HyperparameterOmens at SemEval-2026 Task 13: Various Approaches to Detecting Machine-Generated Code

Dmitry Sukhotin
University of Tübingen
dmitry.sukhotin@student.uni-tuebingen.de

How Chuen Yu
University of Tübingen
how-chuen.yu@student.uni-tuebingen.de

Abstract

We present our systems for SemEval-2026 Task 13 (Orel et al., 2026) (Detecting Machine-Generated Code), built on the Droid resource suite and benchmark setting (Orel et al., 2025). For Subtask A (binary classification of human-written vs. machine-generated code), lexical baselines such as TF–IDF and character n-grams failed to generalize from the LeetCode training distribution to the production-code evaluation split. After correcting pipeline errors that obscured true performance and selecting stable AST features under domain shift, our final system uses 5 uncorrelated features and achieves 0.57 macro F1 on the public test set. For Subtask C (4-way authorship classification of human, AI, hybrid, and adversarial), lexical baselines collapsed under vocabulary shift (55% F1 drop), while deep semantic models achieved near-zero degradation. A per-class weighted ensemble achieved 0.85 macro F1 on the 1K public sample, with a final score of 0.57 on the full test set.

1 Introduction

For Subtask A, the central challenge is extreme domain shift: training data consists of LeetCode-style algorithmic solutions, while evaluation data contains production code and research implementations with fundamentally different structure, intent, and linguistic coverage (including languages unseen during training), as formalized in the Droid benchmark underlying the task (Orel et al., 2025). In the broader SemEval landscape, this setting extends recent shared-task work on machine-generated content detection to source code (Wang et al., 2024).

Initial lexical baselines (TF–IDF and character n-grams) performed poorly (0.37–0.41 F1), suggesting reliance on vocabulary-specific patterns that did not generalize, consistent with shortcut-learning behavior under distribution shift (Geirhos et al., 2020).

During the transition to AST-based features, we also found a parsing error that assigned default labels to 49.4% of test samples, obscuring true model performance. After refactoring the pipeline to enforce train-test parity, we developed a diagnostic framework to identify unstable features, specifically those with predictive sign flips across domains or high within-class distribution shift. This two-stage selection process removed unstable and then redundant features, reducing the set from 25 to 5. As a result, out-of-domain F1 improved from 0.34 to 0.57, while the validation-test gap narrowed from 0.48 to 0.13.

Subtask C, on the other hand, initially did not involve such a major domain shift; the training and test data were compositionally similar in terms of label proportions, languages, lengths, etc. Instead, the difficulty lies in the inclusion of "mixed" code (human-AI hybrid and adversarial AI) in four-way classification, as it contains traits characteristic of both human and AI code.

Bag-of-words approaches fared poorly because of a 41.7% vocabulary shift between training and test. In contrast, deep semantic models performed well because they could capture contextual patterns. All approaches struggled with hybrid human-AI code, partly because the training data contained only a very low percentage of such samples.

Our systems draw on four families of methods with established use in text and code classification. TF–IDF with logistic regression and Naive Bayes-SVM (NBSVM) serve as robust baselines, effective across sentiment and topic classification on both short and long texts (Wang and Manning, 2012). Character-level convolutional networks (CharCNN) bypass vocabulary by operating on raw character sequences, achieving competitive results on million-scale datasets (Zhang et al., 2015). AST-based structural features follow the code stylometry tradition of Caliskan-Islam et al. (2015), who showed that syntactic features are more ro-

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 1372–1378

bust against obfuscators than lexical ones, achieving 94% accuracy across 1,600 authors. CodeBERT (Feng et al., 2020) and GraphCodeBERT (Guo et al., 2020) are transformers pre-trained on multi-language code corpora; GraphCodeBERT additionally incorporates data flow between variables, achieving state-of-the-art on code search, clone detection, and refinement.

2 Task Data and Constraints

2.1 Subtask A data and domain shift

All experiments use only the officially provided data splits (Orel et al., 2025), following task constraints prohibiting external AI-detection tools. Subtask A provides 500,000 training examples, 100,000 validation examples, and 500,000 test examples. Table 1 summarizes key differences between the training/validation data and the test data.

Attribute	Train/Validation	Test
Code type	LeetCode algorithmic	Production/research
Languages	3	8
Avg. length	36 lines, 122 tokens	43 lines, 132 tokens

Table 1: Domain shift in Subtask A.

Training and validation contain algorithmic code in three languages (91% Python, 5% C++, 4% Java), whereas the test set contains production/research code in eight languages, including five languages not present in training.

2.2 Evaluation protocol

We report performance on three splits: (i) validation (`validation.parquet`, in-domain), (ii) public test (initial leaderboard evaluation), and (iii) private test (final ranking). We use the labeled 1,000-sample file (`test_set_sample.parquet`) only for diagnostic analysis, never for training or model selection.

3 Subtask A System

3.1 Lexical approaches

We began by treating code as text and training standard token- and character-level classifiers. Table 2 reports macro F1 on the in-domain validation split and the public test split.

Model	Val F1	Test F1
NBSVM	0.56	0.41
TF–IDF + LogReg	0.56	0.37
Char n-gram + LogReg	0.56	0.32
Char CNN	0.38	0.23
Comment TF–IDF	0.54	0.22

Table 2: Lexical baselines for Subtask A.

All lexical approaches degrade substantially under domain shift, with validation-to-test absolute F1 drops ranging from 15% (NBSVM) to 31% (Comment TF–IDF). These models appear to rely on lexical patterns that do not transfer reliably from LeetCode-style training data to production/research code.

We therefore moved from lexical to structural features to capture syntax-level patterns that are less tied to surface vocabulary.

3.2 AST-based representation and parsing pipeline

We switched to Abstract Syntax Tree (AST) representations using `tree-sitter` parsers.[1] Each code snippet was parsed into a tree of syntactic nodes, and node-type names were normalized across languages to obtain a language-agnostic structural representation for feature extraction.

To control parse quality, we used two acceptance gates: a score threshold (score $=$ node_count $-3\times$ error_count ≥ 5) and a maximum error-rate threshold (≤ 0.3). An early version additionally included a trivial-tree gate (`child_count < 2`), which rejected 12.4% of training samples but 49.4% of test samples (Table 3), with rejected samples defaulting to "Human" and artificially inflating scores. After removing it, coverage reached 99.9%+ across all splits (Table 4), with the score gate rejecting fewer than 0.03% of samples on any split.

Split	Total	OK	REJECT_TRIVIAL
train	500K	87.6%	12.4%
test_sample	1,000	50.6%	49.4%

Table 3: Parsing statistics before removing `REJECT_TRIVIAL`.

[1] https://tree-sitter.github.io/tree-sitter/

Split	Total	OK	LOW_SCORE
train	500K	99.97%	0.03%
validation	100K	99.98%	0.03%
test_sample	1,000	100%	0.00%

Table 4: Parsing statistics after removing `REJECT_TRIVIAL`.

Our initial implementation achieved 0.64 validation F1, substantially above the lexical baselines. We then proceeded with feature extraction and domain-shift diagnostics using the corrected pipeline.

To assess threshold sensitivity, we evaluated macro F1 and parse coverage on the 1,000-sample labeled test set across a range of score and error-rate thresholds. Results were highly stable: varying `min_parse_score` from 0 to 10 produced identical coverage (100%) and F1 (0.5722), with only the extreme value of 20 rejecting a single sample. Varying `max_error_rate` from 0.2 to 1.0 had no effect on either metric. Tightening to 0.1 rejected 28 samples with a marginal F1 change of $+0.0008$, but introduced systematic label bias: 19 of 28 rejected samples were human-written, as production code contains malformed syntax, license headers, and incomplete snippets that inflate parser error counts, while AI-generated code is syntactically clean by construction. Defining precision/recall against a parse-quality gold standard is not straightforward in this setting, as no such annotation exists; however, the low rejection rates and F1 stability suggest the gates are not a significant source of error.

3.3 Initial results with 25 AST features

We extracted 25 AST-based features, which we organize into five conceptual groups: structural complexity, node-category frequencies, per-function ratios, diversity metrics, and compression/style features. This follows the general motivation of code stylometry work that uses structural code signals to distinguish coding behavior (Caliskan-Islam et al., 2015).

These features were designed to capture syntactic organization, local structural variation, and code regularity across languages.

A class-balanced logistic regression model with standardized features performed well in-domain but degraded sharply out-of-domain (Table 5).

Split	Macro F1	Samples	Val-Test Gap
Training	0.89	500K	–
Validation	0.90	100K	–
Test	0.38	500K	0.52

Table 5: Initial AST logistic regression performance.

The 0.52 validation-test gap, despite 99.9% parsing coverage, shows that structural features can still overfit to training-domain regularities. Features that are predictive on LeetCode-style algorithmic code may not transfer to production/research code. This motivated the feature diagnostics described next.

3.4 Feature diagnostics

To identify features that were predictive in-domain but unstable under domain shift, we used two diagnostics, framing the problem as feature-level instability under dataset shift (Subbaswamy et al., 2022).

Sign flip detection. A feature exhibits a sign flip if its association with the AI label reverses across domains:

$$\text{sign}\left(\mathbb{E}[f \mid \text{AI}] - \mathbb{E}[f \mid \text{Human}]\right)_{\text{train}} \neq \text{sign}\left(\mathbb{E}[f \mid \text{AI}] - \mathbb{E}[f \mid \text{Human}]\right)_{\text{test}}$$

Sign-flip features reverse their directional association with the label across domains and are therefore unreliable for cross-domain transfer.

Wasserstein distance. For each feature, we measured within-class distribution shift between training and test (Human and AI separately) using Wasserstein distance. High drift (> 1.5) indicates that a feature changes substantially across domains even within the same label, suggesting sensitivity to domain-specific structure rather than authorship.

Results. The diagnostics flagged 15 of 25 features as unstable: 10 with sign flips and 5 with high within-class drift but no sign flip. The unstable set included depth/branching features, several node-frequency and ratio features, and both compression features (`lzma_ratio`, `zlib_ratio`).

3.5 Two-stage feature ablation

We used ablation to test whether the diagnostics identified features that harmed cross-domain transfer.

Stage 1: Remove unstable features. We retrained logistic regression after removing the 15 features flagged by sign-flip or high-drift diagnostics, retaining 10 features (Table 6).

Stage 2: Remove correlated features. Among the remaining 10 features, we removed one feature from each pair with $|r| > 0.6$, yielding a final set of 5 uncorrelated features.

Configuration	Features	Val F1	Test F1	Gap
All features	25	0.90	0.38	0.52
Remove unstable	10	0.77	0.55	0.22
Remove correlated	5	0.70	0.57	0.13

Table 6: Feature ablation results.

Removing unstable features produced the largest gain in out-of-domain performance (+0.17 F1), while the correlation-pruning stage further improved test F1 (+0.02) and reduced the validation–test gap.

3.6 Final feature set and interpretation

The final model uses five structural features that remained stable across domains after diagnostic filtering and correlation pruning:

- `avg_depth`: mean AST depth (overall nesting depth).
- `branching_std`: variability in branching factor (irregularity of local tree structure).
- `freq_function_def`: frequency of function-definition nodes.
- `calls_per_func`: function-call density normalized by the number of functions.
- `unique_bigrams`: diversity of adjacent AST node-type pairs.

In our data, these features showed no sign flips between train and test, low within-class drift (Wasserstein distance < 0.30), and low pairwise correlation ($|r| < 0.30$).

3.7 Subtask A results and analysis

Table 7 reports performance across all evaluation splits, including the private test set used for final competition ranking.

Split	Macro F1	Samples
Training	0.89	500K
Validation	0.90	100K
Public test	0.57	500K
Private test	0.56	500K

Table 7: AST logistic regression performance.

The public-to-private degradation of only 1.9% (from 0.57 to 0.56) confirms that our feature selection process successfully identified transferable patterns.

3.8 Post-submission ensemble analysis

After the submission deadline, we explored whether combining the AST logistic regression with the lexical and statistical baselines could further improve performance. We optimized ensemble weights on the 1,000-sample labeled test set, treating it as a diagnostic experiment rather than a tuning set. The weighted ensemble of all five models (AST, XGBoost, TF-IDF, statistical, and character n-gram) achieved 0.62 macro F1, a +0.05 improvement over the AST model alone (0.57). AST received the highest weight (0.42), followed by XGBoost (0.30) and TF-IDF (0.20), suggesting that lexical features contribute complementary signal despite failing catastrophically as standalone models. We note that since Subtask A public-to-private degradation was only 1.9% for the AST model, this result is likely indicative of true test performance rather than an artifact of the diagnostic sample.

4 Subtask C System

4.1 Task characteristics and domain analysis

Subtask C is a four-way authorship classification problem. Code is labeled as human-generated, AI-generated, hybrid (partially human/AI), or adversarial (AI-generated code trying specifically to mimic a human style). Additionally, the mixed authorship classes (hybrid and adversarial) are underrepresented in training data (9.4% and 13.1% respectively). Finally, like Subtask A, there is an uneven distribution of languages in the training data, with some being very over or under represented.

4.2 Lexical baselines under vocabulary shift

Our first lexical-based approach failed because models memorized problem-specific tokens rather than authorship patterns. Additionally, the vocabulary shift from training to test, with 41.7% of test tokens being absent from training, proved insurmountable for these models. Table 8 shows validation and test performance for lexical models.

Model	Val F1	Test F1	Drop
TF-IDF + LogReg	0.6207	0.2787	-55%
NBSVM	0.6136	0.2723	-56%

Table 8: Lexical baselines for Subtask C (test on 1K labeled sample). Both models degrade severely under vocabulary shift (55–56% drop).

4.3 A vocabulary-independent approach

Character-level patterns (Zhang et al., 2015) were better able to capture generator signatures: formatting consistency, naming conventions, comment patterns, and syntactic sugar usage. They were also more generalizable across problem domains and language choices, and unlike the lexical models, were not affected by the out-of-vocabulary (OOV) issue.

Model	Val F1	Test F1	Drop
CharCNN	0.7452	0.7432	-0.3%

Table 9: Character-level model for Subtask C (test on 1K labeled sample). Minimal degradation demonstrates vocabulary independence.

4.4 Statistical feature models: Moderate degradation

AST-based statistical models were a notable improvement over lexical ones, but fall short compared to CharCNN and the BERT-based models (next section). This suggests that, with minimal structural domain shift, character-level and semantic patterns are more generalizable than engineered structural features.

Model	Val F1	Test F1	Drop
XGBoost (AST features)	0.6074	0.5689	-6.3%
LightGBM (AST features)	0.5859	0.5547	-5.3%

Table 10: Statistical models using AST-based features (test on 1K labeled sample). Moderate degradation suggests partial generalization.

4.5 Robust generalization with deep semantic models

Pre-trained transformer models achieve near-zero degradation, with GraphCodeBERT dropping only 0.04% from validation to test. Two properties explain this robustness: Byte-pair encoding subword tokenization decomposes novel tokens into known subwords, sidestepping the OOV problem that seriously hampered the lexical models; and large-scale pretraining over diverse code corpora exposed these models to authorship-level stylistic variation across problem domains and languages, rather than vocabulary-specific surface patterns.

Model	Val F1	Test F1	Drop
CodeBERT	0.8500	0.8324	-2.1%
GraphCodeBERT	0.8318	0.8315	-0.04%

Table 11: Deep semantic models for Subtask C (test on 1K labeled sample). Near-zero degradation demonstrates robust cross-domain generalization.

4.6 Final approach: Per-class ensemble approach

Given the complementary robustness profiles shown in Tables 9–11, our final submission combines all seven models in a per-class weighted ensemble, where weights are optimized independently for each of the four authorship classes using Optuna (Akiba et al., 2019) with 15,000 trials per class. This allows the ensemble to exploit each model's class-specific strengths rather than applying a single global weighting.

The ensemble achieves 0.8521 macro F1 and 0.9010 accuracy, a +2.4% improvement over the best single model (CodeBERT). Gains are largest on the difficult mixed-authorship classes — hybrid (+3.9%, 0.80 F1) and adversarial (+1.3%, 0.80 F1) — while human and machine see minimal improvement (+0.9% and 0.0%), as individual models already approach ceiling performance on those classes. Cross-language F1 ranges from 0.67 (C) to 0.90 (Python), indicating the ensemble preserves the language-agnostic generalization of the underlying transformers.

5 Subtask C Results and Analysis

5.1 Model robustness hierarchy

Our results establish a clear hierarchy of approaches under vocabulary shift with minimal structural shift. Table 12 shows the validation-to-test degradation for each approach.

Type	Models	Drop
Semantic	GraphCodeBERT, CodeBERT	0–2%
Character	CharCNN	0.3%
AST	XGBoost, LightGBM	5–6%
Lexical	TF-IDF, NBSVM	55%

Table 12: Model robustness under vocabulary shift (val→1K test drop).

5.2 Mixed-authorship and ensemble

In the case of the hybrid and adversarial classes, mixed stylistic signals and data scarcity (9.4% hybrid, 13.1% adversarial in training) proved challenging for all models, with hybrid and adversarial averaging 0.61 and 0.58 F1 respectively compared to human and AI at 0.84 and 0.86 F1. A per-class weighted ensemble proved effective in addressing this issue.

5.3 Final results with full test set

Despite achieving 0.8521 on the 1000-sample public test set, the final score on the full test set was 0.5722. It seems evident that Subtask C has a domain shift issue similar to Subtask A, but between training/1K test and full test. Given the near-zero degradation in the 1K sample, we attribute the drop primarily to language imbalance in training and distribution shift in the full set, and not a failure of the vocabulary-independent strategy.

6 Conclusion

We presented our systems for SemEval-2026 Task 13, participating in Subtasks A and C.

For Subtask A, we addressed severe structural domain shift (LeetCode → production code). AST-based features proved most robust, achieving 0.57 macro F1 after systematic feature selection. Lexical baselines failed catastrophically (0.37–0.41 F1), demonstrating that structural signals transfer better than token-level patterns under this type of shift.

For Subtask C, we were confronted by more complex classifications and a semantic shift. Deep semantic models (CodeBERT, GraphCodeBERT) achieved robust generalization (0.83 F1, <0.5% degradation), while lexical models failed even more severely than in Subtask A (55% performance collapse). Character-level models provided a middle ground (0.74 F1, <1% degradation), demonstrating vocabulary-independent learning.

Key insights:

1. **The type of domain shift determines model robustness**: Structural shifts challenge all models but favor AST features; vocabulary shifts selectively destroy lexical models while barely affecting semantic/character approaches.

2. **Pre-trained transformers are essential for robust detection**: GraphCodeBERT's -0.04% degradation demonstrates that semantic understanding generalizes across problem domains, languages, and vocabulary distributions.

3. **Mixed authorship remains challenging**: All models struggle with hybrid (0.61 avg F1) and adversarial (0.58 avg F1) classes, likely due to data scarcity (9.4% and 13.1% of training) and inherent ambiguity.

4. **Pipeline correctness is critical**: Our Subtask A results improved from 0.34 to 0.57 F1 after fixing parsing issues, emphasizing the importance of rigorous validation.

Overall, our results suggest that **semantic generalization** is the key to robust AI-generated code detection. While hand-engineered features (AST, statistical) provide value for specific types of domain shift, pre-trained language models capture abstract authorship patterns that transfer reliably across diverse evaluation scenarios. Future work should focus on improving hybrid/adversarial detection through data augmentation and adversarial training techniques.

References

Takuya Akiba, Shotaro Sano, Toshiki Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2623–2631.

Aylin Caliskan-Islam, Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, Fabian Yamaguchi, and Rachel Greenstadt. 2015. De-anonymizing programmers via code stylometry. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 255–270. USENIX Association.

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

1536–1547. Association for Computational Linguistics.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673.

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

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. 2025. 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.

Adarsh Subbaswamy, Bryant Chen, and Suchi Saria. 2022. A unifying causal framework for analyzing dataset shift-stable learning algorithms. *Journal of Causal Inference*, 10(1):64–89.

Sida Wang and Christopher Manning. 2012. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94, Jeju Island, Korea. Association for Computational Linguistics.

Yuxia Wang, Jonibek Mansurov, Petar Ivanov, Jinyan Su, Artem Shelmanov, Akim Tsvigun, Osama Mohammed Afzal, Tarek Mahmoud, Giovanni Puccetti, and Thomas Arnold. 2024. SemEval-2024 task 8: Multidomain, multimodel and multilingual machine-generated text detection. In *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 2057–2079, Mexico City, Mexico. Association for Computational Linguistics.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 649–657.

A AST Feature Definitions

Features are extracted from the normalized AST produced by Tree-sitter.[2] Node types are normalized to language-agnostic categories (e.g., `FUNCTION_DEF`, `CALL`, `LOOP`) before extraction. The five features used in the final model are:

avg_depth. Mean depth of all nodes in the AST, where the root has depth 0. Computed as $\frac{1}{|N|}\sum_{n \in N} d(n)$, where N is the set of all nodes and $d(n)$ is the depth of node n.

branching_std. Standard deviation of the number of children across all nodes. Captures irregularity in local tree structure.

freq_function_def. Frequency of `FUNCTION_DEF` nodes, computed as $\frac{|\{n : \text{type}(n) = \texttt{FUNCTION\_DEF}\}|}{|N|}$. A `FUNCTION_DEF` node is any top-level or nested function or method definition as identified by the language grammar; lambdas are not counted as they map to a different normalized type.

calls_per_func. Function-call density normalized by function count: $\frac{|\{n : \text{type}(n) = \texttt{CALL}\}|}{\max(|\{n : \text{type}(n) = \texttt{FUNCTION\_DEF}\}|, 1)}$. The denominator is clamped to 1 to avoid division by zero in files with no function definitions (e.g., scripts).

unique_bigrams. Count of distinct adjacent node-type pairs (t_i, t_j) where t_j is a direct child of t_i in the AST. Measures the diversity of local syntactic patterns.

[2] https://tree-sitter.github.io/tree-sitter/