Distilling Semantics: Efficient Multi-label Code Comment Classification

Muhammad Abdul Majeed
Department of Computer Science
FAST-NUCES Islamabad
Islamabad, Pakistan
i221216@nu.edu.pk

Ahmed Bin Asim
Department of Computer Science
FAST-NUCES Islamabad
Islamabad, Pakistan
i220949@nu.edu.pk

***Abstract*—Automatic classification of code comments supports program comprehension and maintenance, but remains challenging due to label imbalance and semantically subtle categories. We present a lightweight knowledge distillation pipeline for the NLBSE'26 multi-label code comment classification task, distilling from CodeBERT to a MiniLM student. Training and evaluation are performed strictly on the official NLBSE dataset. We additionally use three small semantic booster CSV files *only during teacher fine-tuning* to better expose rare semantic patterns; the student is trained on the official training split and learns from teacher guidance via distillation. Distillation uses per-sample standardized logit matching and mild semantic label weighting. Across Java, Python, and Pharo, the student achieves macro F1 = 0.6689 with average runtime 0.7955 s and 769.97 GFLOPs, resulting in a final submission score of 0.74.**

***Index Terms*—Code comments, knowledge distillation, multi-label classification, software engineering**

I. Introduction

Code comments encode crucial information that is not always explicit in source code: usage guidance, parameter details, developer intent, design rationale, and maintenance notes. Automatically classifying comment sentences into functional categories can support downstream tooling such as documentation quality analysis, comment linting, maintenance analytics, and technical debt monitoring [4].

The NLBSE'26 Tool Competition [1] frames this as a multi-label classification task across three programming languages (Java, Python, Pharo). The task is not only accuracy-driven: the official scoring incorporates efficiency (runtime and GFLOPs). In practice, this introduces a difficult tradeoff: larger transformer models typically achieve stronger F1, but incur high latency and compute that reduce the final score.

A second key challenge is label imbalance. Common labels (e.g., summary-like categories) dominate the dataset, whereas semantically subtle labels (e.g., development notes, collaborator relationships, rationale) have low prevalence and are harder to learn. Under such imbalance, standard training tends to overfit majority labels and under-represent minority semantics.

To address both efficiency and semantic robustness, we distill a code-aware teacher (CodeBERT) into a compact student (MiniLM). Unlike standard distillation, we explicitly stabilize the knowledge transfer for multi-label logits using per-sample standardization. We also introduce teacher-only semantic booster exposure to improve teacher behavior on rare categories without contaminating the student's hard-label training distribution.

Contributions.

- **Teacher-only semantic boosters:** three small CSV booster sets used only during teacher fine-tuning to better expose rare semantic patterns.
- **Standardized logit distillation:** per-sample logit standardization before MSE matching to prevent gradient domination in multi-label distillation.
- **Semantic label weighting:** mild label-wise weighting inside the KD term to better preserve teacher behavior on semantically subtle labels.

II. Related Work

Pascarella and Bacchelli [4] introduced a taxonomy of Java code comment types, highlighting that comments differ widely in purpose and semantics. Rani et al. [2] extended multi-language classification and showed that semantically subtle labels are difficult under imbalance, particularly when comments are short or contain ambiguous intent.

Transformer-based encoders substantially improved performance, but they are often expensive for real-time use [3]. Knowledge distillation is a standard approach for compressing large language models into small students, but in multi-label settings naive logit matching can be unstable when teacher and student logit scales differ across samples. Our method directly targets this instability via per-sample standardization and focuses distillation on semantic structure rather than absolute logit magnitude.

III. Dataset and Characterization

A. Official Dataset Usage

We use the official dataset released for NLBSE'26 via HuggingFace. No alternative corpora or augmented training sets are used. The dataset contains language-specific train and test splits with a multi-hot label vector per example.

Listing 1. Official dataset loading.

```python
from datasets import load_dataset
ds = load_dataset("NLBSE/nlbse26-code-comment-
    classification")
```

TABLE I
LABEL TAXONOMY PER LANGUAGE (OFFICIAL).

Lang	Labels
Java	summary, Ownership, Expand, usage, Pointer, deprecation, rational
Python	Usage, Parameters, DevelopmentNotes, Expand, Summary
Pharo	Keyimplementationpoints, Example, Responsibilities, Intent, Keymessages, Collaborators

We use the dataset field `comment_sentence` as the input text for classification. Labels are provided as multi-hot vectors (one vector per instance) aligned with the label sets in Table I.

B. Label Taxonomy

The task uses language-specific label sets (7 for Java, 5 for Python, 6 for Pharo). Table I lists all labels.

C. Dataset Size, Imbalance, and Length

Table II summarizes train/test sizes and key dataset characteristics, including: average labels per comment, min/max label prevalence within each language, and token-length percentiles (based on the tokenizer used in analysis).

Implications. Java exhibits extreme long-tail behavior: very long comments exist (max 520 tokens) and the rarest labels have very low prevalence. Python is shorter on average (P95 19 tokens), which can reduce context for subtle categories, while Pharo exhibits moderate length and imbalance. Across all languages, the average number of active labels per comment is close to 1, meaning most examples are effectively single-label in practice, despite the multi-label setup.

IV. METHODOLOGY

A. Overview

Our pipeline consists of:

1) Fine-tune a CodeBERT teacher on the official training split, with *teacher-only* semantic booster CSV exposure.
2) Train a MiniLM student on the official training split using a mixture of hard-label supervision and standardized logit distillation from the teacher.
3) Tune multi-label decision thresholds on validation data and evaluate on the official test splits.
4) Measure runtime and GFLOPs using the official-style averaged measurement procedure.

B. Model Architecture and Training

Teacher: CodeBERT (microsoft/codebert-base, 125M parameters) fine-tuned on the datasets. Input length: 128 tokens, batch size: 16, epochs: 12 (Java), 25 (Python/Pharo), optimizer: AdamW, mixed precision: FP16.

Student: sentence-transformers/all-MiniLM-L6-v2 (22M parameters, 82% reduction). Input length: 64 tokens, batch size: 32, epochs: 25 (Java), 40 (Python), 50 (Pharo), optimizer: AdamW, mixed precision: FP16.

Threshold Optimization: We employ two-stage tuning on validation set: (1) global search for best single threshold $t^* \in [0.2, 0.6]$, (2) per-label refinement searching $t_j \in [t^* - 0.2, t^* + 0.2]$ for each label j.

C. Teacher-only Semantic Boosters

We identify semantic categories with low frequency less then (5%) and high complexity through empirical analysis. We use three small semantic booster CSV files (one per language) *only during teacher fine-tuning*. These boosters are designed to expose the teacher to rare semantic patterns (e.g., developer notes, collaborator relationships) in a controlled form.

- Java: Expand (4.1%), rational (3.2%)
- Python: DevelopmentNotes (6.4%), Expand (8.7%)
- Pharo: Responsibilities (12.3%), Collaborators (2.8%)

The student is *not* trained on these booster samples as hard labels. The booster effect is transferred indirectly via teacher logits during distillation.This design aims to improve rare-label sensitivity without shifting the student's empirical training distribution away from the official dataset. These synthetic examples follow three principles: (1) pattern reinforcement highlighting category specific language, (2) contextual diversity covering various software engineering contexts, and (3) controlled simplicity providing clear learning signals.

D. Teacher Fine-tuning Objective

Let x be a comment sentence and $y \in \{0,1\}^C$ the multi-label target. The teacher f_T outputs logits $z_T \in \mathbb{R}^C$. Teacher fine-tuning uses weighted binary cross-entropy:

$$\mathcal{L}_{\text{teacher}} = \text{BCEWithLogits}(z_T, y; w_{\text{pos}}),$$

where w_{pos} is computed per label to counter imbalance (rare labels get higher positive weight).

E. Why Standardized Logit Distillation

In multi-label classification, each label behaves like an independent binary classifier. A common KD approach matches teacher and student logits using MSE. However, logits often differ in scale across models and across samples; high-magnitude logits can dominate the gradient and drown out learning signals for subtle labels.

We address this by normalizing logits *per sample across labels*. This makes the KD signal emphasize the teacher's *relative* preferences among labels for each comment, improving stability.

F. Standardized Knowledge Distillation

Let $z_T^i, z_S^i \in \mathbb{R}^C$ be teacher/student logits for sample i. We standardize:

$$\tilde{z}^i = \frac{z^i - \mu(z^i)}{\sigma(z^i) + \epsilon}.$$

The standardized KD loss is:

$$\mathcal{L}_{\text{KD}} = \frac{1}{NC} \sum_{i=1}^{N} \sum_{j=1}^{C} (\tilde{z}_T^{ij} - \tilde{z}_S^{ij})^2.$$

TABLE II
DATASET SIZES AND KEY CHARACTERISTICS (TRAIN SPLIT).

Lang	Train	Test	Labels	Avg labels	Min prev (%)	Max prev (%)	P95 tokens	Max tokens
Java	5394	1201	7	1.036522	1.483129	46.996663	54.0	520.0
Python	1368	290	5	1.121345	12.207602	31.505848	19.0	39.0
Pharo	900	208	6	1.122222	5.555556	41.555556	32.0	94.0

G. Semantic Label Weighting

To preserve behavior on semantically subtle categories, we apply mild label-wise weights w_j inside the KD term:

$$\mathcal{L}_{\text{KD}}^{\text{weighted}} = \frac{1}{NC} \sum_{i=1}^{N} \sum_{j=1}^{C} w_j \, (\tilde{z}_T^{ij} - \tilde{z}_S^{ij})^2.$$

Weights are set higher for selected semantic-hard labels (language-specific) and default for others.

H. Final Student Objective

The student is trained with a convex combination of hard-label supervision and KD:

$$\mathcal{L}_{\text{student}} = (1-\alpha)\mathcal{L}_{\text{hard}} + \alpha \mathcal{L}_{\text{KD}}^{\text{weighted}},$$

where $\mathcal{L}_{\text{hard}}$ is weighted BCE with logits on the official training split.

I. Thresholding for Multi-label Prediction

Since outputs are multi-label, we convert predicted probabilities into binary decisions via thresholds. A single global threshold tends to over-predict frequent labels and under-predict rare labels. We therefore tune thresholds on validation data using a global sweep followed by light per-label refinement, prioritizing macro F1.

J. Efficiency Measurement (Runtime and GFLOPs)

Runtime and GFLOPs are measured using repeated inference runs and averaged, matching the official-style protocol. We report average runtime per test set and average GFLOPs (averaged over languages).

V. EXPERIMENTAL EVALUATION

A. Metrics and Official Score

We report:

- Macro F1 (mean across all label categories across all languages).
- Runtime per test set (seconds), averaged across languages.
- GFLOPs, averaged across languages.
- Official submission score:

$$\text{Score} = 0.6 \times \text{F1} + 0.2 \times \text{Efficiency}_{\text{time}} + 0.2 \times \text{Efficiency}_{\text{FLOPs}}.$$

B. Score Breakdown

Table III decomposes the official score into its components. The gain is driven by both macro F1 and efficiency, with a large contribution from GFLOPs reduction.

TABLE III
SCORE DECOMPOSITION (RECOMPUTED).

Model	F1 term	Runtime term	GFLOPs term	Total
Baseline	0.390491	0.163878	0.143803	0.698172
Current	0.401326	0.168181	0.169201	**0.738708**

TABLE IV
PER-LANGUAGE MACRO F1 (MEAN OVER LABELS).

Lang	Baseline	Current	Δ	% change
Java	0.730584	0.706163	-0.024421	-3.342705
Python	0.581946	0.634076	0.052130	8.957882
Pharo	0.615150	0.654377	0.039227	6.376876

C. Per-language Macro F1 (Baseline → Current)

While overall macro F1 improves, the gains are not uniform across languages. Table IV reports per-language macro F1 computed as the mean over that language's label categories.

Interpretation. The final pipeline substantially improves Python and Pharo, which contributes to the overall score improvement. Java decreases slightly in macro F1, but efficiency improvements and gains in other languages still yield a higher final submission score.

D. Per-category Changes (Baseline → Current)

To understand where improvements occur, Table V shows per-category F1 and deltas. Large gains concentrate on rare semantic categories (e.g., Pharo *Collaborators*, Python *DevelopmentNotes*). The largest drops occur in Java *deprecation* and Python *Summary*.

E. Qualitative Error Patterns

Although our evaluation is primarily metric-driven, the observed behavior across categories suggests common error types:

- **Boundary ambiguity:** short comments can plausibly fit multiple categories (e.g., a statement may act as summary or usage depending on context).
- **Multi-intent comments:** some comments naturally express multiple functions, but annotations may represent only a subset, introducing label noise.
- **Rare-label brittleness:** for minority categories, small lexical differences can flip predictions; teacher-only semantic boosters aim to improve teacher sensitivity in these cases, and weighting helps preserve it in the student.

TABLE V
PER-CATEGORY F1 CHANGE (BASELINE → CURRENT).

Language	Category	F1 (Baseline)	F1 (Current)	ΔF1
java	summary	0.878909	0.887490	0.008581
java	Ownership	1.000000	0.982456	-0.017544
java	Expand	0.373626	0.409091	0.035465
java	usage	0.867012	0.860034	-0.006978
java	Pointer	0.861210	0.870504	0.009294
java	deprecation	0.777778	0.615385	-0.162393
java	rational	0.355556	0.318182	-0.037374
python	Usage	0.673913	0.717949	0.044036
python	Parameters	0.719101	0.793103	0.074002
python	DevelopmentNotes	0.304762	0.475000	0.170238
python	Expand	0.539683	0.563636	0.023953
python	Summary	0.672269	0.620690	-0.051579
pharo	Keyimplementationpoints	0.600000	0.567568	-0.032432
pharo	Example	0.881356	0.868132	-0.013224
pharo	Responsibilities	0.681319	0.660377	-0.020942
pharo	Intent	0.782609	0.863636	0.081027
pharo	Keymessages	0.578947	0.591549	0.012602
pharo	Collaborators	0.166667	0.375000	0.208333

VI. DISCUSSION

A. Why Teacher-only Boosters (and not student augmentation)

A key design decision is to keep the student's hard-label training data strictly aligned with the official dataset distribution. Mixing synthetic or booster examples into student hard-label training can shift empirical priors and reduce calibration for frequent categories. Our approach instead:

1) strengthens the teacher on rare semantics using small booster exposure, then
2) transfers that improved decision structure to the student via distillation.

This keeps student supervision "clean" while leveraging the teacher as an information bottleneck.

B. Why Standardization Helps Multi-label Distillation

In multi-label settings, matching logits directly is attractive because softmax-based KD is not naturally aligned with independent binary decisions. However, raw logits can be poorly calibrated across labels and across models, and MSE can overemphasize high-magnitude outputs. Per-sample standardization re-weights the KD signal to emphasize semantic structure across labels for each example, which is especially useful when rare labels need to be learned from small, subtle cues.

C. Where the Improvements Come From

The baseline-to-current deltas indicate that the strongest gains occur in rare semantic categories:

- Pharo *Collaborators* (+0.2083)
- Python *DevelopmentNotes* (+0.1702)
- Pharo *Intent* (+0.0810)

These are precisely the categories most likely to benefit from teacher strengthening and semantic-aware distillation. At the same time, some frequent categories drop (e.g., Java *deprecation*, Python *Summary*), suggesting that additional work is needed on calibration and thresholding for these labels.

VII. CONCLUSION

We presented a lightweight, semantic-aware knowledge distillation pipeline for the NLBSE'26 multi-label code comment classification task. The system trains and evaluates strictly on the official dataset, uses three small semantic booster CSV files only during teacher fine-tuning, and transfers behavior to a MiniLM student via standardized logit matching and semantic label weighting.

Across Java, Python, and Pharo, the final student achieves macro F1 = 0.6689 with average runtime 0.7955 s and 769.97 GFLOPs. Under the official scoring, the submission score improves from 0.6982 to **0.74** (rounded). These results demonstrate that careful distillation design can preserve semantic competence while dramatically improving deployability.

ACKNOWLEDGMENTS

We thank the NLBSE'26 organizers for providing the datasets and evaluation framework. We also acknowledge the HuggingFace and PyTorch communities for their open-source tools.

REFERENCES

[1] M. Mock, P. Rani, F. Santos, B. Carter, and J. Penney, "The NLBSE'26 Tool Competition," in *Proceedings of The 5th International Workshop on Natural Language-based Software Engineering (NLBSE'26)*, 2026.

[2] P. Rani, S. Panichella, M. Leuenberger, A. Di Sorbo, and O. Nierstrasz, "How to identify class comment types? A multi-language approach for class comment classification," *Journal of Systems and Software*, vol. 181, p. 111047, 2021.

[3] A. Al-Kaswan, M. Izadi, and A. Van Deursen, "STACC: Code Comment Classification using SentenceTransformers," in *2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE)*, 2023, pp. 28–31.

[4] L. Pascarella and A. Bacchelli, "Classifying code comments in Java open-source software systems," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 2017.