \to 5)$ that outputs a softmax-normalized trust distribution $(\alpha_1 \dots \alpha_5)$. During the Binary Shielding logic, where in the case of if the humanity score from the shield exceeds a threshold of **0.9**, the system triggers a hard bypass to Class 0 (Human).

The training progression of the aggregator was highly stable, utilizing AdamW with a weight decay of 0.06 to prevent the gate from over-relying on any single specialist early in the process. The model reached convergence around epoch 150, maintaining a low and steady Focal Loss of approximately **0.0540**.

We also implemented a CalibratedAggregator wrapper that applies Temperature Scaling to the raw aggregated logits. What it does it Optimizing the scalar parameter T using the L-BFGS algorithm over 50 iterations, targeting the Negative Log-Likelihood on a dedicated 10% held-out calibration set. The resulting optimal temperature of $T =$ **1.1192**.

5 Results

Our implemented system results in a Macro-F1 of **0.29639** using the full test dataset. This places our team on **rank 26** on the official public scoreboard, where we passed the CodeBERT baseline. Further experimentation after contest shown that the system can be only be fine-tuned into a Macro-F1 of **0.31389**, which might mean the architecture itself is handicapped, failing to completely generalizing to the axes of multi-generator.

Before the deadline, we fine-tuned the humanity threshold for the perfect shielding enforcement. It is found that increasing the threshold reduces the human false-postives made by the system, but an overly aggresive threshold also introduces true-negatives intead. We tested thresholds ranging from 0.75-0.95, where we found that 0.90 is the sweetspot, although the overall change to the Macro-F1 is only 0.013 at most.

Analyzing the confusion matrix of the submitted system, it's revealed that while our Hierarchical Gating architecture effectively navigates the 88.4% human prior, the stylistic boundaries between certain AI families remain riddled. The most prominent success is the Huaman Shield which correctly attributed 87340 human samples, maintaining a near pristine column with minimal machine leakage. This validates our strategy of using a dedicated binary auditor to protect the human code snippets before the machine experts attempt a fine-grained prediction.

However, a clusters of confusion emerges within the machine classes that mirror our initial stylometric judgments. Frist group shown significant mutual interference, with 298 OpenAI samples mislabeled as Meta-LLaMA, likely due to their shared high polish unsuccesfully differentiated. Similarly, the second group exhibist high overlap; Qwen captured over 1000 samples, but 01-ai was frequently misattributed to it, suggesting that shared multilingual pre-training corpora create semantic clouds that are difficult to disentangle even with attentive metric learning. On the other hand, fourth group is relatively more separated than other groups, proven by it's intra-group confusion to be relatively less. But this group struggles more on creating boundaries with human samples, increasing the False-positives of humans.

When comparing expert performance, Residual MLP emerged as the volume leader, successfully identifying 1233 OpenAI instances, while

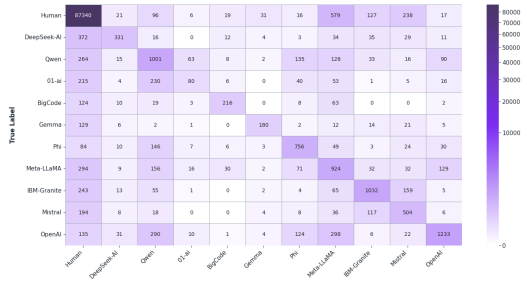

Figure 3: Confusion Matrix of Submitted System

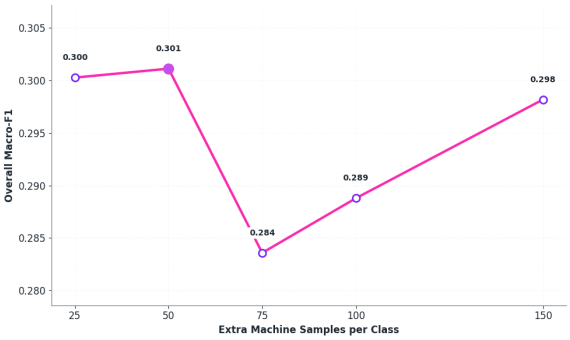

Figure 4: Results of adding extra machine samples to each split

CatBoost proved its worth as a structural specialist, keeping the DeepSeek-AI and BigCode columns relatively clean from cross-family noise. Ultimately, the results demonstrate that our Gated Aggregator successfully mitigates expert disagreement, ensuring that the system relies on the most relevant specialist for a given code context.

After the competition ended, we experimented on altering the data feeded into shallow models. State of the art researches on Contrastive Learning shown that adding small noises results in the model defining the boundary between classes better. Thus, we decided to add a small sample of irrelevant machine classes into the 4 data split, meaning for every split we also include a same amount of data from every classes other than it's own decomposition. We tested on the variability of 25-150 random samples for each class, attributing into 250-1500 extra samples on every split.

Furthermore, we experimented with altering the first stage of our ensemble. Since ModernBERT showing a meagre Macro-F1, one might suspect that model contributed noise rather than increasing the ensemble's performance. This is also proven by UniXcoder's Macro-F1 which initially, ensembled in this system, already achieved higher than the resulting Macro-F1 of the overall ensemble

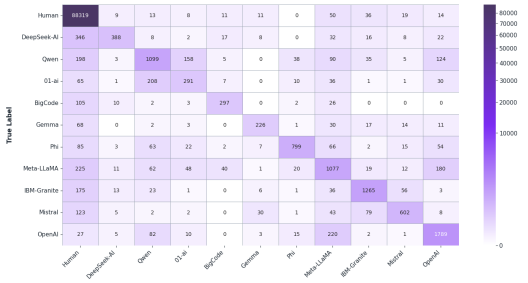

Figure 5: Confusion Matrix of ModernBERT Removal

model. It can be reasoned as the effect of human-like language feature formatting, possibly it was too abstracted for the transformer to learn meaningful signals. Therefore, we removed ModernBERT completely from the system, meaning our shallow model only use a 768-D embedding instead. Surprisingly, this consistently gives a better result of up to 0.013 increase in Macro-F1. With fine-tuning on every shallow model, aiming to enhance the boundaries of each group, we arrived on our best performing system yet evaluated at **0.31389**.

The refined system demonstrates a marked improvement in attribution, particularly within the most challenging confusion clusters identified in the submitted model. By streamlining the input features to a singular 768-D UniXcoder embedding and executing a comprehensive fine-tuning of all shallow models, the architecture significantly sharpened its class boundaries. The most significant progress is observed in the first group, where OpenAI correct attributions surged from 1233 to 1789, and Meta-LLaMA increased from 924 to 1077, effectively disentangling their shared high polish stylometry. This shift suggests that focusing on a specialized code-centric encoder like UniXcoder, rather than a dual-hybrid approach, better preserves the latent signal necessary for cross-generator generalization.

Furthermore, the human vs. machine separation reached a near-perfect state, with the Human Shield correctly identifying 88319 instances, nearly eliminating the machine leakage observed in earlier iterations. The strategic inclusion of additional machine samples and the subsequent fine-tuning also revitalized the Prototypical heads; for example, 01-ai detections jumped from a negligible 80 to 291, and Phi attributions rose to 799. These quantitative gains confirm that the streamlined architecture is far more robust against the system we submitted.

6 Conclusions

Our study provides a specialized, multi-class decomposition architecture that's elevated by contrastive learning, for a more robust AI-code attribution that successfully navigates the high-stakes imbalance and stylistic overlap of the SemEval Task 13 challenge. A key technical takeaway was that transitioning from traditional Cross-Entropy Loss to Supervised Contrastive Learning provided a significant boost in the generalization, as it allowed the experts to more effectively cluster the subtle authorship profile left by different AI families. We also confirmed that even modest data addition across machine classes significantly stabilized the expert's ability to catch more subtle signals. Crucially, our experiments revealed that ModernBERT struggled to extract meaningful signals from the provided code format and instead acted as feature noise; consequently, streamlining the system to a pure UniXcoder backbone proved to be the most robust path for fine-grained discrimination. While our gated ensemble, fortified by a dedicated Human Shield, comfortably outclassed the official CodeBERT baseline, the competition results suggest that a gap remains when compared to other top-tier teams. Nevertheless, this study demonstrates that partitioning the label space into specialized expert groups is an essential strategy for identifying the human soul within messy code while accurately attributing the increasingly polished outputs of modern machine generators.

Limitations

The main limitation of our study is the lack of better computational resources. The proposed pipeline takes a considerable amount of time to be executed, meaning experiments could done is quite limited in our time frame. Limitation of a better resource also led to many idealistic compromises during our design, albeit we're proud of what were realized.

Acknowledgments

This research was supported by computational resources provided by Institut Teknologi Bandung, more precisely Sekolah Teknik Elektro dan Informatika.

References

Alon Arad and Saharon Rosset. 2025. Improving multi-class calibration through normalization-aware isotonic techniques. *Preprint*, arXiv:2512.09054.

Omer Aydin, Enis Karaarslan, Fatih Safa Erenay, and Nebojsa Bacanin. 2025. Generative ai in academic writing: A comparison of deepseek, qwen, chatgpt, gemini, llama, mistral, and gemma. *Preprint*, arXiv:2503.04765.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 785–794. ACM.

Tianyu Gao, Xu Han, Zhiyuan Liu, and Maosong Sun. 2019. Hybrid attention-based prototypical networks for noisy few-shot relation classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:6407–6414.

Renhua Gu and Xiangfeng Meng. 2024. AISPACE at SemEval-2024 task 8: A class-balanced soft-voting system for detecting multi-generator machine-generated text. In *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 1476–1481, Mexico City, Mexico. Association for Computational Linguistics.

Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *Preprint*, arXiv:2203.03850.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *Preprint*, arXiv:1512.03385.

Hasan M. S. Jaashan and Wagdi Rashad Ali Bin-Hady. 2025. Stylometric analysis of ai-generated texts: a comparative study of chatgpt and deepseek. *Cogent Arts & Humanities*, 12(1):2553162.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2021. Supervised contrastive learning. *Preprint*, arXiv:2004.11362.

Haoran Li, Yufan Guo, and Ruibo He. 2023. On the generalization of machine-generated text detectors: A survey and new perspectives. *arXiv preprint arXiv:2312.03668*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, Benjamin Bossan, and Marian Tietz. 2022. PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Shakked Noy and Whitney Zhang. 2023. Experimental evidence on the productivity effects of generative artificial intelligence. *Science*, 381(6654):187–192.

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. 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.

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.

Terence Parr. 2013. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf.

Henry Pearce, Benjamin Ahmad, Benjamin McKenzie, and Dawn Song Be-Ac. 2022. Asleep at the keyboard? assessing the security risks of ai code generators. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2019. Catboost: unbiased boosting with categorical features. *Preprint*, arXiv:1706.09516.

Farhan Nafis Rayhan. 2026. Semeval-2026 task 13 feature dataset. `Farhannr28/NL-Separated-SemEval-2026-Task13`. Hugging Face dataset.

Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical networks for few-shot learning. *Preprint*, arXiv:1703.05175.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *Preprint*, arXiv:2412.13663.

Xiaodan Xu, Chao Ni, Xinrong Guo, Shaoxuan Liu, Xiaoya Wang, Kui Liu, and Xiaohu Yang. 2025. Distinguishing LLM-generated from human-written code by contrastive learning. *ACM Transactions on Software Engineering and Methodology*, 34(4):1–31.

Danyang Zhang, Junhao Song, Ziqian Bi, Xinyuan Song, Yingfang Yuan, Tianyang Wang, Joe Yeong, and Junfeng Hao. 2025. Mixture of experts in large language models. *Preprint*, arXiv:2507.11181.