# SemEval-2026 Task 13: Detecting Machine-Generated Code with Multiple Programming Languages, Generators, and Application Scenarios

**Daniil Orel**[1], **Dilshod Azizov**[1], **Indraneil Paul**[2],
**Yuxia Wang**[1], **Iryna Gurevych**[1,2], **Preslav Nakov**[1]
[1]MBZUAI, UAE [2]TU Darmstadt, Germany
{daniil.orel, dilshod.azizov, yuxia.wang, iryna.gurevych,
preslav.nakov}@mbzuai.ac.ae, indraneil.paul@tu-darmstadt.de

## Abstract

We present the results and main findings of SemEval-2026 Task 13: Detecting Machine-Generated Code with Multiple Programming Languages, Generators, and Application Scenarios. Our task featured three subtasks. Subtask A is a binary classification task that determines whether a given code snippet is written by a human or generated by a machine. This subtask focuses on the development of robust methods for AI-generated code identification, since the training and the test data splits have code in different languages and cover diverse usage domains. Subtask B focuses on defining synthetic code smells and requires participants to identify the provenance of the generator family of the model that generated the given code snippet. Subtask C aims at more fine-grained attribution of the written code: whether it was fully AI-generated, fully human-written, produced in human-AI collaboration (hybrid) or by a model tuned or prompted to give human-like code. The task attracted a large number of team members: subtask A (81), subtask B (34), and subtask C (32). In this study, we present the task, analyze the results, and discuss the submissions of the system and the methods they used.

## 1 Overview

Previous extensive research and shared tasks have focused on the detection of machine-generated text to identify misuse of large language models (LLMs) (Wang et al., 2024a, 2025). However, the potential misuse of machine-generated code remains under-explored despite its importance. For example, in *academia*, students' reliance on LLMs for assignments undermines educational integrity, with professors unable to detect the authorship of submissions and grading AI-generated coding practices (Koike et al., 2024; Ma et al., 2023). Similarly, when *hiring employees*, it is crucial to ensure that the submitted code authentically represents the skills of the candidate (Veselovsky et al., 2023).

Moreover, machine-generated code can introduce serious vulnerabilities (*e.g.,* insecure logic, hidden backdoors, or injection flaws) that jeopardize software reliability (Bukhari, 2024) and data security (Pearce et al., 2025). It can also facilitate industrial-scale obfuscation, producing code that is harder to parse or analyze, which can hide malicious functionality and complicate efforts to trace and fix bugs (Nunes et al., 2025). Furthermore, with larger reasoning models becoming more capable, machine-generated code may contain harmful snippets that are difficult to identify (Baker et al., 2025). The potentially vulnerable nature of machine-generated code and its inappropriate uses strongly necessitate the provenance estimation of code artifacts in both academia and industry.

Current detection methods often rely on superficial cues, such as naming conventions or stylistic patterns; yet advanced models now can emulate humans by generating natural variable names, realistic formatting, coherent commenting, idiomatic coding styles, and consistent code structures, rendering shallow detection ineffective (Xu et al., 2024; Rahman et al., 2024). The challenge grows with hybrid code that mixes human and machine contributions (Orel et al., 2025a). Some models are even adversarially trained to avoid detection, thus making superficial indicators largely ineffective (Nguyen et al., 2024). Moreover, the proliferation of open-weights, locally deployable LLM-based assistants significantly expands the variety of generation styles, thus undermining one-size-fits-all detection approaches (Orel et al., 2025a). Ultimately, robust machine-generated code detection requires a deeper semantic understanding, cross-programming language adaptation, and resilience against adversarial attacks.

To this end, we propose a shared task that involves developing tools to identify machine-generated code in a variety of scenarios. The shared task consists of three subtasks.



*Proceedings of the The 20th International Workshop on Semantic Evaluation (2026)*, pages 3640–3658
July 3-4, 2026 ©2026 Association for Computational Linguistics

## 2 Summary of Tasks

**Subtask A: Binary Machine-Generated Code Detection.** *Given a code snippet, the participants are asked to identify whether it is (i) fully human-written or (ii) fully machine-generated.*

Our evaluation tests generalization along two axes: programming languages and domains. The seen programming languages in the training set include C++, Python, and Java, and the test-time unseen programming languages comprise Go, PHP, C#, C, and JavaScript. Likewise, *algorithmic code* is the seen domain, whereas *research* and *generic deployed code* are unseen.

The test set for Subtask A covers four evaluation settings: *(i)* seen languages and domains: the test code matches the training languages and domains; *(ii)* unseen languages, seen domains: the test code is in new languages, but from familiar domains; *(iii)* seen languages, unseen domains: the test code is in known languages but from different domains; *(iv)* unseen languages and domains: the test code differs in both language and domain, requiring a wide generalization. This approach enables us to analyze submitted solutions from multiple views.

Overall, subtask A aims to solve the problems mentioned in previous work (Orel et al., 2025a; Wang et al., 2024b; Orel et al., 2026): AI-generated content detectors often fail in out-of-domain settings. The simplicity of the task design and its varied evaluation settings make this task appealing to a diverse audience, from AI practitioners to seasoned experts.

**Subtask B: Multi-Class Authorship Detection.** *Given a code snippet, the participants are asked to predict whether it is: (i) fully human-written, or generated by a model belonging to one of the following ten model providers: (ii) DeepSeek-AI, (iii) Qwen, (iv) 01-ai, (v) BigCode, (vi) Gemma, (vii) Phi, (viii) Meta-LLaMA, (ix) IBM-Granite, (x) Mistral, or (xi) OpenAI.* All model family representatives are listed in Table 7.

In the test set of Subtask B, we include samples for two evaluation settings: *(i)* seen authors, where test generators appear in training; and *(ii)* unseen authors, where generators are new but from known families. Our data includes 45 generators, averaging four per family. Subtask B builds on CoDet-M4 (Orel et al., 2025a) and introduces more complex classification challenges to determine whether a code snippet is human-written or generated by a specific LLM.

**Subtask C: Fine-Grained Code Classification.** *Given a code snippet, the participants are asked to identify whether it is: (i) fully human-written, (ii) fully machine-generated, (iii) hybrid, where human-written code is either partially completed or rewritten by an LLM, and (iv) adversarial, where the code is generated in response to prompts designed to produce "human-like" outputs, or the code is generated through adversarial training (RLHF) to mimic human-authored code.*

In Subtask C, we extend ideas introduced by `Droid` (Orel et al., 2025b) and CoDet-M4 (Orel et al., 2025a), which show that code classification should not be treated as purely binary, but should instead account for hybrid and adversarially generated code.

As LLMs increasingly assist in writing programming code, hybrid authorship becomes more common. For example, students may draft assignments and enhance them with AI, while candidates might submit AI-assisted code during hiring, highlighting the need to assess genuine programming skills and originality. Moreover, distinguishing AI-generated code is vital for developing advanced frameworks.

Subtask C addresses these challenges by enabling detection of AI-assisted plagiarism, helping companies verify code authenticity during hiring, and expanding human-written code datasets for developing code-based frameworks. This task drew significant attention from the AI community due to its novelty, practicality, and growing interest in generative AI for programming.

## 3 Dataset and Evaluation Measures

**Data Sources.** We build on `Droid` (Orel et al., 2025b), which includes CodeNet (Puri et al., 2021), ObscuraCoder (Paul et al., 2025), StarCoder-Data (StarCoder, 2023), Vault (Manh et al., 2023), and TACO (Li et al., 2023), add LeetCode and CodeForces problems from Orel et al. (2025a), and expand further using Droid's generation pipeline. We use several generation strategies. For inverse prompting, an LLMs first derives a prompt from the source code, and other LLMs then generate code from that prompt. We also generate code from comments, document strings, and problem statements. For hybrid samples, we ask LLMs to fill in removed regions or rewrite parts of the original code. Adversarial code is produced either by prompting LLMs to imitate human-written code or by tuning them for that objective.

| Split | Generator | #Samples | Languages | Domains |
|---|---|---|---|---|
| Train | Human | 238,475 | Python, Java, C++ | Algorithmic Code |
| | AI | 261,525 | Python, Java, C++ | Algorithmic Code |
| Validation | Human | 47,695 | Python, Java, C++ | Algorithmic Code |
| | AI | 52,305 | Python, Java, C++ | Algorithmic Code |
| Test | Human | 390,054 | C, C#, C++, Go, Java, JavaScript, Python | Algorithmic Code, Research, Generic Deployed Code |
| | AI | 109,946 | C, C#, C++, Go, Java, JavaScript, Python | Algorithmic Code, Research, Generic Deployed Code |

Table 1: **Subtask A: Binary Machine-Generated Code Detection**. Dataset statistics.

**Programming Languages and Generators.** For generation across languages including C++, C#, Python, Java, Go, PHP, and JavaScript, we use models from the families of Yi (01.AI, 2024), Qwen (Qwen, 2023), StarCoder (StarCoder, 2023), DeepSeek (DeepSeek, 2024), Gemma (Gemma, 2024), IBM-Granite (IBM, 2024), Llama (Meta, 2024) (including CodeLlama models), Mistral (AI, 2025), and Phi (Microsoft, 2024).

**Subtask A.** In Table 1, we present the dataset statistics for Subtask A. Although the task is a binary classification, we introduce substantial data drift. The training data consists solely of algorithmic code in Python, Java, and C++, while the test set includes additional languages (C, C#, JavaScript, and Go) and broader use cases. Specifically, the test data spans Algorithmic Code, Research Code, and Generic Deployed Code, making the evaluation more challenging and realistic.

**Subtask B.** In Table 2, we demonstrate statistics for Subtask B splits based on the number of samples and generators. We use multiple generators per model family and construct the test split to include both seen and unseen generators.

**Subtask C.** As shown in Table 3, for Subtask C, we used all available data and kept the number of generators consistent across splits. The test set also includes human-authored solutions from The Heap dataset (Katzy et al., 2025), containing repositories not used in The Stack (the primary source for training and validation), as well as hybrid samples from SwallowCoder (Fujii et al., 2025), where human code is rewritten with Llama 3.3 70B Instruct.

| Split | Generator | #Samples | #Models |
|---|---|---|---|
| Train | Human | 442,096 | 1 |
| | GPT | 10,810 | 1 |
| | Qwen | 8,993 | 5 |
| | Llama | 8,197 | 5 |
| | IBM-Granite | 8,127 | 3 |
| | Phi | 5,783 | 3 |
| | Mistral | 4,608 | 2 |
| | DeepSeek | 4,162 | 3 |
| | Yi | 3,029 | 2 |
| | StarCoder | 2,227 | 3 |
| | Gemma | 1,968 | 3 |
| Validation | Human | 88,490 | 1 |
| | GPT | 2,154 | 1 |
| | Qwen | 1,755 | 5 |
| | Llama | 1,695 | 5 |
| | IBM-Granite | 1,579 | 3 |
| | Phi | 1,118 | 3 |
| | Mistral | 895 | 2 |
| | StarCoder | 847 | 3 |
| | Yi | 650 | 2 |
| | BigCode | 445 | 3 |
| | Gemma | 372 | 3 |
| Test | Human | 243,769 | 1 |
| | GPT | 104,304 | 2 |
| | Qwen | 26,459 | 12 |
| | Llama | 35,411 | 13 |
| | IBM-Granite | 11,202 | 7 |
| | Phi | 20,981 | 7 |
| | Mistral | 14,185 | 5 |
| | DeepSeek | 9,674 | 6 |
| | Yi | 8,471 | 4 |
| | StarCoder | 3,631 | 6 |
| | Gemma | 21,913 | 8 |

Table 2: **Subtask B: Multi-Class Authorship Detection**. Dataset statistics.

**Data Filtering.** To ensure data quality, we apply filtering and validation to both original and generated code. We verify syntactic validity using Abstract Syntax Trees (ASTs) and filter samples by AST depth, line length, and number of lines, removing overly simple, complex, or unstructured entries, retaining only English samples. The filtering pipeline follows Droid's approach (Orel et al., 2025b). MinHash deduplication (Broder, 1997) eliminates redundancy while preserving diversity. To avoid contamination, we use human-written code up to 2022, before widespread code LMs; however, this cannot guarantee that our human-written code is contamination-free.

**Evaluation Measures.** Due to class imbalance, we adopt macro-averaged F1 as the primary evaluation measure across all tasks. It is computed by calculating the F1 score for each class and then averaging across classes, giving equal weight to each class regardless of frequency.

| Split | Label | #Samples | #Generators |
|---|---|---|---|
| Train | Human | 485,483 | 1 |
| | AI | 210,471 | 78 |
| | Hybrid | 85,520 | 28 |
| | Adversarial | 118,526 | 17 |
| Validation | Human | 107,885 | 1 |
| | AI | 46,770 | 78 |
| | Hybrid | 19,006 | 28 |
| | Adversarial | 26,339 | 17 |
| Test | Human | 334,770 | 1 |
| | AI | 58,510 | 78 |
| | Hybrid | 73,889 | 29 |
| | Adversarial | 32,831 | 17 |

Table 3: **Subtask C: Fine-Grained Code Classification**. Dataset statistics.

## 4 Task Organization

The shared task was conducted in two phases: development and testing.

**Development Phase.** The training and validation sets, along with a small portion of the test set, were provided to the participants. Teams competed on a 1K-sample test subset, tracked via a live Kaggle leaderboard. Up to 50 daily submissions were allowed, with each team's best score displayed in real time.

**Testing Phase.** All previous submissions were invalidated and the test set was updated with unseen samples. Participants had 14 days to evaluate their systems; no test labels were provided and evaluation was conducted on Kaggle with a real-time leaderboard reflecting the best-performing solutions. The task attracted many teams: Subtask A (81), Subtask B (34), and Subtask C (32). After the competition ended, we did not release the gold test labels to prevent saturation, but kept the submission system open for post-task evaluation and monitoring state-of-the-art results across subtasks.

## 5 Participating Systems

We summarize common characteristics across participating teams and describe in more detail the approaches of the top three systems in each subtask, followed by brief notes on the remaining submissions. System performance is reported in Tables 4, 5, and 6, while a broader quantitative analysis of the leaderboard results is provided in Appendix C. Complete system descriptions can be found in Appendix D.

### 5.1 General Solution Trends

We analyze the approaches used by the teams in Table 8 (see Appendix E). Transformer-based encoders were the dominant backbone: CodeBERT was the most widely used, followed by UniXcoder and Modern-BERT. Several teams relied on large instruction-tuned LLMs with LoRA fine-tuning, while others preferred smaller pretrained code models with task-specific adaptations. Feature engineering also played a major role, with many systems incorporating hand-made structural, stylometric, or AST-based features. In contrast, advanced strategies such as pseudo-labeling, curriculum learning, and focal loss were adopted by only a few teams. Ensemble techniques were common, particularly logit-level stacking and tree-based models (e.g., XGBoost/CatBoost), whereas specialized components such as Multiple Instance Learning (MIL) or explicit label-flip post-processing were rare. In general, the table highlights a methodological split between purely neural approaches and hybrid systems that integrate transformer embeddings with engineered structural signals.

### 5.2 Top Solutions of Subtask A

**Team TeleAI.** Team TeleAI participated in Subtasks A, B, and C with a unified multi-stage framework that combines data-centric preprocessing, prompt optimization, fine-tuning, and assembly. They addressed class imbalance via repeated oversampling and standardized prompts to improve output consistency. The core model, `Qwen3-30B-A3B`, was fully fine-tuned to capture stylistic and structural differences between human and AI-generated code. The final predictions were produced through multi-level assembly, integrating multiple models and checkpoints through soft voting, hard voting, and LightGBM stacking to improve robustness to distribution changes (Wang et al., 2026).

**Team Alejandro Mosquera.** Alejandro Mosquera employed a transformer-based approach centered on UniXcoder, complemented by data-centric training strategies. The system incorporates pseudo-labeling and curriculum learning to refine decision boundaries, alongside feature engineering to capture structural signals. Training is stabilized using focal loss to better handle difficult examples. Rather than relying on ensembling or classical baselines, the approach improves representation learning within a single neural backbone through structured training enhancements.

**Team UIT-AMMC.** For Subtask A, the team proposed a *Structure-Aware Contrastive Cascade*, a multi-stage architecture combining generative reasoning with explicit structural features to distinguish human-written from AI-generated code. Their approach exploits the formatting and structural signatures introduced by alignment-tuned LLMs. The core model, `Qwen-2.5-Coder-14B`, was fine-tuned with QLoRA and enhanced with stochastic data augmentation for cross-language robustness. Final predictions were produced via late fusion of contrastive probabilities and structural statistics, with high-uncertainty cases refined through a multi-agent adversarial debate module (Cuong et al., 2026).

### 5.3 Top Solutions of Subtask B

The best solution is provided by the **TeleAI team** and is already described in Section 5.2.

**Team Yuvan.** For Subtask B, Team Yuvan tackled extreme class imbalance in the 11-way setting using direct multi-class training with inverse-frequency class weighting. They fine-tuned `Qwen-2.5-Coder-1.5B` with LoRA (rank=16, $\alpha$ = 32) in stratified samples for five epochs using distributed training. An earlier two-stage pipeline (binary detection followed by generator classification) was discarded due to cascading errors, and the final end-to-end multi-class formulation proved more stable. Training used bfloat16 mixed precision, gradient checkpointing, and fixed seeds for reproducibility (Ramesh and Wu, 2026).

**Team SYSUpporter.** Team SYSUpporter fine-tuned the UniXcoder-base for 11-way classification. To handle long sequences within the 1024-token constraint, they introduced a language-aware truncation strategy that preserves both head and tail segments, with ratios dynamically adjusted per language. Training included randomized truncation and window sampling for robustness. Additionally, whitespace and formatting cues such as indentation type, trailing whitespace, and line endings were injected as special tokens, exploiting generator-specific stylistic signatures. The final system employed 5-fold stratified cross-validation with Focal Loss, Generalized Mean Pooling, Multi-Sample Dropout, and bucket batching, averaging fold predictions for final outputs (Chen and Zheng, 2026).

### 5.4 Top Solutions Subtask C

**Team YoungDSMLKZ.** Team YoungDSMLKZ proposed a multi-level ensemble combining three complementary pipelines with prior-adjustment calibration and a hierarchical decision strategy. The first pipeline applies Dynamic Multiple Instance Learning (MIL) with UniXcoder to process long files by aggregating chunk-level evidence, followed by a CatBoost classifier. The second uses meta-stacking, fusing probabilities from fine-tuned UniXcoder and ModernBERT via a CatBoost meta-learner. The third applies XGBoost over 200+ handcrafted stylometric and structural features, including entropy, comment density, and AST statistics. The final predictions follow a hierarchical class-priority rule to mitigate systematic class biases (Gainulla and Shamsadinov, 2026).

The second-best solution was proposed by the **TeleAI team** and is detailed in Section 5.2.

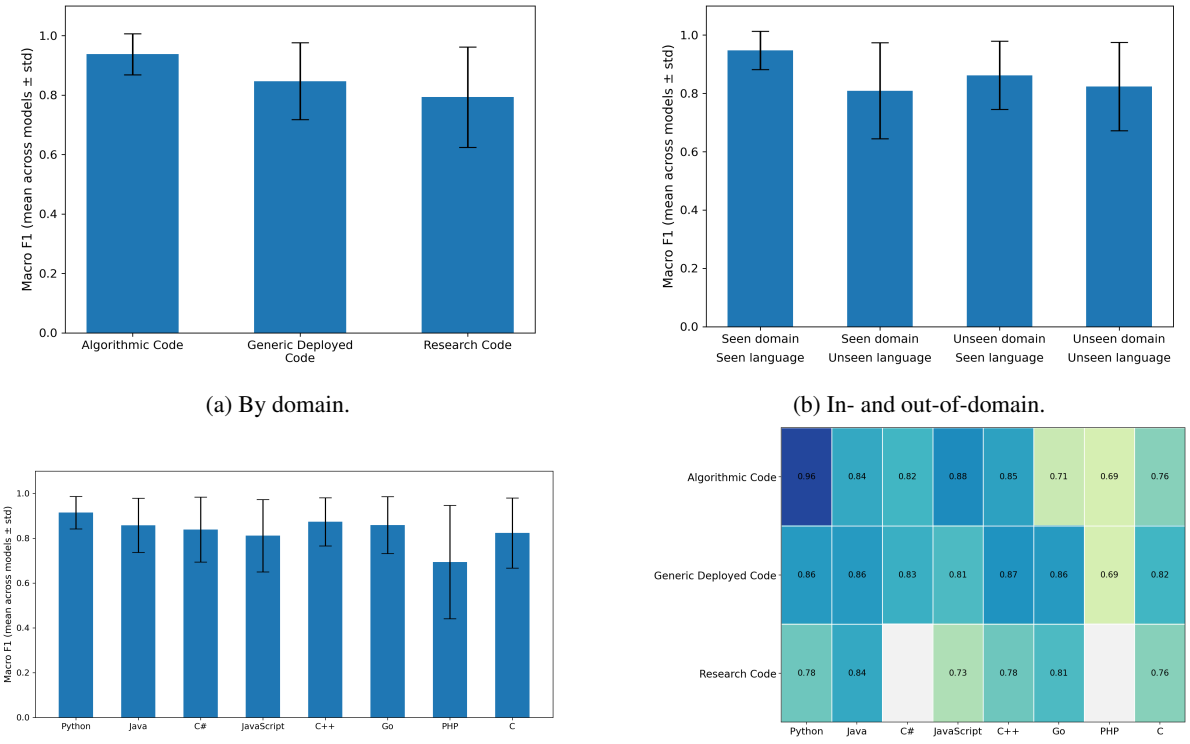
**Team Yuvan.** For Subtask C, Team Yuvan used a three-model weighted ensemble to maximize feature diversity. The system combines LoRA-fine-tuned `Qwen-2.5-Coder-1.5B` (65%), a TF-IDF character n-gram model with Logistic Regression (25%), and CodeBERT with head-tail splitting for long-context modeling (10%). The weights were optimized by searching the validation data grid. In particular, feature diversity proved more valuable than individual model strength, as the lexical model contributed complementary signals despite lower standalone performance. The neural components were trained with mixed precision and fixed seeds for reproducibility (Ramesh and Wu, 2026).

## 6 Performance Analysis

We analyze the error patterns of the Top-3 submissions across all subtasks, focusing on how performance degrades under distribution shifts in language, domain, generator identity, and label granularity.

### 6.1 Subtask A

Figure 1 reveals several clear patterns. **The domain shift dominates the language shift.** Performance is highest for *algorithmic code* (the seen domain), while both *research* and *generic deployed code* show lower Macro-F1 and higher variance. This confirms that detectors trained primarily on algorithmic code struggle to generalize stylistically and structurally to more realistic or heterogeneous coding environments.




(a) By domain.

(b) In- and out-of-domain.

(c) By programming language.

(d) Source group × language.

Figure 1: Error analysis on difficult samples using the Top-3 best submissions. (a) Macro-F1 by domain; (b) Macro-F1 by seen/unseen domain and seen/unseen language; (c) Macro-F1 by programming language; (d) source-group × language Macro-F1 heatmap, where blank cells indicate that no such combination exists.

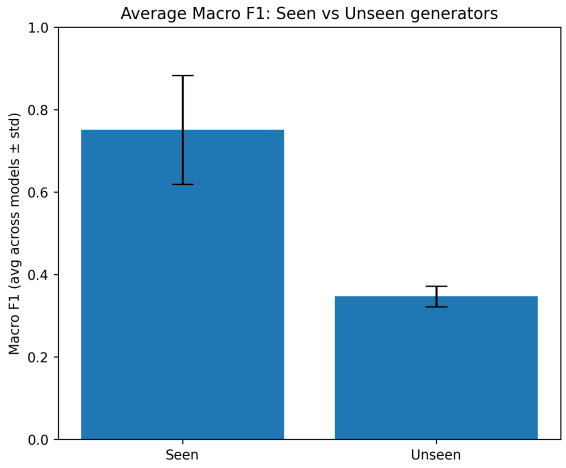
**Unseen domains hurt more than unseen languages.** When isolating evaluation regimes, performance degradation is more pronounced for unseen domains than for unseen languages. Models transfer stylistic signals better across languages than across domains, suggesting that generator artifacts are partially language-agnostic but domain-sensitive. **Language-specific brittleness.** Across languages, Python yields the highest Macro-F1, likely reflecting the size and stylistic regularity of the dataset. In contrast, PHP and JavaScript show the lowest performance and higher variance, indicating instability in detecting artifacts in web-oriented or loosely structured languages. **The compound shift is the hardest setting.** The source-group × language heatmap shows that combining unseen languages and unseen domains produces the most challenging cells. The Research Code in JavaScript and Java shows substantial degradation, highlighting the difficulty of generalizing across domains and languages.

In general, Subtask A confirms that domain generalization is the primary bottleneck in binary machine-generated code detection.

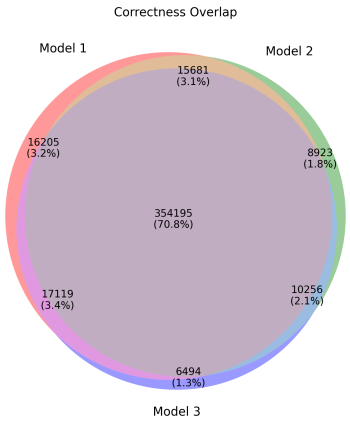
### 6.2 Subtask B

**The generator shift is severe.** As shown in Fig. 2a, Macro-F1 drops substantially for unseen generators. Unlike Subtask A, which aggregates artifacts into a binary signal, Subtask B requires distinguishing stylistically similar LLM families, and the performance gap indicates a partial overfitting to generator-specific signatures rather than robust family-level traits. The correctness-overlap analysis (Fig. 2b) shows that 70.8% of the samples are correctly classified by all three top systems. Each model contributes uniquely correct predictions, suggesting complementary decision boundaries, yet the union covers 85.7% of the dataset, leaving 14.3% misclassified by all Top-3 models. The most difficult cases come from the Phi and Llama families, which show the greatest internal diversity, whereas the fewest errors occur for human-written code, indicating that models capture human patterns more reliably than fine-grained stylistic differences within LLM families.

Compared to Subtask A, Subtask B highlights that fine-grained attribution is substantially more sensitive to generator shift than binary detection.




(a) Macro-F1 on seen vs. unseen generators (mean ± std).




(b) Correctness overlap among Top-3 submissions. The outside-union portion (14.3%) corresponds to cases where none of the models is correct.

Figure 2: Error analysis for Subtask B: generator generalization and agreement among top submissions.

### 6.3 Subtask C

**Language-level trends persist.** Figure 3a shows that performance remains strongest for Python, C++, Go, and C, while PHP is consistently more challenging due to its distinct syntax. This mirrors the Subtask A patterns, suggesting that language structure influences the detectability of generative artifacts even under finer-grained labeling.

**Research code remains the most challenging domain.** The source-group × language heatmap (Fig. 3b) shows substantial degradation for Research Code, especially in JavaScript, Go, and Java, likely due to higher stylistic variability and subtler human-LLM blending patterns.

**Hybrid and adversarial signals increase the ambiguity.** Unlike Subtask A, Subtask C requires distinguishing partial machine assistance (hybrid) and deliberately human-like machine outputs (adversarial). Although these settings blur stylistic boundaries and reduce artifact-based separability, overall system performance in Subtask C is higher than in Subtask A, indicating that learning these patterns is easier than adapting to domain changes.

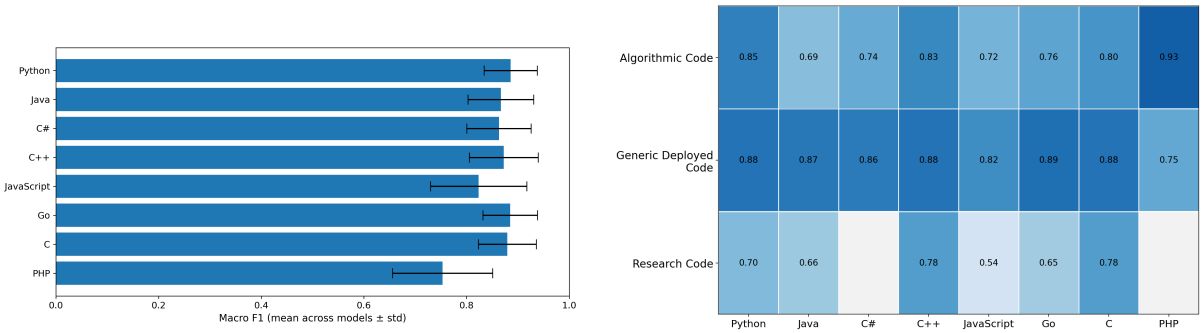
In general, Subtask C shows that increasing the granularity of the label does not necessarily lead to performance drops.

## 7 Conclusion and Future Work

In this work, we summarized SemEval-2026 Task 13 on Detecting Machine-Generated Code with Multiple Programming Languages, Generators, and Application Scenarios.

The task attracted significant interest from the research community due to its practical relevance and methodological challenges. Subtask C was comparatively simpler, Subtasks A and B posed greater difficulty due to the strong out-of-domain evaluation settings, revealing the limitations of current systems under distribution shift.

In future work, we plan to extend the task toward multi-modal (including audio and images) settings, incorporating audio and visual modalities to study cross-modal consistency in AI-generated artifacts. We also aim to develop open-source detection systems to support transparent and reproducible benchmarking. Beyond these directions, several research avenues remain open. First, as AI coding assistants, such as Cursor and Claude Code, become deeply integrated into everyday development workflows, machine-generated code detection must adapt to a setting where AI assistance is ubiquitous and continuously updated. It goes beyond the hybrid category, since it is important to study how detection systems perform when AI participation is subtle, iterative, and shaped by rapidly evolving toolchains. This shift also calls for stronger considerations of safety, reliability, and efficiency in realistic deployment environments. Second, more principled domain generalization techniques, such as invariant representation learning and causal modeling, could reduce overfitting to generator-specific artifacts. Third, expanding evaluation to low-resource programming languages would improve ecological validity.




(a) Macro-F1 by programming language (mean ± std across models).

(b) Macro-F1 by source group × language for Subtask C.

Figure 3: Error analysis for Subtask C: performance across languages and source groups.

Additionally, interpretability-focused methods could help identify which structural or stylistic cues drive predictions, increasing trust in deployment settings. Finally, investigating adversarial robustness and adaptive generators is essential to understanding how detection systems perform in dynamic, competitive environments where models may intentionally obfuscate their signatures.

## 8 Limitations

Despite providing a comprehensive dataset that spans multiple languages, generators, and domains in three distinct tasks in machine-generated code detection, our study faces several limitations that pave the way for future research.

Firstly, the methods described in the competition rely solely on black-box information. These methods might exhibit reduced effectiveness and struggle to generalize across new domains, generators, and languages.

Secondly, although the task is designed to approximate realistic coding scenarios, its coverage remains limited in two main respects. The data is filtered using constraints such as the depth of the AST, the length of the line, and the overall size of the code to ensure quality and consistency. Although this produces clean samples, it excludes highly verbose, overly complex, extremely short, or poorly structured code that often appears in real-world repositories, potentially reducing the diversity of syntactic and stylistic patterns represented. Additionally, the task currently includes 8 widely used programming languages (C++, C, C#, Go, Java, JavaScript, PHP, Python). Although these languages cover many practical use cases, they constitute a fixed set and do not evaluate model generalization beyond this selection.

## 9 Ethical Statement

**Data Collection and Privacy.** The dataset is constructed from publicly available research corpora and model outputs generated via documented interfaces, in compliance with platform terms and licenses. Human-written code is sourced from published datasets, with no private repositories, paywalled content, or sensitive personal data included.

**Bias.** Human- and LLM-generated code can reflect biases that stem from data availability, platform popularity, community practices, and training corpora. Consequently, the task may inherit distributional skews (e.g., language, domain, or stylistic imbalances) that affect external validity. We mitigate these risks through diverse sampling across platforms, programming languages, and generator families; however, complete representativeness cannot be guaranteed.

**Risk of Misuse.** Our objective is to improve transparency in AI-assisted programming and enable applications such as plagiarism detection, compliance checks, and provenance auditing. To reduce potential misuse (e.g., evasion of detection systems) and to prevent saturation of the benchmark, we avoid releasing the entire testing set.

**Broader Impact.** Our task attracted significant attention from researchers around the world, highlighting challenges in AI-generated code detection. By framing detection as a multi-setting, realism-oriented benchmark, we move beyond simplified binary setups toward robustness under distribution shifts. Despite limitations, the task provides a foundation for more transparent, accountable, and trustworthy AI-assisted programming systems. We are also releasing an enlarged version (with more models and slightly harder subtask constraints) of the dataset as part of AICD Bench (Orel et al., 2026).

| Rank | Team Name | Macro F-score | #Entries |
|---|---|---|---|
| 1 | TeleAI | 0.99708 | 3 |
| 2 | Alejandro Mosquera | 0.82409 | 78 |
| 3 | UIT_AMMC | 0.80188 | 21 |
| 4 | CEIA | 0.75318 | 23 |
| 5 | Juan David Villate | 0.75209 | 12 |
| 6 | Jany-Gabriel Ispas | 0.73759 | 67 |
| 7 | Dream | 0.73087 | 16 |
| 8 | UIT_DoubleDat | 0.73006 | 5 |
| 9 | QQQ | 0.72234 | 7 |
| 10 | mcdok | 0.69753 | 69 |
| 11 | Andrei-Iosif Iojă | 0.69334 | 22 |
| 12 | YNU-HPCC | 0.69231 | 15 |
| 13 | FMI_SU_Yotkova_Kastreva | 0.67347 | 155 |
| 14 | TransfomerTrio | 0.67240 | 61 |
| 15 | hristoboyanov | 0.66046 | 27 |
| 16 | CodeMonks | 0.64735 | 50 |
| 17 | YZhang | 0.63825 | 22 |
| 18 | Yuvan Ramesh | 0.63825 | 38 |
| 19 | WWR SFE | 0.63558 | 17 |
| 20 | Pham Quoc Cuong | 0.63524 | 3 |
| 21 | shubham kannaujiya | 0.62362 | 16 |
| 22 | Shantanu Thorat | 0.61432 | 2 |
| 23 | Sruthi Santhanam | 0.61248 | 10 |
| 24 | CoDetect | 0.60204 | 60 |
| 25 | Coderz | 0.60169 | 3 |
| 26 | CodeHunters | 0.58724 | 4 |
| 27 | Talida Caraman | 0.57659 | 4 |
| 28 | andedge | 0.57659 | 2 |
| 29 | Daniel Scheider | 0.57341 | 5 |
| 30 | Adithya Srinivasan133 | 0.56773 | 24 |
| 31 | Rincu Stefania | 0.56367 | 2 |
| 32 | dsukhotin | 0.56089 | 2 |
| 33 | ReLU Deal | 0.55929 | 4 |
| 34 | Atharv Ramesh | 0.54725 | 16 |
| 35 | CUET_Luminaries_0227 | 0.54110 | 23 |
| 36 | Md. Rafat Kabir | 0.53834 | 10 |
| 37 | Mind_Flayer | 0.53822 | 13 |
| 38 | LATE-IIMAS | 0.53809 | 21 |
| 39 | CSMetro | 0.53743 | 52 |
| 40 | Udaythalavesh S | 0.53588 | 7 |
| 41 | WWTC@UniA | 0.53473 | 3 |
| 42 | Shibam Mandal | 0.52391 | 4 |
| 43 | contestant001 | 0.51756 | 3 |
| 44 | Team_Omega | 0.51321 | 2 |
| 45 | Isun chehreh | 0.51180 | 51 |
| 46 | Syntax_Sentinels | 0.49680 | 2 |
| 47 | Team_SLS | 0.49655 | 4 |
| 48 | 823 | 0.49432 | 15 |
| 49 | Ayah Khaldi | 0.49072 | 4 |
| 50 | Walisa | 0.47733 | 11 |
| 51 | NUST-CodeIntel | 0.47328 | 5 |
| 52 | Dipannita Orni | 0.46344 | 8 |
| 53 | Codexa | 0.45245 | 6 |
| 54 | NLP-UB | 0.44209 | 1 |
| 55 | Segmentation fault | 0.43913 | 4 |
| 56 | waxemej799 | 0.43823 | 1 |
| 57 | Sadiba Nusrat Nur | 0.43823 | 2 |
| 58 | Barsha Mishra | 0.42663 | 1 |
| 59 | AI Detector | 0.40893 | 5 |
| 60 | Varun Nivass G | 0.40282 | 1 |
| 61 | Salvo Cappello | 0.39984 | 1 |
| 62 | Diana Sician | 0.39879 | 2 |
| 63 | Tony Ironman | 0.38380 | 5 |
| 64 | AIgenvalues @UniA | 0.38377 | 3 |
| 65 | Königsberg | 0.38262 | 7 |
| 66 | adobosz | 0.37267 | 2 |
| 67 | Janani Hariharakrishnan | 0.36866 | 6 |
| 68 | SSN-CSE-CODECATALYSTS | 0.32826 | 3 |
| 69 | Grzegorz (Gregory) Kłosiński | 0.32303 | 2 |
| 70 | MedHastra | 0.32222 | 3 |
| 71 | Ruwad Naswan | 0.31726 | 1 |
| - | Baseline | 0.30530 | |

Table 4: **SemEval-2026 Task 13 Subtask A leaderboard.**

| Rank | Team Name | Macro F-score | #Entries |
|---|---|---|---|
| 1 | TeleAI | 0.50819 | 2 |
| 2 | TocToc | 0.45595 | 30 |
| 3 | Yuvan Ramesh | 0.44882 | 4 |
| 4 | SYSUpporter | 0.43399 | 7 |
| 5 | WWTC@UniA | 0.42986 | 1 |
| 6 | Wooo2002 | 0.42953 | 53 |
| 7 | Jany-Gabriel Ispas | 0.42219 | 20 |
| 8 | CodeDet | 0.41570 | 5 |
| 9 | Codexa | 0.41278 | 62 |
| 10 | ICI Innolabs | 0.41104 | 35 |
| 11 | MALTO | 0.40500 | 16 |
| 12 | TransformerTrio | 0.39968 | 9 |
| 13 | mcdok | 0.39553 | 16 |
| 14 | YNU-HPCC | 0.39371 | 34 |
| 15 | Giovanni Iacuzzo | 0.38493 | 1 |
| 16 | Hutan Mihai-Alexandru | 0.38294 | 1 |
| 17 | AIgenvalues @UniA | 0.37486 | 2 |
| 18 | AI_Detector | 0.36811 | 5 |
| 19 | Osint | 0.35693 | 2 |
| 20 | Shifali Agrahari | 0.34796 | 2 |
| 21 | Sandhya Giribabu | 0.34425 | 1 |
| 22 | Zhennor | 0.34027 | 1 |
| 23 | CodeHunters | 0.32469 | 6 |
| 24 | Vedajanaani | 0.31425 | 1 |
| 25 | LATE-IIMAS | 0.30287 | 1 |
| 26 | Farhan Nafis Rayhan | 0.29639 | 9 |
| 27 | Aadit P | 0.28898 | 1 |
| 28 | Alimasigna | 0.28177 | 1 |
| 29 | Saud Khan | 0.28177 | 1 |
| 30 | SSN-CSE-CodeBrothers | 0.25715 | 1 |
| 31 | adobosz | 0.25483 | 1 |
| - | Baseline | 0.22858 | |

Table 5: **SemEval-2026 Task 13 Subtask B leaderboard.**

| Rank | Team Name | Macro F-score | #Entries |
|---|---|---|---|
| 1 | Young DSMLKZ | 0.78546 | 356 |
| 2 | TeleAI | 0.73332 | 14 |
| 3 | Yuvan Ramesh | 0.71398 | 2 |
| 4 | Brazil AI | 0.69770 | 57 |
| 5 | mcdok | 0.68643 | 22 |
| 6 | WWTC@UniA | 0.64765 | 1 |
| 7 | TransformerTrio | 0.63368 | 16 |
| 8 | YNU-HPCC | 0.62599 | 27 |
| 9 | Ilyaaaas_N | 0.61865 | 7 |
| 10 | MALTO | 0.61557 | 24 |
| 11 | Antonella Coviello | 0.61434 | 11 |
| 12 | Jany-Gabriel Ispas | 0.61429 | 12 |
| 13 | Stefania Rincu | 0.59928 | 2 |
| 14 | Hutan Mihai-Alexandru | 0.59204 | 2 |
| 15 | Saud Khan | 0.58925 | 1 |
| 16 | Ruwad Naswan | 0.58548 | 1 |
| 17 | AIgenvalues @UniA | 0.58454 | 2 |
| 18 | TeamBots | 0.57873 | 7 |
| 19 | ai detector | 0.57415 | 1 |
| 20 | N4G1B4T0R5 | 0.57319 | 2 |
| 21 | Kyle Yu | 0.57222 | 10 |
| 22 | LATE-IIMAS | 0.56775 | 1 |
| 23 | Alimasigna | 0.53865 | 1 |
| 24 | Vedajanaani | 0.53134 | 1 |
| 25 | Carson Cheng | 0.52936 | 4 |
| 26 | AI Detectors | 0.52691 | 1 |
| 27 | SSN-CSE-CodeBrothers | 0.52556 | 1 |
| 28 | Dipit Saha | 0.52498 | 1 |
| 29 | Aadit P | 0.49754 | 1 |
| 30 | adobosz | 0.49674 | 2 |
| - | Baseline | 0.48120 | |

Table 6: **SemEval-2026 Task 13 Subtask C leaderboard.**

## References


Team 01.AI. 2024. Yi: Open foundation models by 01.ai. *ArXiv*, abs/2403.04652.

Mistral AI. 2025. Mistral small – a new balance of performance and efficiency. Online. Available at https://mistral.ai/news/mistral-small-3 (Accessed: 1 April 2025).

Azher Ali and Mehwish Fatima. 2026. NUST-CodeIntel at SemEval-2026 task 13: Cross-domain detection of machine-generated code via stylometric features and transformer models. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Abhishek Anand, Shifali Agrahari, Shubham Nilesh Kannaujiya, Sanasam Ranbir Singh, and Sujit Kumar. 2026. Osint at SemEval-2026 task 13: A distribution-aware framework for code generation detection and multi-source authorship attribution. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Hüseyin Arda Arslan, Evren Ayberk Munis, Timofei Khudonogov, Mert Akgün, Murat Beşli, Ayhan Meherrem, Claudio Savelli, and Flavio Giobergia. 2026. MALTO at SemEval-2026 task 13: Detecting human, AI, and hybrid code via hard negative mining and curriculum-driven ensembles. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y. Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. 2025. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*.

Sebastian Balmuș and Bogdan Dura. 2026. ICI Innolabs at SemEval-2026 task 13: Sliding windows meet code transformers. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Ruslan Berdichevsky, Shai Nahum-Gefen, and Elad Ben-Zaken. 2026. Dream at SemEval-2026 task 13: SALSA for single-pass machine-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 21–29. IEEE.

Sufiyan Ahmed Bukhari. 2024. Issues in detection of AI-generated source code. *University of Calgary*.

Shruti Chandrasekar, Vedajanaani R S, and Vijayalakshmi P. 2026. Medhastra at SemEval-2026 task 13: Stylometric ensembles and transformer fine-tuning for robust AI code detection, attribution, and adversarial analysis. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Longfeng Chen and Xiao Zheng. 2026. SYSUpporter at SemEval-2026 task 13: Leveraging stylistic signals and language-aware truncation for code generation detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Shiti Chowdhury and Adnan Faisal. 2026. CUET_Luminaries_0227 at SemEval-2026 task 13: Invariance-oriented representation learning for robust AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Aldan Creo, Atharv Nair, Mohana Kannan Ravikumar, Vaishak Menon, Dario Wisznewer, and Vaibhav Jain. 2026. COODetect at SemEval-2026 task 13: Unsupervised latent domain adaptation for out-of-distribution AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Pham Quoc Cuong, Nguyen Ngoc Minh, Le Quang Minh, Nguyen Hoai Nguyet An, and Nguyen Trong Chinh. 2026. UIT-AMMC at SemEval-2026 task 13: Exploiting structural formatting signatures for robust AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Team DeepSeek. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Anıl Dervişoğlu and Atakan Site. 2026. Codexa at SemEval-2026 task 13: Loss engineering and diverse ensemble strategies for multi-class code authorship attribution. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Vivek Dhayal. 2026. Team vivek dhayaal at SemEval-2026 task 13: Multi-class authorship detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

How Chuen Yu Dmitry Sukhotin. 2026. Segmentation Fault at SemEval-2026 task 13: Various approaches to detecting machine- generated code. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Daniel-Antoniu Dumitru, Simina Lazăr, Nicoleta Danilă (Amargheoalei), Daniela Gîfu, and Diana Trăndăbăț. 2026. CodeHunters at 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.

Fariska Zakhralativa Ruskanda Farhan Nafis Rayhan. 2026. Farhan nafis rayhan at SemEval-2026 task 13: Supervised contrastive learning approach with gated multiclass decomposition ensemble architecture for code authorship identification. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Kazuki Fujii, Yukito Tajima, Sakae Mizuki, Hinari Shimada, Taihei Shiotani, Koshiro Saito, Masanari Ohi, Masaki Kawamura, Taishi Nakamura, Takumi Okamoto, Shigeki Ishida, Kakeru Hattori, Youmi Ma, Hiroya Takamura, Rio Yokota, and Naoaki Okazaki. 2025. Rewriting pre-training data boosts LLM performance in math and code. *ArXiv preprint*, abs/2505.02881.

Yeraly Gainulla and Agzam Shamsadinov. 2026. YoungDSMLKZ at SemEval-2026 task 13: Milunixcoder with meta-stacking and handcrafted features for AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Team Gemma. 2024. Gemma: Open models based on Gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Sai Laasya Gorantla, Shreemithra Naveen, and Steven Bethard. 2026. Team_SLS at SemEval-2026 task 13: Detecting machine-generated code with CodeBERT and structural features. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Shahir Habib and Dipankar Das. 2026. Königsberg at SemEval-2026 task 13: Beyond language models: A low-resource feature-driven and data-flow embedding approach for machine-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Team IBM. 2024. Granite code models: A family of open foundation models for code intelligence. *ArXiv*, abs/2405.04324.

Jany-Gabriel Ispas and Sergiu Nisioi. 2026. Archaeology at SemEval-2026 task 13: Fine-tuning pretrained code models for AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Bhuvana J, Ramanan Mahendran, Siddharth Chandrasekar S, Pragatheesh J, and Rethanya P. 2026. SSN-CSE-CODECATALYSTS at SemEval-2026 task 13: Integrating transformer semantics and AST-derived structural features for code classification. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Talukder Naemul Hasan Naem Jerin Romijah Tuli, MD. Sartaj Alam Pritom. 2026. Mind_Flayer at SemEval-2026 task 13: Calibration-aware ensemble routing for cross-language AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Jonathan Katzy, Razvan Mihai Popescu, Arie van Deursen, and Maliheh Izadi. 2025. The Heap: A contamination-free multilingual code dataset for evaluating large language models. In *IEEE/ACM Second International Conference on AI Foundation Models and Software Engineering*, Forge@ICSE '25, pages 151–155, Ottawa, ON, Canada. IEEE.

Ryuto Koike, Masahiro Kaneko, and Naoaki Okazaki. 2024. OUTFOX: LLM-generated essay detection through in-context learning with adversarially generated examples. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 21258–21266. AAAI Press.

Linda Kupfer, Lisa Hader, Christian Jaumann, and Annemarie Friedrich. 2026. WWTC@UniA at SemEval-2026 task 13: BERT-based code authorship detection and qualitative analysis. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. 2023. TACO: topics in algorithmic code generation dataset. *CoRR*, abs/2312.14852.

Yongqiang Ma, Jiawei Liu, Fan Yi, Qikai Cheng, Yong Huang, Wei Lu, and Xiaozhong Liu. 2023. AI vs. human–differentiation analysis of scientific content generation. *arXiv preprint arXiv:2301.10416*.

Dung Nguyen Manh, Nam Le Hai, Anh T. V. Dau, Anh Minh Nguyen, Khanh Nghiem, Jin Guo, and Nghi D. Q. Bui. 2023. The vault: A comprehensive multilingual dataset for advancing code understanding and generation.

Team Meta. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Microsoft. 2024. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*.

Ruwad Naswan, Dipit Saha, Md. Rafat Kabir, and Nabiha Tahseen. 2026. ASTraNet at SemEval-2026 task 13: Not all code looks the same: Multi-view structural and semantic detection of machine-generated code. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

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. *J. Syst. Softw.*, 214:112059.

Henrique Gomes Nunes, Eduardo Figueiredo, Larissa Rocha Soares, Sarah Nadi, Fischer Ferreira, and Geanderson E. dos Santos. 2025. Evaluating the effectiveness of LLMs in fixing maintainability issues in real-world projects. *CoRR*, abs/2502.02368.

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.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026. 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*, Rabat, Morocco. 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 31263–31289, Suzhou, China. Association for Computational Linguistics.

Avi Patel, Manthan Laddha, Pushti Sapovadiya, Pruthwik Mishra, and Shrikant Malviya. 2026. Transformertrio at SemEval-2026 task 13: Navigating domain shift and representation instability in machine-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Indraneil Paul, Haoyi Yang, Goran Glavaš, Kristian Kersting, and Iryna Gurevych. 2025. ObscuraCoder: Powering efficient code LM pre-training via obfuscation grounding. In *The Thirteenth International Conference on Learning Representations*.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2025. Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions. *Commun. ACM*, 68(2):96–105.

Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks.

Team Qwen. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Musfiqur Rahman, SayedHassan Khatoonabadi, Ahmad Abdellatif, and Emad Shihab. 2024. Automatic detection of LLM-generated code: A case study of Claude 3 Haiku. *CoRR*, abs/2409.01382.

Yuvan Ramesh and Tongtong Wu. 2026. Yuvan at SemEval-2026 task 13: Multi-strategy ensemble approaches for AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Oarisa Rebayet and Radiul Walee. 2026. CSMetro at SemEval-2026 task 13: A feature-driven weighted ensemble for AI-generated code detection. In *Proceedings of the 20th Int. Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Lekkala Sai Teja, Kshitij Patiyal, Sangam Sai Anish, Annepaka Yadagiri, and Partha Pakray. 2026. CodeDetNITS at SemEval-2026 task 13: AI code authorship detection beyond truncation. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Nahid Niyaz Shovon and MD. Naim Parvez. 2026. Team_Omega at SemEval-2026 task 13: Frozen vs. trainable representations for out-of-distribution AI-generated code detection: A CodeBERT fine-tuning study. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Parthib Sarkar Sruthi Santhanam and Yashvardhan Sharma. 2026. Stylometry at SemEval-2026 task 13: Clustered stylometric modeling for machine-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Team StarCoder. 2023. Starcoder: may the source be with you!

D. Thenmozhi, M. Monesh, S. K. Paveethiran, K. S. Mayur, and K. Chandraprakash. 2026. SSN-CSE-CodeBrothers at SemEval-2026 task 13: Authorship attribution of LLM-generated code using fine-tuned CodeBERT. In *Proceedings of the 20th Int. Workshop*

*on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

S Udaythalavesh, Rajalakshmi Sivanaiah, and Angel Deborah S. 2026. UdayThalavesh S at SemEval-2026 task 13: Fine-tuned CodeBERT with balanced sampling and adaptive threshold optimization for multi-language AI-generated code detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Andric Valdez, Emmanuel Ancona, Sebastián Bernardino, Helena Gomez-Adorno, Fazlourrahman Balouchzahi, and Fabian Herrera. 2026. Late-iimas at SemEval-2026 task 13: Evaluating GNNs, PLMs, LLMs, and stylometry for automatic code identification. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Veniamin Veselovsky, Manoel Horta Ribeiro, and Robert West. 2023. Artificial artificial artificial intelligence: Crowd workers widely use large language models for text production tasks. *CoRR*, abs/2306.07899.

Shiquan Wang, Ruiyu Fang, Shuangyong Song, Yongxiang Li, and Xuelong Li. 2026. TeleAI at SemEval-2026 task 13: Data-centric full-parameter fine-tuning with multi-level ensembling for code generation detection. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, Thomas Arnold, Chenxi Whitehouse, Alham Fikri Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. 2024a. Semeval-2024 task 8: Multigenerator, multidomain, and multilingual black-box machine-generated text detection. In *Proceedings of the 18th International Workshop on Semantic Evaluation*, SemEval 2024, Mexico, Mexico.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Chenxi Whitehouse, Osama Mohammed Afzal, Tarek Mahmoud, Toru Sasaki, Thomas Arnold, Alham Fikri Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. 2024b. M4: Multi-generator, multi-domain, and multi-lingual black-box machine-generated text detection. In *Proceedings of the 18th Conference of the European Chapter of the ACL (Volume 1: Long Papers)*, pages 1369–1407, St. Julian's, Malta. Association for Computational Linguistics.

Yuxia Wang, Artem Shelmanov, Jonibek Mansurov, Akim Tsvigun, Vladislav Mikhailov, Rui Xing, Zhuohan Xie, Jiahui Geng, Giovanni Puccetti, Ekaterina Artemova, Jinyan Su, Minh Ngoc Ta, Mervat Abassy, Kareem Ashraf Elozeiri, Saad El Dine Ahmed El Etter, Maiya Goloburda, Tarek Mahmoud, Raj Vardhan Tomar, Nurkhan Laiyk, Osama Mohammed Afzal, Ryuto Koike, Masahiro Kaneko, Alham Fikri Aji, Nizar Habash, Iryna Gurevych, and Preslav Nakov. 2025. GenAI content detection task 1: English and multilingual machine-generated text detection: AI vs. human. In *Proceedings of the 1stWorkshop on GenAI Content Detection (GenAIDetect)*, pages 244–261, Abu Dhabi, UAE. International Conference on Computational Linguistics.

Xiaodan Xu, Chao Ni, Xinrong Guo, Shaoxuan Liu, Xiaoya Wang, Kui Liu, and Xiaohu Yang. 2024. Distinguishing LLM-generated from human-written code by contrastive learning. *CoRR*, abs/2411.04704.

Elitsa Yotkova, Violeta Kastreva, Dimitar Dimitrov, Ivan Koychev, and Preslav Nakov. 2026. FMI_SU_Yotkova_Kastreva at SemEval-2026 task 13: Lightweight detection of LLM-generated code via stylometric signals. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

# A Appendix

# B Used Generators

In Table 7 we list all generators included in the shared task. In total, the dataset comprises 11 generator families: ten LLM families and human-written code. Each family is represented by multiple checkpoints spanning different parameter scales and instruction-tuning variants (e.g., base vs. instruct/chat), which increases both intra-family diversity and the overall difficulty of attribution. We also aim to support both locally deployable and API-based models (e.g., OpenAI's GPT models). The largest variety of generators is observed for the `Llama`, `Qwen`, and `Phi` families, reflecting their broad public availability and rapid model iteration. Overall, this selection aims to balance coverage of widely used code-capable LLMs with diversity in model size and alignment strategy, providing a realistic benchmark for detection and fine-grained generator family attribution.

# C Final Leaderboard

In Table 4, Table 5, and Table 6 we provide a final snapshot of the task leaderboards, showing submissions that surpassed the baseline. Subtask A attracted the highest number of participants, indicating strong interest in the binary detection environment. The performance range is substantial, with the top system achieving a near-perfect Macro F-score of 0.997, while the majority of teams cluster between 0.50 and 0.70. The median performance lies in the mid-0.50s, suggesting that although high scores are attainable, robust and generalizable solutions remain challenging.

Subtask B, on the contrary, appears considerably more challenging. The best-performing system achieved a Macro F-score of 0.508, and most teams scored between 0.30 and 0.45. The narrower score range and lower ceiling indicate that fine-grained distinctions required in this setting are harder to model effectively. Despite attracting fewer teams, Subtask B demonstrates tighter competition among mid-ranked systems, with relatively small differences that separate adjacent ranks.

Subtask C presents a different dynamic: despite participation comparable to Subtask B, top performance is substantially higher, with the leading team achieving 0.785 F-score and several systems exceeding 0.65, suggesting that this multi-class configuration is tractable.

| Model Family | Model |
|---|---|
| **Yi** | Yi-Coder-9B<br>Yi-Coder-9B-Chat<br>Yi-Coder-1.5B-Chat<br>Yi-Coder-1.5B |
| **GPT** | GPT-4o-mini<br>GPT-4o |
| **Qwen** | Qwen2.5-Coder-7B<br>Qwen2.5-Coder-7B-Instruct<br>Qwen2.5-Coder-1.5B-Instruct<br>Qwen2.5-Coder-32B-Instruct<br>Qwen2.5-72B-Instruct<br>Qwen2.5-Coder-1.5B<br>Qwen2.5-Coder-14B-Instruct |
| **Gemma** | codegemma-7b-it<br>codegemma-7b<br>codegemma-2b |
| **Deepseek** | deepseek-coder-6.7b-instruct<br>deepseek-coder-6.7b-base<br>deepseek-coder-1.3b-instruct<br>deepseek-coder-1.3b-base |
| **Granite** | granite-8b-code-instruct-4k<br>granite-8b-code-base-4k |
| **Llama** | Llama-3.1-8B-Instruct<br>Llama-3.2-3B<br>Llama-3.1-70B-Instruct<br>Llama-3.3-70B-Instruct<br>Llama-3.3-70B-Instruct-Turbo<br>Llama-3.2-1B<br>Llama-3.1-8B<br>CodeLlama-70b-Instruct-hf<br>CodeLlama-34b-Instruct-hf<br>CodeLlama-7b-hf |
| **Phi** | Phi-3-small-8k-instruct<br>Phi-3-mini-4k-instruct<br>phi-4<br>Phi-3-medium-4k-instruct<br>phi-2<br>Phi-3.5-mini-instruct |
| **Mistral** | Mistral-Small-24B-Instruct-2501 |
| **StarCoder** | starcoder2-15B<br>starcoder<br>starcoder2-7b<br>starcoder2-3b |

Table 7: Model families and their selected models used in our task.

However, similar to Subtask A, there is a noticeable performance tail, with lower-ranked systems dropping below 0.50.

Across all subtasks, the number of submissions per team varies widely, from single-entry systems to highly iterative teams that submit hundreds of runs, indicating various experimentation strategies. It is worth noting that too many entries may indicate that a team was trying to probe the leaderboard, and their final solution could be overfitting to the leaderboard situation rather than solving the task.

## D System Descriptions

**Team CSMetro.** Team CSMetro proposed a feature-driven ensemble for AI-generated code detection across languages. Their Advanced Feature Extractor computes 36 structural, stylistic, and token-level features, which are used to train multiple classifiers (L1/L2-regularized Logistic Regression, Random Forest, and Gradient Boosting), with linear models operating on scaled features and tree-based models on raw inputs. Final predictions are produced via weighted logit-level stacking to improve robustness across heterogeneous programming languages (Rebayet and Walee, 2026).

**Team SSN-CSE-CODECATALYSTS.** This team introduced a late-fusion neural architecture that combines deep semantic representations with handcrafted software-engineering features. CodeBERT generates 768-dimensional embeddings, which are concatenated with a 16-dimensional language embedding and a 7-dimensional handcrafted feature vector. The fused representation is processed by an MLP with Layer Normalization and Dropout to mitigate overfitting. This hybrid design bridges transformer-based semantics and explicit structural signals (J et al., 2026).

**Team Stylometry.** Team Stylometry developed a stylometric mixture-of-experts framework for machine-generated code detection. Code samples are represented using character-level TF-IDF features and structural attributes such as length, punctuation usage, and comment density. A clustering step partitions samples into stylistically homogeneous groups, and separate classifiers are trained per cluster using Naive Bayes and Logistic Regression. The final predictions are produced through adaptive threshold calibration conditioned on cluster and code-length distributions (Sruthi Santhanam and Sharma, 2026).

**Team SLS.** Team SLS used a hybrid semantic-structural approach integrating CodeBERT embeddings with Tree-sitter-based structural features. Language-agnostic statistics that include counts of functions, loops, classes, and imports are concatenated with contextual transformer embeddings to form a unified representation. A classification layer predicts human vs. machine labels, allowing the system to jointly capture semantic intent and structural regularities (Gorantla et al., 2026).

**Team Udaythalavesh S.** This team fine-tuned `microsoft/codebert-base` for binary AI-generated code detection. To handle large-scale data, they balanced training by randomly sampling 100K instances per class and limited inputs to 384 tokens. The model was trained for three epochs using AdamW (LR=$2 \times 10^{-5}$) on TPU with `torch_xla`. Classification thresholds were calibrated on validation data by sweeping decision thresholds between 0.25 and 0.75 to maximize Macro F1, and final predictions applied the calibrated threshold followed by label inversion (Udaythalavesh et al., 2026).

**Team Dream.** Team Dream used SALSA, which frames machine-generated code detection as a single-token structured classification. An instruction-tuned Qwen model outputs binary labels (0/1) in a fixed format, with predictions derived from the logits of the two class tokens in a single forward pass. The model is LoRA-tuned with label-token-only supervision, balanced sampling across language-label pairs, and conservative optimization (low learning rate, early stopping) to enhance out-of-distribution robustness (Berdichevsky et al., 2026).

**Team Mind Flayer.** Mind Flayer designed a language- and domain-aware transformer ensemble. UniXCoder and GraphCodeBERT were fine-tuned and fused via weighted logit-level combination to capture complementary semantic and structural signals. A lightweight language identifier distinguishes seen from unseen languages, and a domain classifier separates algorithmic from production code. Adaptive thresholding calibrates predictions under distribution shift, reducing overconfident AI predictions in out-of-distribution scenarios (Jerin Romijah Tuli, 2026).

**Team David.** Team David combined extensive language-agnostic feature engineering with gradient boosting, extracting 168 features spanning structural metrics (e.g., nesting depth, control flow density, cognitive complexity), character n-grams, whitespace and punctuation patterns, and micro-style cues such as operator spacing and brace placement. These features, including multiplicative and ratio-based interactions, were modeled with XGBoost optimized via Optuna, using Leave-One-Language-Out specialization for seen languages and isotonic calibration with percentile-based thresholding for final predictions.

**Team Wooo2002.** Team Wooo2002 implemented a reproducible fine-tuning pipeline for CodeBERT, GraphCodeBERT, and UniXcoder using only official data, comparing code-only inputs with AST-augmented representations formed by concatenating parsed node sequences with source tokens. They examined truncation strategies, token budgets, label smoothing, imbalance handling, and small ensembles, finding that encoder choice and token-budget configuration had the strongest impact, while AST augmentation provided mixed gains depending on token trade-offs and parsing consistency.

**Team Archaeology.** Team Archaeology fine-tuned pretrained code models (CodeBERT, GraphCodeBERT, UniXcoder, CodeT5+) with subtask-specific pipelines. For Subtask A, they used leave-one-language-out cross-validation with language-balanced sampling, stochastic code augmentations, and overlapping chunk aggregation with per-language threshold calibration to improve cross-language generalization; for Subtask B, they addressed class imbalance with weighted loss and preserved long-range context by packing head and tail segments into a single input sequence (Ispas and Nisioi, 2026).

**Team NUST CodeIntel.** Team NUST CodeIntel compared classical ML models over TF-IDF and stylometric features with LoRA-adapted pretrained transformers (CodeBERT, BERT-base, UniXCoder, RoBERTa). Although transformers achieved near-perfect in-distribution validation scores, they degraded under cross-domain evaluation, while a lightweight Logistic Regression over lexical features obtained the best test Macro-F1, indicating greater robustness to distribution shift (Ali and Fatima, 2026).

**Team CodeHunters.** Team CodeHunters fine-tuned three pretrained encoders (UniXCoder, CodeT5, and CodeBERT), ultimately selecting UniXCoder as their primary backbone. Code snippets were encoded using UniXCoder pooling to obtain fixed-size representations. For Subtask A, a binary head distinguished human-written from LLM-generated code, while for Subtask B the classifier was extended to 11 classes (Human + 10 LLM families). Predictions were generated via softmax over pooled embeddings and evaluated under both seen and unseen language/author conditions (Dumitru et al., 2026).

**Team Omega.** Team Omega proposed a CodeBERT-based classifier with strategic partial encoder defrost. Instead of full fine-tuning, they froze the first 11 transformer layers and adapted only the final layer using a reduced learning rate to mitigate catastrophic forgetting. The `[CLS]` embedding was passed through a residual pre-classifier and a deeper projection network with Layer Normalization and GELU. Training used differential learning rates with AdamW, class-weighted cross-entropy, and a OneCycleLR scheduler with cosine annealing (Shovon and Parvez, 2026).

**Team Königsberg.** Team Königsberg developed a resource-efficient hybrid framework combining handcrafted software metrics with frozen GraphCodeBERT embeddings. The feature branch captured stylistic, structural, and complexity cues, while the embedding branch provided semantic representations. Multi-stage feature selection using statistical ranking and permutation importance reduced redundancy. Final predictions used late fusion between the branches to combine surface and semantic signals with low computational overhead (Habib and Das, 2026).

**Team CodeDet-NITS.** Team CodeDet-NITS built an 11-way authorship detector by fine-tuning instruction-tuned Qwen2.5-Coder models (3B/7B) as sequence classifiers using QLoRA (4-bit quantization with LoRA adapters) for memory efficiency. To address long-context limitations, overlength snippets were split into overlapping 512-token windows and trained with the same label; at inference, window-level logits were aggregated by averaging to produce a single prediction per snippet (Sai Teja et al., 2026).

**Team SYSUpporter.** Team SYSUpporter fine-tuned **UniXcoder-base** for 11-way authorship classification (Human + 10 LLM families) in Subtask B. To address the 1024-token limit, they introduced language-aware truncation preserving head and tail segments with dynamic ratio sampling. Whitespace-style indicators (e.g., indentation type and trailing spaces) were injected as special tokens to capture generator-specific formatting patterns. Training used 5-fold stratified cross-validation with Focal Loss, Generalized Mean Pooling, and multi-sample dropout, with predictions averaged across folds (Chen and Zheng, 2026).

**Team CUET Luminaries 0227.** Team CUET Luminaries 0227 addressed Subtask A with a UniXcoder-based binary classifier augmented by structural and statistical feature fusion, incorporating supervised contrastive learning and adversarial language-invariant training to improve robustness under cross-language and cross-domain shift (Chowdhury and Faisal, 2026).

**Team SSN-CSE-CodeBrothers.** Team SSN-CSE-CodeBrothers fine-tuned **CodeBERT** for Subtask B authorship attribution and handled label imbalance via class-weighted cross-entropy, using AdamW and a linear warmup schedule (Thenmozhi et al., 2026).

**Team TransformerTrio.** Team TransformerTrio participated in Subtasks A, B, and C, comparing handcrafted stylistic features, pretrained transformers, and stacking ensembles; they used XGBoost and hybrid ModernBERT embeddings with adversarial validation for Subtask A, fine-tuned ModernBERT-Large for 11-way classification in Subtask B, and applied single-stage fine-tuning with logit-level XGBoost stacking for Subtask C (Patel et al., 2026).

**Team HyperparameterOmens.** Team HyperparameterOmens participated in Subtasks A and C, using a compact AST-based feature set extracted with Tree-sitter and classified via logistic regression for Subtask A to improve domain robustness, and comparing lexical, character-level, AST-based, and transformer models for Subtask C with a per-class weighted ensemble optimized using Optuna (Dmitry Sukhotin, 2026).

**Team MALTO.** Team MALTO addressed Subtasks B and C using a unified pipeline that combines Tree-sitter canonicalization, hard-negative mining, and a three-phase curriculum training strategy. Transformer representations from UniXCoder and CodeT5 were pooled and aggregated via weighted soft-voting across temporal checkpoints to sharpen decision boundaries and improve robustness (Arslan et al., 2026).

**Team Codexa.** Team Codexa built a 12-model ensemble of **UniXcoder-base** encoders with MLP classifiers for Subtask B, addressing class imbalance with Label-Distribution-Aware Margin (LDAM) loss. Final predictions were aggregated via hard voting to mitigate validation-test mismatch (Dervişoğlu and Site, 2026).

**Team FMI SU Yotkova Kastreva.** Team FMI SU Yotkova Kastreva proposed a lightweight stylometric pipeline for Subtask A based on length-normalized ratio features, including comment density and text-like content ratios derived from character n-gram modeling, with language-aware parsing via Tree-sitter and shallow classifiers combined with heuristics (Yotkova et al., 2026).

**Team MedHastra.** Team MedHastra combined stylometric feature engineering with transformer-based modeling in a hybrid detection framework. For Subtask A, they used TF–IDF and structural formatting features in a classical soft-voting ensemble; for Subtasks B and C, they fine-tuned CodeBERT for multi-class attribution with imbalance handling via downsampling and class-weighted losses (Chandrasekar et al., 2026).

**Team ICI Innolabs.** Team ICI Innolabs built a Subtask B ensemble of **CodeBERT, UniXcoder, and StarEncoder**, fine-tuned with stratified sampling and class-weighted cross-entropy with label smoothing to address imbalance. They mitigate sequence truncation via stochastic fixed-length window sampling during training and overlapping sliding windows at inference, averaging window-level logits per model and then across backbones for final prediction (Balmuș and Dura, 2026).

**Team WWTC@UniA.** Team WWTC@UniA fine-tuned ModernBERT-large with a 2048-token input window across Subtasks A, B, and C, leveraging extended context to retain full code, comments, and natural language descriptions without truncation. They applied weighted cross-entropy for imbalanced settings and standard fine-tuning otherwise, and complemented this with a zero-shot chain-of-thought LLM strategy for Subtask A to address cross-domain and cross-language generalization gaps (Kupfer et al., 2026).

**Team COODetect.** Team COODetect proposed an interpretable, multi-view framework combining lexical, structural, and symbolic indicators to improve out-of-distribution robustness. To counter language-specific bias, they apply unsupervised score normalization using Z-scoring and Gaussian Mixture Models, enabling latent domain adaptation and improved generalization to unseen languages without relying solely on deep encoder memorization (Creo et al., 2026).

**Team Farhan Nafis Rayhan.** Team Farhan Nafis Rayhan designed a multi-stage ensemble for Subtask B that decomposes code into two parallel views: human-like textual content (comments, identifiers, strings) and sanitized structural code with placeholders. These complementary representations are processed by multiple models and combined through a gated multiclass ensemble to improve robustness in authorship attribution (Farhan Nafis Rayhan, 2026).

**Team ASTraNet.** Team ASTraNet adopted a multi-view strategy combining **code-pretrained transformers** (CodeT5p, CodeBERT, UniXcoder) with contrastive heads and focal loss, alongside a Flow-Augmented AST pipeline using GNNs over canonicalized syntax trees. Their extended system further integrates TF-IDF features, stylometric signals, token statistics derived from LLM, LightGBM stacking, and class-priority post-processing in a layered ensemble (Naswan et al., 2026).

**Team LATE-IIMAS.** Team LATE-IIMAS systematically compared GNNs, pre-trained language models, large language models, and stylometric feature engineering with XGBoost across subtasks. Their framework evaluates structural graph representations, neural embeddings, and handcrafted stylistic signals under distribution shift, emphasizing cross-domain robustness and generalization analysis (Valdez et al., 2026).

**Team Vivek Dhayaal.** Team Vivek Dhayaal proposed GPAD, an enhanced **CodeBERT-based** architecture for Subtask B that integrates entropy-biased attention to emphasize uncertain token regions and incorporates explicit code-style features (e.g., line structure and complexity metrics) into the detection pipeline to capture generator-specific stylistic patterns (Dhayal, 2026).

**Team Osint.** For Subtask A, StarCoderBase-1B was fine-tuned using a long-context classification setup with stratified sampling and class-weighted optimization to better address class imbalance and distribution shifts. For Subtask B, two TinyStarCoder variants were trained: one on clean code using undersampling and class weighting, and another incorporating syntactic noising with a clean-to-noisy curriculum to enhance robustness in an imbalanced authorship attribution setting (Anand et al., 2026).

**Team Segmentation Fault.** Team Segmentation Fault participated in Subtask A with a CodeBERT-base classifier to distinguish human- and AI-generated code. After identifying generator overlap between training and validation data, which led to overfitting, they created a custom out-of-distribution validation split by reserving Llama, DeepSeek, and Yi families exclusively for validation. To improve generalization, they applied heavy data augmentation (variable renaming, comment stripping, spacing normalization), incorporated additional dropout and label smoothing, and used early stopping based on validation loss trends to mitigate overfitting.

## E Approaches Overview

Table 8 shows strong convergence around pre-trained transformer encoders, forming the backbone of most submissions across subtasks. Many teams combine neural representations with engineered lexical or structural features, making hybridization a primary robustness strategy. Classical classifiers and ensembles are used as aggregation layers on embeddings, suggesting innovation occurs mainly at the integration level rather than through novel architectures, while advanced training schemes remain rare.

| Team | Subtasks | UnixCoder | Modern-BERT | CodeBERT | RoBERTa | bert-base-uncased | Metric Learn. | LLM | LoRA | PseudoLbl | Curriculum | CodeT5+ | NomicAI | FeatEng | AdvFilter | LogitStack | TreeEns | MIL | LLMMarker | Focal | AST | LateFusion | ClassBal | TFIDF | LinearSVM | Multinomial NB | KNN | LogReg | LabelFlip |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Alejandro Mosquera | A | ✓ | | | | | | | ✓ | ✓ | ✓ | | | ✓ | | | | | | ✓ | | | | | | | | | |
| CSMetro | A | | | | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | | | | | | | | | | |
| CUET_Luminaries_0227 | A | ✓ | | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | |
| CoDetect | A | | | | | | | | | | | | | ✓ | | | | | | | | | | | | | | | |
| CodeHunters | A/B | ✓ | | ✓ | | | | | | | | ✓ | | | | | | | | | | | ✓ | | | | | | |
| David | A | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | |
| Dream | A | | | | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | |
| FMI_SU_Yotkova_Kastreva | A | | | ✓ | | | | ✓ | | ✓ | | ✓ | ✓ | ✓ | | | | | | | | | | | | | | | |
| Königsberg | A | ✓ | | ✓ | | | | | | | | | | ✓ | | | | | | | | | | | | | | | |
| Mind_Flayer | A | ✓ | | ✓ | | | | | | | | | | | | ✓ | | | | | ✓ | | | | | | | | |
| Osint | B | | | ✓ | | | | | | | | | | | | ✓ | | | | | | | | | | | | | |
| SSN-CSE-CODECATALYSTS | A | | | ✓ | ✓ | | | | | | | | | ✓ | | | | | | ✓ | ✓ | ✓ | ✓ | | | | | | |
| Stylometry | A | | | | | | | | | | | | | ✓ | | ✓ | | | | | ✓ | | ✓ | | | | | | |
| Team_SLS | A | | | ✓ | | | | ✓ | | | | | | ✓ | | | | | | | | | | | | | | | |
| TransformerTrio | A/B/C | ✓ | ✓ | | | | | | | | | | | ✓ | ✓ | | ✓ | | | | | | | | | | | | |
| UIT_AMMC | A | | | | | | | ✓ | ✓ | | | | | ✓ | ✓ | | | ✓ | | | ✓ | | ✓ | | | | | | |
| Udaythalavesh S | A | | | ✓ | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | | | |
| YoungDSMLKZ | C | ✓ | ✓ | | | | | | | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | | |
| Yuvan | A/B/C | | | ✓ | | | | ✓ | ✓ | | | | | ✓ | | ✓ | ✓ | | | | ✓ | ✓ | ✓ | | | | | | ✓ |
| Archaeology | A/B | ✓ | | ✓ | | | | | | | ✓ | | | | | | | | ✓ | | | | | | | | | | |
| NUST-CodeIntel | A | ✓ | ✓ | ✓ | ✓ | | | ✓ | | | | ✓ | | | ✓ | | | | ✓ | ✓ | | | | | | | | | |
| Team_Omega | A | | | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | |
| CodeDet-NITS | B | | | | | | ✓ | ✓ | | | | ✓ | | | | | | | | | | | | | | | | | |
| Code_detect | A | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | |
| SSN-CSE-CodeBrothers | A/C | | | ✓ | ✓ | ✓ | | ✓ | | | | | | | | | | | | | | | ✓ | ✓ | | | | ✓ | |
| Segmentation Fault | A | ✓ | | ✓ | ✓ | | | | | | | | | | | | | | | | | | ✓ | | | | | ✓ | |
| HyperparameterOmens | A/B/C | | ✓ | ✓ | | | | ✓ | | | | ✓ | ✓ | | | | | ✓ | | | ✓ | | | ✓ | ✓ | | ✓ | ✓ | |
| MALTO | C | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | | ✓ | | | | | | | | ✓ | ✓ | | ✓ | ✓ | |
| SYSUpporter | B | ✓ | | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | ✓ | | | ✓ | | | | ✓ | | | | | |
| CLRG | A | | | | | | | | | | | ✓ | | ✓ | | | | | | | ✓ | ✓ | | ✓ | | | | ✓ | |
| Farhan Nafis Rayhan | B | ✓ | ✓ | | | | ✓ | | ✓ | | | | | ✓ | | ✓ | ✓ | | | ✓ | | ✓ | ✓ | ✓ | | | | | |
| Codexa | B | ✓ | | ✓ | | | ✓ | | | | | | | ✓ | | | ✓ | | | ✓ | ✓ | | ✓ | ✓ | | | | | |
| WWTC@UniA | A/B/C | | ✓ | ✓ | ✓ | | | ✓ | | | | | | | | | | | | ✓ | | | | | | | | | |
| ASTraNet | A/C | ✓ | | ✓ | | | ✓ | | | | | ✓ | | ✓ | | ✓ | | | ✓ | ✓ | ✓ | | ✓ | | | | ✓ | ✓ | |
| MedHastra | A/B/C | | | ✓ | | | | | | | | | | ✓ | | ✓ | ✓ | | | | | | | ✓ | | | | ✓ | |
| ICI Innolabs | B | ✓ | | ✓ | | | | | | | | | | | | ✓ | | | | | | | | ✓ | | | | | |
| LATE-IIMAS | A/B/C | | | ✓ | | | | ✓ | | | | | | ✓ | | | | | | | | | | ✓ | | | | | |
| Team_Vivek Dhayaal | A | | | ✓ | | | | | | | | | | | | | | | | | ✓ | | | | | | | | |

Table 8: **Methods used by teams across subtasks.** Subtasks indicate participation (A/B/C). A checkmark means the team reported using the method in at least one of the subtasks they entered.