Team Yuvan at SemEval-2026 Task 13: Task-Adaptive Ensemble Strategies for AI-Generated Code Detection

Yuvan Ramesh[1] **Tongtong Wu**[2]
[1]Dougherty Valley High School
[2]Department of Data Science & AI, Monash University
yuvanramesh11@gmail.com tongtong.wu@monash.edu

Abstract

We describe our system for SemEval-2026 Task 13 on detecting machine-generated code across eight programming languages and three subtasks: binary human-vs-AI detection, 11-way source identification, and 4-way generator classification. Our approach uses a task-specific combination of Qwen2.5-Coder-1.5B with LoRA fine-tuning, abstract syntax tree features, CodeBERT with head-tail chunking, and TF-IDF baselines. Experiments reveal three main findings. For Task A, neural detectors degrade markedly on the official test split, while structural features remain more stable, suggesting substantial distribution shift. For Task B, inverse-frequency class weighting is essential under extreme label imbalance and greatly improves macro-F1. For Task C, combining neural and statistical models performs better than relying on a single model alone, indicating complementary strengths across representations. Our final system achieves 0.449 macro-F1 on Task B and 0.714 macro-F1 on Task C, and offers practical insights into robustness, imbalance handling, and model complementarity for AI-generated code detection.

1 Introduction

Large language models can now generate source code that is often difficult to distinguish from human-written code, raising important concerns for software engineering, education, and open-source ecosystems (Idialu et al., 2024; Nguyen et al., 2024; Shi et al., 2025; Orel et al., 2025a, 2026a). As a result, detecting code provenance, i.e., whether a snippet was written by a human or generated by a model, and in some cases by which generator, has become an increasingly important benchmark problem (Idialu et al., 2024; Nguyen et al., 2024; Orel et al., 2025a,b, 2026a).

SemEval-2026 Task 13 formulates this problem through three subtasks across eight programming languages (Python, Java, C++, JavaScript, Go, C, PHP, and C#) (Orel et al., 2026b): **Task A** (Binary): classify code as human-written or AI-generated. **Task B** (11-way): identify the source among one human class and ten AI generators under extreme class imbalance. **Task C** (4-way): classify code among four AI generators, without a human class. All subtasks are evaluated using macro-averaged F1, which emphasizes balanced performance across classes rather than performance on majority categories (Orel et al., 2026b).

This shared task is challenging for at least three reasons. First, models must generalize under substantial train–test mismatch, particularly in Task A, where strong validation performance does not reliably transfer to the official test split. Recent benchmark studies have shown that AI-generated code detection can be highly sensitive to domain, generator, and application scenario shifts, making robust generalization a central challenge (Orel et al., 2025a,b, 2026a). Second, Task B involves a highly imbalanced 11-way classification setting, making naive training objectives poorly aligned with the macro-F1 metric. Third, all three subtasks span multiple programming languages, requiring models to remain robust across different syntax, style, and structural conventions, a difficulty also highlighted in prior multilingual and multi-generator detection settings.

In this paper, we present a practical multi-strategy system for SemEval-2026 Task 13, where different subtasks benefit from different modeling choices. Our approach combines statistical and neural representations, including pre-trained code encoders such as CodeBERT (Feng et al., 2020), parameter-efficient adaptation with LoRA (Hu et al., 2022), and complementary non-neural features, motivated by prior evidence that machine- and human-written code exhibit distinguishable but heterogeneous signals (Nguyen et al., 2024; Shi et al., 2025). Our study yields three main empirical findings:

Proceedings of the The 20th International Workshop on Semantic Evaluation (2026), pages 2089–2097
July 3-4, 2026 ©2026 Association for Computational Linguistics

1. For **Task A**, we observe that several neural detectors are highly unstable under the official split, while structural representations provide more robust performance. This pattern is consistent with a substantial distribution shift between training/validation and test data (§3.2).
2. For **Task B**, we find that **inverse-frequency class weighting** is critical in the extreme-imbalance setting, substantially improving macro-F1 and making the model much better aligned with the evaluation objective (§3.3).
3. For **Task C**, we show that **combining complementary models** is more effective than relying on the single strongest model alone: statistical and neural representations capture different signals, and their combination leads to the best final performance (§3.4).

2 Related Work

Generated-text and code provenance detection. Generated-content detection began in natural language with zero-shot probability methods (e.g., DetectGPT (Mitchell et al., 2023)), watermarking, and supervised classification. In source code, provenance has deeper roots in *code stylometry*: Caliskan-Islam et al. (2015) showed that abstract syntax tree features can identify programmers with high accuracy, establishing that structural signals carry strong authorship information. Because code exhibits stronger syntactic regularity than natural language, code-focused methods have adopted code-aware modeling, including encoder classifiers like GPTSniffer (Nguyen et al., 2024), stylometric analyses (Idialu et al., 2024), and perturbation-oriented approaches such as DetectCodeGPT-style methods (Shi et al., 2025). Our Task A findings are consistent with this view: AST-based features transfer to the shifted test set more robustly than fine-tuned neural detectors.

Benchmarks and robustness under distribution shift. Recent benchmarks move beyond small-scale in-distribution classification to multilingual, multi-generator, and OOD evaluation. CoDet-M4 (Orel et al., 2025a) studies detection across languages, domains, and generators, Droid (Orel et al., 2025b) adds adversarial and co-authorship conditions, and AICD Bench (Orel et al., 2026a) scales both data diversity and evaluation scope. SemEval-2026 Task 13 (Orel et al., 2026b) builds on this line under severe imbalance and fine-grained attribution.

Task	Train	Val	Test	Classes	Imbalance
A	500K	100K	500K	2	1:1
B	500K	100K	500K	11	224:1
C	900K	200K	500K	4	6:1

Table 1: Dataset statistics across the three subtasks.

Code representations, efficient adaptation, and class imbalance. Methodologically, our work relates to pretrained code representations, parameter-efficient adaptation, and imbalance-aware training. CodeBERT (Feng et al., 2020) is a strong backbone for code understanding; GraphCodeBERT (Guo et al., 2021) adds data-flow information (we use it as a logistic-regression backbone on Task A); for long code we follow the head–tail truncation of Sun et al. (2019); LoRA (Hu et al., 2022) provides efficient fine-tuning; and Qwen2.5-Coder (Hui et al., 2024) supplies a strong code-specialized base model. For Task B's $\sim$224:1 imbalance, Cui et al. (2019) propose effective-number re-weighting, generalizing inverse-frequency weighting; in our setting, inverse-frequency weights already break majority-class collapse and substantially improve macro-F1. In contrast to prior code-provenance studies centered on binary detection (Nguyen et al., 2024; Idialu et al., 2024; Shi et al., 2025), our setting jointly requires multilingual coverage, distribution shift, extreme imbalance, and multi-class attribution, motivating a combination of neural, statistical, and structural signals.

3 Task Setup and Our System

Each subtask provides train/val/test splits as Parquet source-code snippets across 8 programming languages (Table 1); Task B is the most imbalanced, with a 224:1 ratio between the majority Human class and the smallest generator class, making macro-F1 optimization particularly challenging. Our system consists of four complementary components (§3.1) that we combine differently for Tasks A–C (§3.2–§3.4).

3.1 Shared Model Components

Qwen-2.5-Coder with LoRA. Our primary neural model is `Qwen2.5-Coder-1.5B-Instruct` (Hui et al., 2024), chosen for its code specialization and long-context support. We fine-tune it with LoRA (Hu et al., 2022) (r=16, α=32, dropout

Model	Val F1	Test F1	Flipped
CodeBERT (125M)	0.995	0.245	0.561
Qwen (1.5B)	0.998	0.269	0.587
GraphCodeBERT+LR	0.900	0.328	—

Table 2: Task A prediction inversion phenomenon. Flipping neural predictions substantially improves test performance, indicating a strong distribution shift between validation and test data.

0.05) on attention and MLP projections, yielding 18.5M trainable parameters; full hyperparameters are in Appendix E.

CodeBERT with Head–Tail Chunking. As a complementary encoder-based baseline, we use `codebert-base` (Feng et al., 2020). Because CodeBERT is limited to 512 tokens, we apply head–tail truncation (Sun et al., 2019) for long inputs:

$$\mathbf{x}_{\text{chunk}} = \mathbf{x}_{[:256]} \oplus \mathbf{x}_{[-256:]}, \tag{1}$$

fully fine-tuned for 5 epochs at 2×10^{-5}.

TF-IDF with Logistic Regression. A lightweight statistical model based on character n-grams (orders 1–3, 20K features) with balanced Logistic Regression, which captures surface-form regularities (lexical preferences, naming, formatting) that complement neural encoders in ensembles.

3.2 Task A: Binary Classification

Task A requires distinguishing human-written code from AI-generated code. Although the training data are balanced, this subtask shows the strongest mismatch between validation and test behavior: CodeBERT- and Qwen-based classifiers achieved near-perfect validation F1 but transferred poorly to the test set, and post-hoc analysis revealed a consistent label-inversion pattern across architectures (Table 2), pointing to severe train–test mismatch rather than a model-specific failure.

AST-Based Structural Features. To reduce reliance on fragile lexical cues, we build a LightGBM classifier over 41 structural features (node statistics, depth patterns, complexity, semantic counts, identifier properties, Halstead-style metrics) extracted with tree-sitter across 8 languages. Feature-importance analysis (Appendix C, Table 5) shows that formatting-related cues dominate, and the AST model has a much smaller validation–test gap than neural variants. Our final Task A prediction blends it with a small flipped-Qwen component, $P_{\text{final}} = 0.95 \cdot P_{\text{AST}} + 0.05 \cdot (1 - P_{\text{Qwen}})$, though our main conclusion does not depend on the second term.

3.3 Task B: 11-Way Generator Identification

Task B requires identifying the source among 11 classes: one Human class and ten AI generators. This is the most imbalanced subtask, with Human accounting for 88.3% of the training set and the rarest generator only 0.39%. Under macro-F1, such imbalance makes naive majority prediction ineffective and requires explicit treatment of minority classes.

Direct Classification with Class-Weighted Loss. We initially tried a two-stage pipeline (Human-vs-AI, then generator identity), but early-stage errors propagated and hurt macro-F1, so we adopted a single end-to-end 11-way Qwen classifier with class-weighted cross-entropy:

$$w_c = \frac{N}{K \cdot n_c}, \tag{2}$$

where N is the total training size, K=11, and n_c is the class count. This downweights the dominant Human class and upweights rare generators, encouraging more even capacity allocation. We train 5 epochs on a 300K stratified subset; the resulting class weights range from 0.10 (Human) to 23.10 (Gen-5), with the full distribution and training-progression checkpoints in Appendix C (Table 7).

Per-Class Analysis. Despite the heavy skew, the class-weighted model achieves non-trivial F1 on all classes rather than collapsing to the Human label, with several minority generators reaching competitive F1 (Appendix C); the harder-to-recover classes point to substantial overlap between certain AI-generated and human-written styles.

3.4 Task C: 4-Way Generator Classification

Task C classifies AI-generated code among 4 generators without a Human class. Compared with Task B, the label distribution is much less extreme, which makes this subtask more suitable for model combination. This is also our strongest setting overall.

Weighted Probability Averaging. We train three complementary models — Qwen-2.5-Coder (strongest single model), CodeBERT with head–tail chunking (alternative encoder representation), and TF-IDF with Logistic Regression (lexical regularities not captured by neural encoders) — and combine them by weighted probability averaging,

$$P_{\text{ens}}(y{=}c \mid x) = \sum_{m \in \mathcal{M}} w_m \cdot P_m(y{=}c \mid x), \quad (3)$$

with $\mathcal{M} = \{\text{Qwen, CodeBERT, TF-IDF}\}$ and $\sum_m w_m = 1$. Probability averaging (vs. hard voting) preserves confidence information and lets weaker but complementary models contribute when confident.

Weight Search and Per-Class Behavior. We tune ensemble weights by grid search over the simplex with step size 0.05. The best settings consistently assign the largest weight to Qwen with smaller but useful contributions from TF-IDF and/or CodeBERT, and the narrow gap among top-ranked configurations (Appendix C) indicates that the ensemble is robust to moderate weight changes. Notably, TF-IDF remains useful despite being weaker than Qwen as a standalone model, showing that complementarity matters beyond individual validation strength. Per-class validation F1 (Appendix C) confirms that the ensemble slightly improves over the strongest single model while yielding the most balanced performance across classes, particularly narrowing the gap on the smallest class.

Inference at Scale. For the large Task C test set, Qwen inference is distributed across 8 GPUs, while CodeBERT and TF-IDF use standard batched inference, making the final ensemble practical at evaluation scale.

4 Experimental Setup

We use the official train/validation/test splits (Table 1); models are trained on the training set, selected by validation macro-F1, and evaluated through the shared-task submission platform. Total training cost is approximately 119 A100 GPU-hours, dominated by Task B ($\sim$60 GPU-hours for 5 epochs on 300K samples). Full software versions, deterministic settings, and a per-component compute breakdown are in Appendix F and Appendix D.

5 Results and Analysis

5.1 Main Results

The validation macro-F1 in Table 3 is reported for the final selected checkpoint, while the per-class breakdown in Table 12 is shown for an intermediate analysis checkpoint; the two numbers are therefore not directly comparable. Our final system achieves competitive performance on Task B (0.449 macro-F1) and Task C (0.714 macro-F1). Across all three tasks, test performance is consistently lower than validation performance, indicating substantial distribution mismatch between development and evaluation conditions. This effect is especially pronounced for Task B and Task C, where the validation–test gap remains large despite strong validation results.

Task	Val F1	Test F1	OOD Drop
A	0.665	0.638	-4%
B	0.698	0.449	-36%
C	0.911	0.714	-22%

Table 3: Official results and validation-to-test performance drop.

The three tasks also differ in what kinds of signals transfer reliably. Task C is the most favorable setting, where complementary signals from neural and statistical models can be combined effectively. Task B is substantially harder due to its 11-way label space and severe class imbalance, making minority-class recovery central to macro-F1. In Task A, we observe that high validation performance from neural models does not translate to strong test results, whereas structural features remain comparatively stable: the neural Qwen classifier drops by $\sim$73% from validation to test while the AST model drops by only $\sim$4% (full per-model validation–test comparison in Appendix C, Table 10). Overall, these findings suggest that robust AI-generated code detection depends not only on model strength, but also on representation choice and robustness under shift.

5.2 Stability and Baselines

Table 4 compares our final system against several implemented baselines: our method performs best on all three tasks, with the largest absolute gain on Task B, supporting the benefit of task-specific design choices rather than a single uniform baseline. The Task C ensemble is also stable across random seeds: training with seeds 42, 123, and 2024 yields 0.909 ± 0.0004 validation F1 and 0.712 ± 0.002 test F1, so the final result is not driven by a favorable seed.

Method	A	B	C	Avg
Random	0.50	0.09	0.25	0.28
Majority class	0.33	0.08	0.14	0.18
TF-IDF + LR	0.41	0.21	0.58	0.40
CodeBERT	0.56	0.35	0.65	0.52
Our system	**0.64**	**0.45**	**0.71**	**0.60**

Table 4: Comparison with implemented baselines on the test set (macro-F1).

6 Discussion

Task-adaptive detection under shift. Task A is better understood as a robustness challenge than a standard in-distribution problem: the consistent label-flip effect across neural models indicates that high validation performance may not transfer when test code is heavily post-processed, while the stronger transfer of AST-based features suggests that structural signals are more stable under such shifts. Task B further shows that explicit class reweighting is essential under extreme imbalance, and Task C shows that ensemble gains come primarily from representation diversity rather than from adding stronger neural models. Together these results support a task-adaptive view of code provenance detection that combines robustness-aware modeling, imbalance-aware optimization, and heterogeneous ensembling; remaining limitations include the lack of newer encoder baselines, no ensemble for Task B, and limited analysis of the label-flip effect across languages and code lengths.

Is the label-flip a sign of overfitting? A reviewer raised whether the label-flip effect on Task A could simply reflect overfitting. We view overfitting as a plausible partial explanation: a classifier that fits training-time surface cues (e.g., formatting or lexical regularities) can mechanically produce inverted predictions when those cues are altered or flipped in the evaluation split. However, the effect is consistent across very different model families and capacities, including fully fine-tuned CodeBERT (125M parameters), GraphCodeBERT with a logistic-regression head, and Qwen-2.5-Coder with only 1.2% trainable LoRA parameters. Classical parameter-capacity overfitting is therefore unlikely to be the sole cause, and a systematic distribution shift between training/validation and test data is the more likely dominant factor, with overfitting to non-robust features acting as a contributing mechanism.

7 Conclusion

We presented Team Yuvan's system for SemEval-2026 Task 13, combining structural features, reweighted classification, and heterogeneous ensembling; our results show that no single method is best across subtasks and that robust detection depends on matching model design to each task's data characteristics.

8 Limitations

Our findings are benchmark-specific and may not fully generalize to broader real-world code provenance settings with unseen generators or stronger distribution shifts. In addition, our analysis is primarily empirical and does not yet explain the mechanisms behind the observed robustness and imbalance effects. Finally, we explore only a limited set of models and features, leaving broader architectural and representation choices to future work.

Acknowledgments

We thank the SemEval-2026 Task 13 organizers for this challenging benchmark.

References

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.

Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge Belongie. 2019. Class-balanced loss based on effective number of samples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9268–9277.

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, Online. Association for Computational Linguistics.

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

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, and 5 others. 2024. Qwen2.5-Coder technical report. *Preprint*, arXiv:2409.12186.

Oseremen Joy Idialu, Noble Saji Mathews, Rungroj Maipradit, Joanne M. Atlee, and Mei Nagappan. 2024. Whodunit: Classifying code as human authored or GPT-4 generated – a case study on CodeChef problems. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR)*, pages 394–406. ACM.

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)*.

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

Daniil Orel, Dilshod Azizov, and Preslav Nakov. 2025a. CoDet-M4: Detecting machine-generated code in multi-lingual, multi-generator and multi-domain settings. In *Findings of the Association for Computational Linguistics: ACL 2025*. Association for Computational Linguistics.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026a. AICD bench: A challenging benchmark for AI-generated code detection. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2026)*. Association for Computational Linguistics.

Daniil Orel, Dilshod Azizov, Indraneil Paul, Yuxia Wang, Iryna Gurevych, and Preslav Nakov. 2026b. SemEval-2026 task 13: Detecting machine-generated code with multiple programming languages, generators, and application scenarios. In *Proceedings of the 20th International Workshop on Semantic Evaluation (SemEval-2026)*, San Diego, USA. Association for Computational Linguistics.

Daniil Orel, Indraneil Paul, Iryna Gurevych, and Preslav Nakov. 2025b. Droid: A resource suite for AI-generated code detection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Yuling Shi, Hongyu Zhang, Chengcheng Wan, and Xiaodong Gu. 2025. Between lines of code: Unraveling the distinct patterns of machine and human programmers. In *Proceedings of the 47th International Conference on Software Engineering (ICSE)*.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? In *Proceedings of the 18th China National Conference on Computational Linguistics (CCL)*. Springer.

A Ablation Studies

This appendix presents the full ablation studies for each task that were summarized in the main body.

Task A: All Approaches Tested. The full six-variant Task A comparison (Table 6) reinforces the pattern visible in the main results: neural and stylometric variants with near-perfect validation F1 transfer poorly, while the AST model yields the strongest test F1 despite much lower validation F1, indicating that Task A is particularly sensitive to distribution mismatch. Table 5 lists the most informative AST features, dominated by formatting cues.

Feature	Importance (%)
space_indent_count	14.3
tab_space_ratio	13.4
tab_count	12.4
indent_std	7.9
indent_max	5.1
blank_line_count	3.9

Table 5: Top structural features for Task A. Formatting-related cues account for a large portion of the model's discriminative signal.

Approach	Val F1	Test F1
CodeBERT (normal)	0.995	0.245
CodeBERT (flipped)	0.005	0.561
Qwen (normal)	0.998	0.269
Qwen (flipped)	0.002	0.587
GraphCodeBERT + LR	0.900	0.328
AST (LightGBM)	0.665	**0.638**

Table 6: Task A ablation. The strongest validation results do not correspond to the strongest test performance; the AST-based model generalizes best on the official test set.

Task B: Design Decision Impact. Full data-volume and pipeline ablations (Table 8) show that balanced direct training is decisive: both the two-stage pipeline and the unweighted direct classifier collapse on test (from 0.35–0.41 Val F1 to 0.085 Test F1), and gains from scale plateau near 300K (Test F1 0.449) with only marginal improvement at 500K (0.450). Table 7 shows the underlying class distribution and inverse-frequency weights used during training.

Class	Label	Count	%	w_c
0	Human	441,729	88.4	0.10
1	Gen-1	4,161	0.8	10.93
2	Gen-2	8,993	1.8	5.06
3	Gen-3	3,028	0.6	15.01
4	Gen-4	2,225	0.4	20.43
5	Gen-5	1,968	0.4	23.10
6	Gen-6	5,782	1.2	7.86
7	Gen-7	8,195	1.6	5.55
8	Gen-8	8,125	1.6	5.59
9	Gen-9	4,609	0.9	9.87
10	Gen-10	10,811	2.2	4.21

Table 7: Task B class distribution and inverse-frequency weights.

Configuration	Val F1	Test F1
Two-stage (CodeBERT)	0.41	0.085
Direct, no weights	0.35	0.085
Direct, balanced, 50K	0.60	0.394
Direct, balanced, 100K	0.68	0.350
Direct, balanced, 200K	0.68	0.429
Direct, balanced, 300K	**0.70**	**0.449**
Direct, balanced, 500K	0.71	0.450

Table 8: Task B ablation. Balanced direct classification is substantially more effective than either the two-stage pipeline or unweighted training. Increasing data volume generally helps, with diminishing returns beyond 300K.

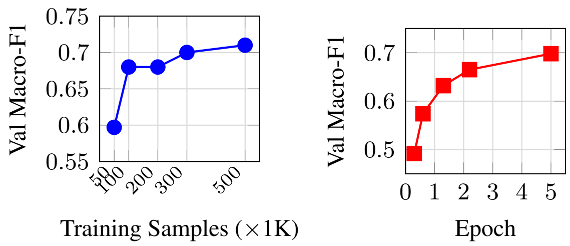
Task C: Incremental Ensemble. Incremental addition of CodeBERT and TF-IDF to Qwen each yields small but consistent test-F1 gains, with the full three-model ensemble best overall at 0.714 (Table 9), supporting the view that complementary representations beat a single strong backbone.

Configuration	Val F1	Test F1
Qwen only	0.909	0.710
Qwen + CodeBERT	0.910	0.711
Qwen + TF-IDF	0.910	0.713
All three models	**0.911**	**0.714**

Table 9: Task C ensemble ablation. Combining neural and statistical models yields small but consistent gains over the single-model baseline.

B Sensitivity Analysis

We examine four sensitivity dimensions for our final pipeline. For Task B, validation F1 generally improves with training data volume, with diminishing returns beyond 300K and a small non-monotonicity at 100K (Figure 1a); most training-epoch gains occur within the first two epochs and continue more gradually through epoch 5 (Figure 1b); and learning rate is the most fragile hyperparameter, with validation macro-F1 peaking at 2×10^{-4} and dropping sharply at 5×10^{-4} and 1×10^{-3} (Figure 2). For Task C, a 231-point grid search shows that performance is relatively stable across a broad range of Qwen-centered weight mixtures rather than requiring a single sharply tuned setting (Figure 3).



(a) Val F1 vs. data volume (b) Val F1 vs. training epoch

Figure 1: Task B sensitivity analysis. (a) Validation F1 generally improves with more training data, though the trend is not strictly monotonic at smaller scales. (b) Validation F1 improves rapidly in early epochs and continues to increase more gradually through epoch 5.

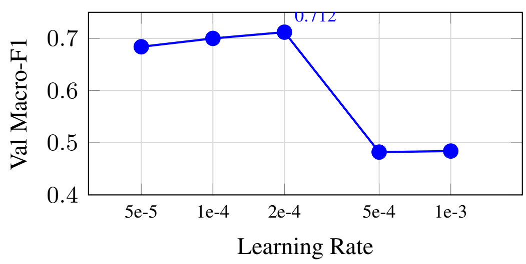


Figure 2: Task B learning-rate sensitivity (300K samples, 5 epochs). Performance peaks at 2×10^{-4} and drops sharply at larger learning rates.

C Additional Tables

This appendix collects the per-model validation-to-test comparison, the Task B training progression, and per-class breakdowns for Task B and Task C.

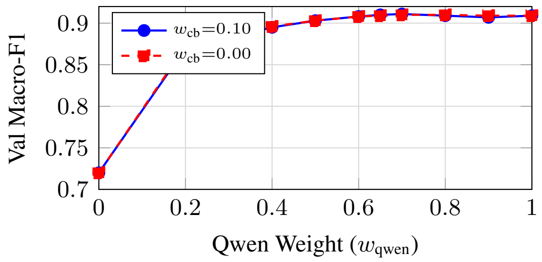


Figure 3: Task C ensemble-weight sensitivity. Validation performance remains strong across a broad range of Qwen-centered mixtures, indicating a relatively flat optimum.

Model / Task	Val F1	Test F1	Drop
A – Neural (Qwen)	0.998	0.269	73%
A – AST (LightGBM)	0.665	0.638	4%
B – Qwen (balanced)	0.698	0.449	36%
C – Ensemble	0.911	0.714	22%

Table 10: Validation-to-test degradation across tasks and model types. Structural features are substantially more stable than neural representations on Task A, while Task B and Task C also exhibit clear OOD degradation.

Step	Epoch	Loss	Val F1	Val Acc.
500	0.3	40.29	0.492	92.6%
1,000	0.6	2.86	0.574	94.7%
2,000	1.3	1.66	0.632	95.1%
3,500	2.2	0.71	0.665	95.6%
7,815	5.0	0.60	**0.698**	95.8%

Table 11: Task B training progression. Validation macro-F1 improves substantially during training, indicating that class weighting helps the model move beyond majority-class behavior.

Class	Prec.	Rec.	F1	Support
0 (Human)	0.995	0.998	0.997	88,490
1 (Gen-1)	0.745	0.628	0.682	847
2 (Gen-2)	0.480	0.557	0.516	1,755
3 (Gen-3)	0.597	0.460	0.520	650
4 (Gen-4)	0.642	0.614	0.628	445
5 (Gen-5)	0.798	0.734	0.765	372
6 (Gen-6)	0.649	0.657	0.653	1,118
7 (Gen-7)	0.594	0.560	0.577	1,695
8 (Gen-8)	0.783	0.789	0.786	1,579
9 (Gen-9)	0.566	0.539	0.552	895
10 (Gen-10)	0.837	0.793	0.814	2,154
Macro	**0.699**	**0.666**	**0.681**	100K

Table 12: Task B validation per-class results. The weighted model maintains usable performance across all 11 classes despite severe imbalance.

Qwen	TF-IDF	CodeBERT	Val F1
0.70	0.20	0.10	0.91051
0.75	0.20	0.05	0.91048
0.70	0.15	0.15	0.91045
0.80	0.20	0.00	0.91042
0.65	0.20	0.15	0.91038

Table 13: Top ensemble configurations for Task C. Differences among the best settings are very small, suggesting that performance is stable under nearby weight choices.

Class	Qwen	CBERT	TFIDF	Ens.
0 (54%)	0.988	0.949	0.907	**0.988**
1 (23%)	0.894	0.809	0.735	**0.896**
2 (9%)	0.875	0.680	0.609	**0.876**
3 (13%)	0.882	0.754	0.628	**0.883**
Macro	0.910	0.798	0.720	**0.911**

Table 14: Task C per-class F1. The ensemble provides the strongest overall balance across classes.

D Computational Resources

Table 15 details the computational cost for each component.

Component	GPU-h	Train	Infer
Task A – Qwen	6	1.5h	2h
Task A – AST/LightGBM	0	0.5h	0.5h
Task B – Qwen (5 epochs)	60	7.5h	5h
Task C – Qwen	48	6h	3h
Task C – CodeBERT	5	5h	0.75h
Task C – TF-IDF	0	15m	5m
Total	**119**		

Table 15: Full computational resource breakdown. GPU-h reflects total GPU time; Train/Infer are wall-clock.

E Hyperparameter Details

Table 16 provides the LoRA configuration shared across all Qwen fine-tuning experiments.

F Reproducibility

Software Versions. Python 3.10, PyTorch 2.1.0 (CUDA 12.1), Transformers 4.36.0, PEFT 0.7.0, Accelerate 0.25.0, scikit-learn 1.3.2, LightGBM 4.1.0, tree-sitter 0.20.4.

Deterministic Settings. We fix random seed 42 for Python, NumPy, and PyTorch, and enable deterministic cuDNN behavior for reproducibility.

Parameter	Value
Rank (r)	16
Alpha (α)	32
Dropout	0.05
Target modules	q/k/v/o_proj, gate/up/down_proj
Trainable parameters	18.5M (1.2%)
Optimizer	AdamW (β_1=0.9, β_2=0.999)
Weight decay	0.01
Learning rate	2×10^{-4}
Warmup	Linear, 6% of steps
Effective batch size	128
Precision	bfloat16

Table 16: LoRA and training configuration for Qwen fine-tuning.

```
random.seed(42); np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed_all(42)
torch.backends.cudnn.deterministic
    = True
```

G Task B Confusion Patterns

Table 17 shows the most common misclassification pairs from Task B validation, providing insight into which generators produce the most similar code.

Confusion Pair	Errors	Analysis
Human → Gen-2 Gen-2 → Human	142 778	Gen-2 mimics human style
Human ↔ Gen-3	∼700	Gen-3 also human-like
Gen-7 ↔ Gen-10	∼180	Similar architectures?
Human ↔ Gen-7	∼400	Gen-7 moderately confusable

Table 17: Task B top confusion pairs. Human↔AI confusion dominates, particularly for Generators 2 and 3 which produce the most human-like code. Inter-generator confusion is smaller but notable between Generators 7 and 10.

The asymmetric error pattern for Generator-2 is notable: 778 samples were incorrectly classified as Human (false negatives), but only 142 Human samples were misclassified as Generator-2 (false positives). This suggests Generator-2 produces code that genuinely resembles human writing, rather than the model being biased toward predicting Generator-2. The highest inter-generator confusion (Gen-7↔Gen-10) may indicate shared underlying model architectures or training data between these generators.