LAFED at SemEval-2026 Task 13: Language-Agnostic Feature Engineering for Cross-Lingual AI-Generated Code Detection

Juan David Villate
Independent Researcher
jdvillate@hotmail.com

Abstract

Robust detection of AI-generated source code across programming languages remains challenging due to language-specific cues and train–test distribution shifts. We present **LAFED** (*Language-Agnostic Feature Engineering Detector*), a feature-engineering approach trained on {Python, Java, C++} and evaluated on a multilingual test set that includes unseen languages {C, C#, Go, JavaScript, PHP}. LAFED combines (i) structural *skeletal* features (indentation, control-flow density, and approximations of McCabe/Halstead complexity), (ii) character and whitespace statistics inspired by stylometry, and (iii) micro-style patterns (operator spacing, blank lines, indentation consistency). Using XGBoost (Chen and Guestrin, 2016) with Optuna hyperparameter search (Akiba et al., 2019), our best model achieves **macro-F1=0.7570** on the organizer-provided 1,000-sample test set for reproducible experiments, while the official hidden evaluation yields **macro-F1=0.75209** (**5th** place in Subtask A). Per-language analysis shows strong transfer to C# (0.7753) and JavaScript (0.7683), but weaker performance on Go (0.6400) and PHP (0.5238).

1 Introduction

SemEval-2026 Task 13 focuses on detecting AI-generated source code across multiple programming languages and code generators; we refer to the task paper for dataset details and the official evaluation protocol (Orel et al., 2026). Cross-language detection is particularly challenging because many lexical cues identify the programming language rather than the origin (human vs. AI), yielding brittle detectors under language shift.

Core hypothesis. We hypothesize that cross-language generalization is driven primarily by *language-invariant* signals (structure, character-level patterns, and micro-style), rather than by post-hoc domain adaptation over language-specific representations.

Contributions.

- We design language-agnostic features spanning skeletal structure, character/whitespace stylometry, and micro-style patterns.
- We build an end-to-end pipeline (extraction, interactions, normalization, leakage-aware selection, Optuna optimization) and report cross-language results with per-language analysis and ablations.

The full code, including the feature-extraction pipeline, is available at: github.com/DavidVilem/LAFED-SemEval2026.

2 Related work

Our approach connects (i) classic software metrics such as cyclomatic complexity (McCabe, 1976) and Halstead-style measures (Halstead, 1977), (ii) character-level stylometry for authorship attribution (Stamatatos, 2013; Sari et al., 2017), and (iii) probability calibration and label-shift correction for distribution changes (Zadrozny and Elkan, 2002; Niculescu-Mizil and Caruana, 2005; Saerens et al., 2002). Prior work on AI-generated code detection spans stylometry-based signals, perplexity/perturbation-style detectors (Mitchell et al., 2023; Xu and Sheng, 2024), and learned token/AST representations in benchmark settings for machine-generated code detection (Orel et al., 2026b). Our contribution is complementary: we explicitly reduce language leakage during feature selection, which is especially relevant under cross-language OOD evaluation such as SemEval-2026 Task 13.

3 Task and metric

Given a code snippet x in language ℓ, we predict $y \in \{0, 1\}$, where $y = 1$ denotes AI-generated

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 59–64
July 3-4, 2026 ©2026 Association for Computational Linguistics

code. The key challenge is generalization to unseen languages.

We use **macro-F1** as the primary metric, defined as the average of class-wise F1:

$$\mathrm{F1}_{\mathrm{macro}} = \frac{\mathrm{F1}_{y=0} + \mathrm{F1}_{y=1}}{2}. \tag{1}$$

We also report per-language precision and recall.

4 System overview

Figure 1 summarizes the pipeline. The system is a deterministic feature extractor followed by a discriminative classifier.

4.1 Pipeline

Our training script implements a deterministic pipeline: data loading, feature extraction, feature interactions, global normalization, leakage-aware feature selection, Optuna optimization, and final training with lightweight post-processing.

5 Language-agnostic features

We extract 286 candidate features and select 214 in the best configuration. Below we summarize the families and motivation.

5.1 Skeletal features (structure and complexity)

Skeletal features approximate program structure without relying on language-specific tokens. They include indentation statistics, approximate control-flow counts, and lightweight complexity proxies.

Indentation nesting and instability. We estimate per-line indentation (tabs mapped to 4 spaces) and compute summary statistics over indentation levels $\{d_i\}$, including the mean, maximum, standard deviation, and the average change magnitude $|d_i - d_{i-1}|$.

Control flow. We approximate counts of control-flow constructs (e.g., `if/for/while/try/switch`) via regexes and normalize by non-empty lines:

$$\text{control\_density} = \frac{N_{\text{control}}}{N_{\text{lines}}}. \tag{2}$$

Approximate cyclomatic complexity. We approximate McCabe complexity (McCabe, 1976) as:

$$\mathrm{CC}_{\text{approx}} = 1 + N_{\text{decision}}, \tag{3}$$

where decision points come from the control-flow counts.

Halstead-style metrics. Following Halstead (Halstead, 1977), we count operators and operands to build vocabulary $n = n_1 + n_2$, length $N = N_1 + N_2$, and volume:

$$V = N \log_2(n). \tag{4}$$

These proxies capture coarse notions of code "effort"/"difficulty".

Approximate cognitive complexity. Inspired by cognitive complexity guidelines (Campbell, 2017), we penalize control-flow nesting and logical connectors (e.g., `and/or`, `&&/||`).

5.2 Character and whitespace features

Character-level features capture spacing/indentation regularities that are often more uniform in model-generated code (Stamatatos, 2013; Sari et al., 2017). We use informative character n-grams, statistics of space runs (mean/std/max), an indentation-multiples proxy (divisible by 2/4), and character entropy.

5.3 Micro-style patterns

Micro-style features capture fine-grained habits (operator spacing, blank-line patterns, indentation consistency). In our experiments, blank-line features (`micro_blank_*`) are among the most transferable signals. [1]

6 Feature engineering and selection

6.1 Interactions

We add a small set of interaction features (products and stabilized ratios), which rank among the most important signals.

6.2 Global normalization (domain-invariant)

We apply global z-normalization to a subset of universal features (Eq. 5) to reduce sensitivity to scale changes across domains:

$$\tilde{f} = \frac{f - \mu_f}{\sigma_f + \epsilon}. \tag{5}$$

6.3 Leakage-aware selection

We follow supervised feature selection principles (Guyon and Elisseeff, 2003) while explicitly penalizing language predictivity. We compute:

- **AI vs. human discriminability**: mean/min per-language AUC (using $\max(\mathrm{AUC}, 1 - \mathrm{AUC})$).

[1] All experiments use a fixed random seed (42) and 5-fold stratified CV.

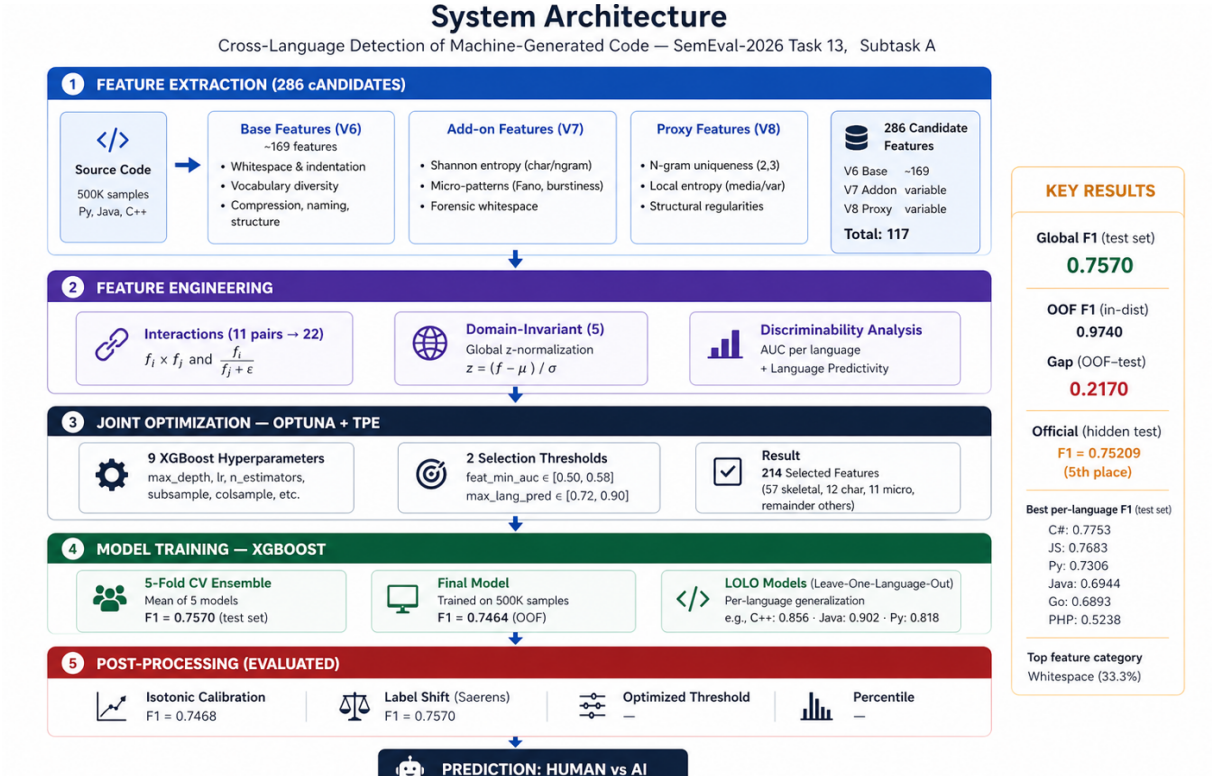

Figure 1: Overview of the proposed pipeline (feature extraction/engineering, selection, and training).

- **Language predictivity**: mean one-vs-rest AUC for predicting the programming language.

We select a (non-risky) feature if it satisfies:

$$\begin{aligned} \mathrm{AUC}_{\mathrm{mean}} &\geq \tau_{\mathrm{auc}}, \\ \mathrm{AUC}_{\mathrm{min}} &\geq \tau_{\mathrm{auc}} - 0.05, \\ \mathrm{LangPred} &\leq \tau_{\mathrm{lang}}. \end{aligned} \tag{6}$$

We also always include a small forced set (universal + skeletal) and exclude explicitly risky syntax-dependent features (e.g., counting `main` or semicolons per line).

7 Model, optimization, and post-processing

7.1 Classifier

We train an XGBoost classifier (Chen and Guestrin, 2016), using inverse-frequency language weights with clipping for stability.

7.2 Optuna optimization

We jointly optimize XGBoost hyperparameters and selection thresholds $(\tau_{\mathrm{auc}}, \tau_{\mathrm{lang}})$ with Optuna (Akiba et al., 2019) (700 trials). Each trial trains a 5-fold stratified CV ensemble; optimization is performed using OOF-CV. The test score is reported for completeness and was not used for selecting hyperparameters. We also track the generalization gap:

$$\mathrm{gap} = \mathrm{F1}_{\mathrm{OOF}} - \mathrm{F1}_{\mathrm{test}}. \tag{7}$$

7.3 Post-processing: calibration and label shift

Given a prior shift (train 52.30% AI vs. test 22.30% AI), we evaluate:

- **Calibration** via isotonic regression (Zadrozny and Elkan, 2002; Niculescu-Mizil and Caruana, 2005) fitted on OOF predictions.

- **Label-shift correction** following Saerens et al. (Saerens et al., 2002).

In our experiments, these post-processing steps did not improve macro-F1, so we report the uncalibrated model.

8 Experimental setup

Training: 500,000 samples in {C++, Java, Python}. Test: 1,000 samples in 8 languages {C, C#, C++, Go, Java, JavaScript, PHP, Python}. Unseen languages: {C, C#, Go, JavaScript, PHP}.

Split	#Samples	AI ratio
Train	500 000	52.3%
Test	1 000	22.3%

Table 1: Dataset summary.

8.1 Official 100K validation set and model selection

We primarily relied on 5-fold OOF-CV for model selection to reduce sensitivity to a single split and to estimate stability under re-sampling (Stone, 1974; Bousquet and Elisseeff, 2002). We did not use the official 100K validation set for hyperparameter tuning; instead, all selection decisions were made from OOF-CV only. This avoids tuning on a single external split and prevents test-set optimization.

9 Results

9.1 Overall results and generalization gap

We report macro-F1=0.7570 on the organizer-provided 1,000-sample test set for reproducible experiments, while the official hidden evaluation yields macro-F1=0.75209 (5th place in Subtask A). Our best trial (243) reaches OOF macro-F1=0.9740, implying a gap of 0.2170 relative to the 1,000-sample test set and suggesting additional shift beyond language.

Post-processing note. Isotonic calibration reduced macro-F1 to 0.7468, while label-shift correction matched the uncalibrated score (0.7570); therefore we report the uncalibrated model.

9.2 Domain shift as a contributor to the OOF–test gap (discussion)

Beyond language transfer, domain differences (algorithmic vs. research/production) can alter formatting regularities and structural/style cues (including whitespace consistency), affecting feature-engineered detectors even when the language is unchanged. Thus, the OOF–test gap (0.2170) likely reflects combined language, domain, and generator/prompt/style shifts. Future work should better disentangle these factors and explore format-robust signals (e.g., AST-based representations).

9.3 Multiple runs and variants

Beyond the official submission, we tested internal baselines and robustness variants (all negative); Table 2 summarizes representative runs.

Setting	Test macro-F1	Notes
Baseline (V2, 17 feats)	0.7180	Whitespace + structure
Intermediate (V5)	0.7623	Skeletal features
Final system (V6, 214 feats)	0.7570	Joint selection/HP
V6 + Augmentation	0.7500	ΔF1=-0.007
V6 + Projections	0.7410	ΔF1=-0.016
V6 + DRO-style	0.7310	ΔF1=-0.026

Table 2: Representative runs/variants (Augmentation=Mixup-style feature mixing (Zhang et al., 2018); Projections=linear language-signal suppression (Elazar and Goldberg, 2018); DRO-style=group-robust reweighting (Sagawa et al., 2020)).

Setting	OOF macro-F1	Test macro-F1	Gap	#Features
Proposed model (best, Trial 243)	0.9740	0.7570	0.2170	214

Table 3: Overall result of the best model.

9.4 Baselines

Simpler whitespace/structure subsets are competitive but consistently below the final model, suggesting that micro-style signals and interactions add robustness under cross-language shift. Although V5 peaks slightly higher on aggregate macro-F1 internally, V6 was selected as the official run due to leakage-aware selection and reduced reliance on language-predictive cues (Eq. 6).

9.5 Negative results: why DRO did not help

Group-DRO over seen-language groups did not improve OOD performance. Reweighting increases variance and worst-case optimization over seen languages does not necessarily transfer to unseen languages or domains, which likely dominate our test-time shifts.

9.6 Per-language transfer

Table 4 reports test performance by language: transfer is strong to C# and JavaScript, while Go shows high precision but low recall, and PHP is close to chance. Averaging over languages, seen languages reach macro-F1=0.7458 while unseen languages reach macro-F1=0.6797, highlighting the expected OOD drop under cross-language transfer.

9.7 Error analysis

Manual inspection suggests frequent Go false negatives under enforced formatting (e.g., `gofmt`) and mixed PHP errors due to hybrid syntax and embedded blocks (e.g., HTML) that distort indentation-based cues.

Language	Seen?	F1	Prec.	Rec.	N
C	Unseen	0.6912	0.6781	0.7146	51
C#	Unseen	0.7753	0.7598	0.8005	122
C++	Seen	0.7208	0.7208	0.7208	75
Go	Unseen	0.6400	0.8929	0.6250	60
Java	Seen	0.7161	0.7071	0.7270	256
JavaScript	Unseen	0.7683	0.7649	0.7774	85
PHP	Unseen	0.5238	0.5299	0.5778	48
Python	Seen	0.8004	0.7904	0.8140	303

Table 4: Per-language results on test.

Feature	Gain
tab_space_mix_ratio_div_style_spacing_consistency	0.1301
tab_space_mix_ratio_div_avg_line_length	0.1061
skel_control_density	0.0536
tab_space_mix_ratio	0.0532
tab_space_mix_ratio_gnorm	0.0498
tab_char_ratio	0.0428
tab_char_ratio_gnorm	0.0402
bracket_balance	0.0266
tab_space_mix_ratio_x_style_spacing_consistency	0.0226
tab_space_mix_ratio_x_avg_line_length	0.0220

Table 5: Top-10 features by XGBoost gain (Trial 243).

9.8 Feature analysis

Feature importances (XGBoost gain) are dominated by whitespace cues and their interactions; Table 5 shows the top-10 features (8/10 whitespace-related).

9.9 Ablation

Component-level ablations indicate that (i) a whitespace/structure baseline is already strong, (ii) skeletal features provide additional gains, (iii) micro-style features (notably blank-line patterns) transfer well, and (iv) the tested adaptation variants did not improve performance (details in the repository).

10 Discussion

We evaluate four hypotheses based on our experiments.

H1: Whitespace provides a universal discriminative signal. Supported. Top-ranked features are dominated by whitespace cues, especially `tab_space_mix_ratio` and its interactions.

H2: Structural metrics (McCabe/Halstead) help cross-language detection. Partially supported. Skeletal features carry signal but are also more predictive of language, requiring leakage-aware selection.

H3: Character n-grams are fully language-agnostic in code. Not supported. Several n-grams have high AUC but also predict the programming language, explaining why only a small subset is selected.

H4: Blank-line micro-patterns generalize best. Supported. `micro_blank_*` features show lower language predictivity and rank among the most useful.

11 Limitations and future work

- **Multiple simultaneous shifts**: the gap likely reflects combined language, domain, and generator/prompt/style shifts. Disentangling these effects and designing signals robust to domain conventions remains important future work (Bousquet and Elisseeff, 2002; Orel et al., 2026b).
- **Outlier languages**: PHP and Go likely require additional semantic signals (e.g., parsing/AST).
- **Reducing the gap**: training on more languages and using stricter validation (LOLO/LOOCV) may improve robustness (Stone, 1974).

12 Conclusion

LAFED reaches macro-F1=0.7570 on the organizer-provided 1,000-sample test set and macro-F1=0.75209 (5th place) on the official hidden evaluation. Ablations show consistent gains from the full feature set over simpler baselines, while robustness/adaptation variants did not improve macro-F1. Feature importance is dominated by whitespace interactions (Table 5); skeletal signals are complementary but require leakage-aware selection (Eq. 6). Remaining failures on Go/PHP and the OOF–test gap (0.2170) motivate more format-robust semantic signals (e.g., AST) in future work.

References

Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Tetsuya 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.

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)*.

M. Stone. 1974. Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147.

Olivier Bousquet and André Elisseeff. 2002. Stability and Generalization. *Journal of Machine Learning Research*, 2:499–526.

Thomas J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320.

Maurice H. Halstead. 1977. *Elements of Software Science*. Elsevier North-Holland.

G. Ann Campbell. 2017. Cognitive Complexity: A new way of measuring understandability. SonarSource Technical Report.

Efstathios Stamatatos. 2013. On the Robustness of Authorship Attribution Based on Character N-gram Features. *Journal of Law and Policy*, 21.

Yunita Sari, Mark Stevenson, and Andreas Vlachos. 2017. Continuous N-gram Representations for Authorship Attribution. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 267–273.

Bianca Zadrozny and Charles Elkan. 2002. Transforming Classifier Scores into Accurate Multiclass Probability Estimates. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 694–699.

Alexandru Niculescu-Mizil and Rich Caruana. 2005. Predicting Good Probabilities with Supervised Learning. In *Proceedings of the 22nd International Conference on Machine Learning*.

Marco Saerens, Patrice Latinne, and Christine Decaestecker. 2002. Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure. *Neural Computation*, 14(1):21–41.

Isabelle Guyon and André Elisseeff. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, 3:1157–1182.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*.

Zhenyu Xu and Victor S. Sheng. 2024. Detecting AI-Generated Code Assignments Using Perplexity of Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38.

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 (Volume 1: Long Papers)*.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. 2018. mixup: Beyond Empirical Risk Minimization. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*.

Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. 2020. Distributionally Robust Neural Networks for Group Shifts: On the Importance of Regularization for Worst-Case Generalization. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Yanai Elazar and Yoav Goldberg. 2018. Adversarial Removal of Demographic Attributes from Text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 11–21.