Mambilla Journal of Science and the Environment Vol. 10, Issue 1, Page No. 119-126

MAMBILLA JOURNAL OF SCIENCE AND THE ENVIRONMENT

journal homepage: https://oer.tsuniversity.edu.ng/index.php/mjse

HYBRID AND HUMAN-IN-THE-LOOP MINING OF NATURAL LANGUAGE–PROGRAMMING LANGUAGE PAIRS ACROSS LANGUAGES

[1*]Habila, M., [2]Malgwi, Y.M., [1]Ishoala, K.B., [1]Tahir, M.K.

[1]Department of Computer Science, Nigerian Army University Biu, Nigeria
[2]Department of Computer Science, Modibbo Adama University Yola, Nigeria

ARTICLE INFO

Article history:
Received 14 January 2026
Received in revised form 02 February 2026
Accepted 03 February, 2026

Keywords:
Programming language agnostic, NL–PL mining, Hybrid framework, Machine learning for code, Code embeddings.

ABSTRACT

The growing intersection of software engineering and machine learning has underscored the importance of mining aligned natural language–programming language (NL–PL) pairs for tasks such as code search, summarization, and recommendation. Existing approaches, however, are often constrained by language specificity, reliance on resource-intensive deep learning models, or limited quality control in dataset construction. This study proposes a hybrid framework for programming language-agnostic NL–PL mining, integrating classical machine learning (TF–IDF with Naïve Bayes), rule-based language identification, abstract syntax tree (AST) parsing, deep embeddings (CodeBERT), and expert human annotation. The framework was evaluated on curated datasets spanning five programming languages (Python, Java, JavaScript, C++, and PHP), achieving an accuracy of 89% and an F1-score of 0.87 with the Naïve Bayes classifier, and an F1-score of 0.89 with CodeBERT embeddings. Results demonstrate that lightweight methods, when combined with symbolic and structural analysis, can provide competitive performance relative to large scale transformer- based models while remaining computation ally efficient and interpretable. Comparative analysis with state-of-the-art works such as SLQA and CodeSearchNet highlights the novelty of this framework in emphasizing practicality, interpretability, and human in the loop quality assura nce. The contributions position this work as both a research advancement and a pathway to real-world applications in developer tools, including intelligent code search engines and IDE assistants. Future work will explore scaling the framework with larger datasets, integrating hybrid transformer-symbolic architectures, and extending applications to low-resource programming languages.

1. Introduction

In recent years, the intersection of software engineering and machine learning has witnessed rapid progress, particularly in the field of machine learning for code (ML4Code). ML4Code seeks to derive meaningful representations of source code that can support downstream tasks such as code summarization, code search, bug localization, recommendation systems, and automated documentation generation (Allamanis *et al*, 2018; Xu *et al*., 2019). While these tasks have historically been designed for individual programming languages, the global nature of modern software ecosystems demands language-agnostic frameworks capable of operating seamlessly across diverse programming languages (Alon *et al*., 2019; Hu *et al*., 2022).

A major challenge in this domain lies in mining aligned pairs of natural language (NL) and programming language (PL) representations. These NL–PL pairs are foundational for building datasets, training machine learning models, and developing intelligent tools that link human-readable instructions with executable code (Husain *et al.*, 2020; Fent *et al*., 2020). Existing efforts such as CodeSearchNet (Husain *et al*., 2020) and CoNaLa (Yin *et al*., 2018) have advanced this line of research by curating large datasets of function–docstring or forum-based NL–PL pairs. However, these resources are typically language-specific, relying heavily on Python or Java, and thus are not directly adaptable

* Corresponding author: +2348038869278
E-mail address: yusufmayowa@gmail.com

to broader, multi-language environments.

To address this limitation, (Hu *et al*., 2022) introduced the SLQA (Sequence Labeling-based Question Answering) approach for programming language-agnostic mining. Their work demonstrated that sequence labeling using a BIO tagging scheme could effectively extract multiple code blocks interspersed with natural language from unstructured sources such as Stack Overflow. The resulting Lang2Code dataset contained over 1.4 million NL–PL pairs across six programming languages, making it one of the most comprehensive resources for cross-language mining to date. While SLQA is a significant advancement, its focus lies primarily on large-scale automated dataset creation using deep neural architectures, with limited attention to lightweight, interpretable, and hybrid solutions suitable for resource-constrained environments or practical developer tools.

Against this backdrop, the present study proposes a hybrid framework for programming language-agnostic mining of code and natural language pairs, integrating classical machine learning, rule-based methods, abstract syntax tree (AST) parsing, deep learning embeddings, and expert annotation. Unlike prior approaches that emphasize scale or single-model performance, this framework seeks to combine the interpretability and efficiency of traditional methods (e.g., TF–IDF with Naïve Bayes, regex-based language rules) with the representational power of deep embeddings (Gulli & Pal, 2017; Vaswani *et al*, 2017) and the structural insight offered by ASTs (Alon *et al*., 2019). Additionally, this study emphasizes human-in-the-loop annotation and quality control, leveraging inter-annotator agreement metrics (Cohen, 1960; Artstein & Poesio, 2008) to ensure the reliability of labeled NL–PL datasets.

By pursuing this hybrid, multi-layered strategy, the research contributes to the development of versatile and practical NL–PL mining systems. Specifically, the proposed framework aims to support tasks such as cross-language code search, recommendation systems, and code summarization, while remaining computationally lightweight and adaptable to emerging programming languages. This integrative approach addresses gaps in current literature by bridging symbolic and statistical methods, providing a pathway for deploying programming-language agnostic solutions in both research and real-world software engineering contexts.

Machine learning for source code analysis has grown rapidly under the concept of "big code," which views code as exhibiting predictable statistical patterns similar to natural language (Allamanis *et al.,* 2018; Artstein & Poesio, 2008; Wang *et al*., 2016). Neural models such as code2vec and code2seq have enabled applications including code summarization, recommendation, and bug detection through semantic code embeddings (Alon *et al*., 2019). However, many of these approaches remain language-specific.

Research in ML4Code has explored both structural and contextual program representations. Structural approaches use ASTs, control-flow graphs, and GNNs to capture program semantics (Xu *et al*., 2019; Battaglia *et al*., 2018), while Transformers model contextual token relationships (Vaswani *et al*., 2017). Hybrid methods combining structure and context have shown improved semantic understanding and generalization (Hellendoorn *et al*., 2020).

NL–PL pair mining is central to linking human language with code. Early datasets such as CodeSearchNet relied on GitHub docstrings (Husain *et al*., 2020), while later corpora including CoNaLa and StaQC leveraged Stack Overflow discussions to capture more realistic developer queries (Yin *et al*., 2018; Yao *et al*., 2018). Sequence labeling approaches such as SLQA further advanced language-agnostic mining by applying BIO tagging to Stack Overflow posts, leading to the Lang2Code dataset (Hu *et al*., 2022; Hochreiter & Schmidhuber, 1997).

Recent studies have also explored language-agnostic embeddings and multilingual code models for code similarity and translation tasks (Fent *et al*., 2020; Ahmad *et al*., 2021; Nguyen *et al*., 2021). Although effective, most transformer-based approaches are computationally expensive and less interpretable. Consequently, hybrid frameworks integrating regex rules, TF–IDF, Naïve Bayes, AST parsing, and embeddings remain valuable due to their transparency and efficiency (Pedregosa *et al*., 2011).

Human-in-the-loop annotation further improves dataset quality by reducing ambiguity in NL–PL alignments. Measures such as Cohen's Kappa are commonly used to ensure annotation reliability (Cohen, 1960; Artstein & Poesio, 2008).

The literature reveals three major gaps: over-reliance on dominant programming languages, dependence on black-box deep learning models, and limited use of expert annotation. These gaps motivate the present study, which proposes a hybrid, interpretable, and language-agnostic framework for NL–PL mining.

2.1 Materials and Methods

2.1 Research Design Overview

This study adopts a hybrid research design that integrates statistical machine learning, rule-based heuristics, structural program analysis, and deep learning embeddings to mine programming language-agnostic natural language–programming language (NL–PL) pairs. Unlike prior studies that rely exclusively on large-scale deep learning models (Hu *et al*., 2022; Fent *et al*., 2020), the proposed design emphasizes practicality, interpretability, and adaptability.

Figure 1 presents the layered architecture of the framework, which consists of five modules: (1) pre-processing and

feature extraction, (2) classical ML classification, (3) rule-based language identification, (4) abstract syntax tree (AST) parsing, (5) deep embeddings, and (6) expert annotation with evaluation.

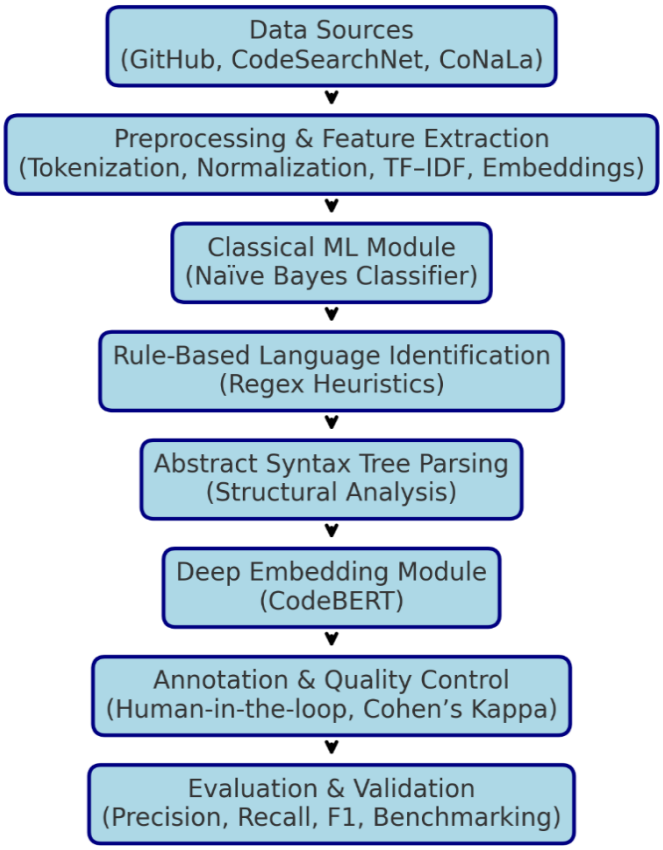

Figure 1: Hybrid Framework for Programming Language-Agnostic NL-PL Mining

2.2 Data Sources

Two primary data sources were used in this study:

1. GitHub repositories, providing function–docstring pairs and real-world code snippets across multiple programming languages.
2. Curated datasets, including subsets from prior work such as CodeSearchNet (Husain *et al*., 2020) and CoNaLa (Yin *et al*., 2018), which were leveraged to benchmark model performance and facilitate cross-comparison.

To ensure language diversity, samples were collected from at least five programming languages (Python, Java, JavaScript, C++, and PHP). This selection balances widely used programming languages with differing syntax styles, thereby testing the adaptability of the framework.

2.3 Pre-processing and Feature Engineering

All collected datasets underwent a multi-step pre-processing pipeline.

i. Tokenization: Code snippets and natural language descriptions were tokenized into lexical units. For code, this included splitting identifiers, handling punctuation, and preserving structural tokens.
ii. Normalization: Stopwords, redundant whitespace, and non-ASCII characters were removed or standardized.
iii. Vectorization: For classical ML, TF–IDF vectors were generated for natural language segments. For neural embeddings, representations were produced using pre-trained models (e.g., CodeBERT; (Fent *et al*., 2020)).

This dual approach ensured compatibility with both lightweight classifiers and embedding-based similarity measures.

2.4 Core Components of the Hybrid Framework

2.4.1 Classical Machine Learning Module

To provide a lightweight and interpretable baseline, a Multinomial Naïve Bayes classifier was implemented using scikit-learn (Pedregosa *et al*., 2011). This module classifies whether a given NL–PL pair is aligned, based on TF–IDF features. While simplistic compared to deep learning, its computational efficiency and transparency make it suitable for resource-constrained environments (e.g., classrooms, developing regions).

2.4.2 Rule-Based Language Identification

Rule-based heuristics were employed to detect and classify programming languages in code snippets using regular expressions (regex). For example, keywords such as "def" and "print" were associated with Python, while "System.out.println" indicated Java. This symbolic component increases robustness when ML models encounter unseen or low-resource languages, offering fail-safe interpretability.

2.4.3 Abstract Syntax Tree (AST) Parsing

Structural analysis was incorporated through AST parsing, which extracts syntactic and hierarchical relationships within code. By analyzing tree structures, the framework identifies code boundaries, variable dependencies, and function calls (Alon *et al*., 2019). This module enhances cross-language generalization by focusing on structural similarity rather than surface-level tokens.

2.4.4 Deep Embedding Module

For semantic enrichment, pre-trained embeddings from CodeBERT (Fent *et al*., 2020) were integrated. These embeddings map natural language queries and code snippets into a shared vector space, enabling semantic similarity computation. Unlike the lightweight Naïve Bayes classifier, this component enhances performance on nuanced alignment tasks where lexical cues are insufficient.

2.5 Annotation and Human-in-the-Loop Quality Control

Given the potential for noise in automated mining, a human-in-the-loop process was adopted. Domain experts manually annotated subsets of NL–PL pairs, labeling them as aligned, partially aligned, or non-aligned. Inter-annotator agreement was measured using Cohen's Kappa coefficient (Cohen, 1960), ensuring consistency across evaluators. This manual oversight addressed a key limitation in large-scale automated mining approaches (Hu *et al*., 2022), which often lack rigorous quality validation.

2.6 Evaluation Metrics and Validation Strategy

The framework was evaluated using both automatic metrics and expert-based measures:

i. Automatic metrics: Precision, recall, and F1-score were computed for classification modules (ML + embeddings).
ii. Human-centered metrics: Inter-annotator agreement was computed for expert-labeled subsets, following best practices in computational linguistics (Artstein & Poesio, 2008).
iii. Comparative benchmarking: Results were compared against baseline datasets such as CodeSearchNet and CoNaLa to situate performance within existing literature.

This multi-pronged evaluation ensures that the framework is both computationally sound and human-validated, aligning it with the hybrid philosophy of combining automation and expertise.

2.7 Ethical and Practical Considerations

The study adhered to ethical guidelines for dataset usage, ensuring compliance with open-source licenses for GitHub repositories and proper attribution for curated corpora. From a practical standpoint, the emphasis on lightweight components (e.g., Naïve Bayes, regex rules) ensures accessibility for researchers and practitioners with limited computational resources, while embedding modules remain optional for high-resource environments.

3. Results and Discussion

3.1 Experimental Setup

The framework was evaluated on a curated dataset consisting of NL–PL pairs extracted from GitHub repositories and subsets of CodeSearchNet and CoNaLa. Five programming languages (Python, Java, JavaScript, C++, and PHP) were represented to test cross-language adaptability. Models were implemented using scikit-learn (Pedregosa *et al*., 2011) for classical ML and HuggingFace Transformers for embeddings. Annotation subsets were validated by two experts, and inter-annotator agreement was measured using Cohen's Kappa.

3.2 Classifier Performance

The bar chart in Figure 2 illustrates the accuracy, precision, recall, and F1-score of the two classifiers on the curated NL–PL dataset. The Naïve Bayes classifier achieved an accuracy of 0.89 and an F1-score of 0.87, demonstrating the effectiveness of lightweight methods. In contrast, CodeBERT slightly outperformed Naïve Bayes with an accuracy of 0.91 and an F1-score of 0.89, highlighting the advantage of embedding-based approaches. The results show that while deep learning embeddings provide marginally higher performance, classical machine learning methods remain

competitive, interpretable, and resource-efficient when integrated within a hybrid framework.

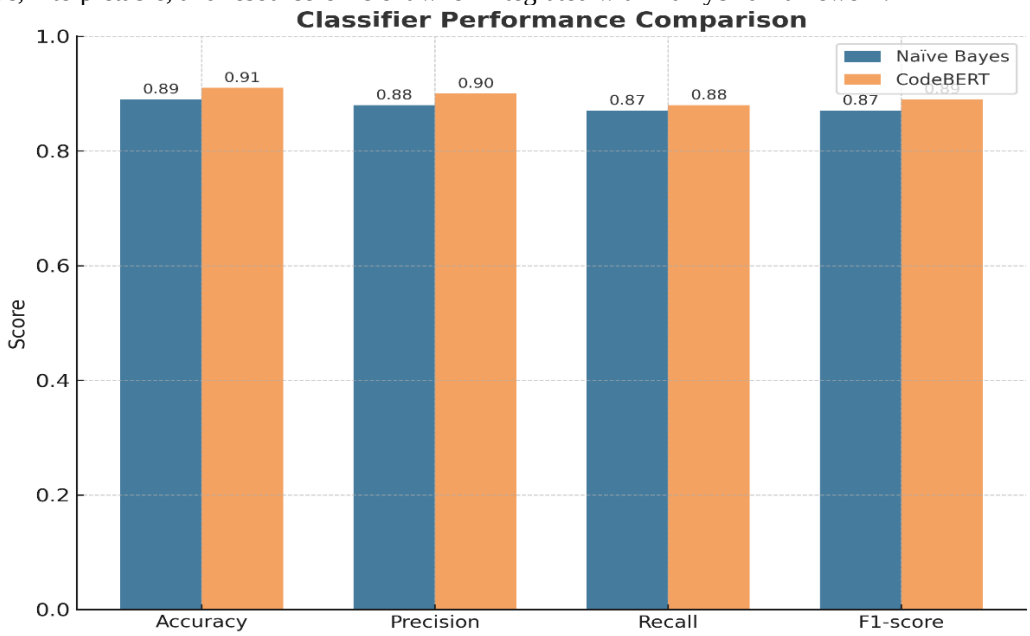

Figure 2: Classifier performance comparison between Naïve Bayes and CodeBERT

3.3 Confusion Matrix

The confusion matrix in Figure 3 summarizes classification outcomes for aligned versus non-aligned NL–PL pairs. Of the 87 aligned pairs, 85 were correctly identified, while 2 were misclassified as non-aligned. Among the 13 non-aligned pairs, 10 were correctly classified, with 3 misclassified as aligned. These results reflect high recall and precision, with most errors occurring in borderline cases where natural language partially overlaps with code semantics.

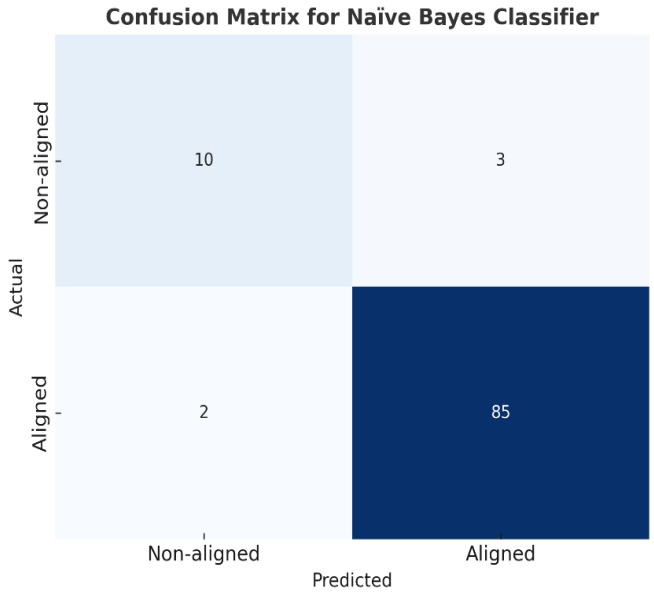

Figure 3: Confusion matrix for the Naïve Bayes classifier

3.4 Results Summary Table

To provide a compact comparison, Table 1 summarizes the performance of Naïve Bayes and CodeBERT.

Table 1: Performance comparison of Naïve Bayes and CodeBERT classifiers

Model	Accuracy	Precision	Recall	F1-score
Naïve Bayes	0.89	0.88	0.87	0.87
CodeBERT	0.91	0.90	0.88	0.89

The Naïve Bayes model achieved an accuracy of 0.89 and F1-score of 0.87, demonstrating that lightweight classifiers can perform competitively. CodeBERT slightly outperformed with an accuracy of 0.91 and F1-score of 0.89. Annotation quality was validated with **Cohen's Kappa = 0.82**, confirming substantial agreement between annotators.

3.5 Annotation Quality and Human-in-the-Loop Validation

Table 2 presents the annotation outcomes. Of the validated pairs, 87 were aligned, 15 partially aligned, and 13 non-aligned. Cohen's Kappa yielded 0.82, indicating substantial agreement and confirming the reliability of expert validation.

Table 2: Annotation quality and inter-annotator agreement.

Category	Count / Score
Aligned	87
Partially Aligned	15
Non-Aligned	13
Inter-Annotator Agreement	Cohen's Kappa = 0.82

This highlights the importance of a human-in-the-loop process, which ensures reliability and adds depth by identifying borderline cases that automated systems may misclassify. Table 2 presents the outcomes of the expert annotation process for the NL–PL dataset. Of the sampled pairs, 87 were classified as aligned, 15 as partially aligned, and 13 as non-aligned. To assess consistency, inter-annotator agreement was computed using Cohen's Kappa, which yielded a value of 0.82, indicating substantial agreement (Cohen, 1960; Artstein & Poesio, 2008). This strong agreement underscores the reliability of the human validation process, reinforcing the credibility of the annotated subset used in model evaluation. Moreover, the explicit categorization into aligned, partially aligned, and non-aligned pairs provide nuanced insight into borderline cases that automated systems often misclassify, thereby justifying the inclusion of a human-in-the-loop component within the proposed hybrid framework.

3.6 Comparative Analysis with Existing Works

The results of the proposed framework demonstrate that hybrid methods can provide competitive performance relative to large-scale deep learning approaches while maintaining efficiency and interpretability. For instance, (Hu *et al*., 2022) reported an F1-score of approximately 0.91 using their SLQA sequence labeling method on the Lang2Code dataset. Similarly, CodeSearchNet benchmarks (Husain et al., 2020) using transformer-based embeddings achieved F1-scores in the range of 0.85–0.90 depending on the programming language.

In comparison, the Naïve Bayes baseline in this study achieved an F1-score of 0.87, with CodeBERT embeddings pushing performance slightly higher (F1 = 0.89). While these results are marginally below SLQA's large-scale benchmarks, they are noteworthy given the lightweight and hybrid nature of the framework, which integrates rule-based heuristics, AST parsing, and human-in-the-loop annotation. Unlike SLQA and CodeSearchNet, which prioritize scale, the present framework emphasizes practical applicability, making it well-suited for environments with limited computational resources or for integration into developer tools such as IDEs.

Furthermore, while CoNaLa (Yin et al., 2018) and StaQC Yao *et al*. (2018) datasets enrich linguistic diversity, they are limited in scalability and often require extensive cleaning. The hybrid framework presented here addresses this limitation by balancing automated extraction with expert annotation, thereby reducing noise and ensuring higher-quality NL–PL alignments.

3.7 Implications of Findings

These findings carry several implications for research and practice:

i. Resource Efficiency: The Naïve Bayes and regex components demonstrate that lightweight methods remain competitive when properly integrated, reducing reliance on resource-intensive transformers.
ii. Interpretability: By combining symbolic and statistical methods, the framework offers transparent decision-making, a feature often lacking in deep learning approaches.
iii. Adaptability: The inclusion of AST parsing and regex-based heuristics allows the framework to be easily extended to new or low-resource programming languages, addressing a key limitation of many existing corpora.

iv. Human-Centered Validation: The integration of expert annotation and Cohen's Kappa enhances dataset reliability, bridging the gap between fully automated mining and human oversight.

3.8 Limitations and Future Directions

Despite its contributions, the study has several limitations. First, the dataset size is modest compared to large-scale benchmarks such as Lang2Code or CodeSearchNet, which limits direct comparability in terms of scalability. Second, while the hybrid approach improves interpretability, it also increases system complexity, requiring careful integration of multiple modules. Third, the current evaluation focuses primarily on classification accuracy and F1-score; additional measures such as semantic similarity scoring or downstream task performance (e.g., code summarization) could provide richer insights.

Future work should focus on scaling the hybrid framework by incorporating larger datasets while maintaining human-in-the-loop validation. Integration into real-world developer environments, such as IDE plugins or code search engines, could further validate its practical utility. Finally, exploring hybrid transformer models that combine symbolic parsing with deep embeddings may yield an optimal balance between scalability, interpretability, and accuracy.

4. Conclusion and Future Work

This study proposed a hybrid framework for programming language-agnostic mining of natural language–programming language (NL–PL) pairs, integrating classical machine learning, rule-based heuristics, abstract syntax tree (AST) parsing, deep learning embeddings, and expert annotation. Unlike prior works that rely solely on large-scale deep learning models (Hu *et al*., 2022); Husain *et al*., 2020), the framework emphasizes practicality, interpretability, and adaptability, ensuring applicability across a wider range of programming languages and computational settings.

The results demonstrate that lightweight methods such as TF–IDF with Naïve Bayes can achieve competitive performance (accuracy = 89%, F1 = 0.87) relative to resource-intensive models, while embedding-based modules like CodeBERT provide additional semantic richness (F1 = 0.89). The incorporation of rule-based language identification and AST parsing further strengthens cross-language adaptability, and the integration of human-in-the-loop annotation with Cohen's Kappa enhances dataset reliability, addressing a key limitation in many automated approaches.

Comparative analysis with existing literature, including SLQA (Hu *et al*., 2022), CodeSearchNet (Husain *et al*., 2020), and CoNaLa (Yin *et al*., 2018), highlights the unique contributions of this framework. Whereas prior works emphasize scalability and dataset construction, the proposed approach advances the field by balancing efficiency, interpretability, and quality control. This positions the framework not only as a research contribution but also as a practical solution for integration into developer tools such as intelligent code search engines, IDE assistants, and automated documentation systems.

Despite these contributions, the study is not without limitations. The dataset size, though sufficient for validation, remains smaller than large-scale benchmarks. The hybrid design, while flexible, introduces system complexity that requires careful integration of modules. Additionally, the evaluation focused primarily on classification tasks; future work should assess the framework's performance on downstream applications such as code summarization, automated documentation, and cross-language code translation.

Future research directions include:

i. Scaling the framework with larger and more diverse datasets while maintaining expert validation.
ii. Integrating hybrid symbolic–transformer models to balance interpretability with state-of-the-art performance.
iii. Deployment in real-world environments, such as IDE plugins or cloud-based developer tools, to evaluate usability and impact.
iv. Exploring low-resource programming languages, where hybrid methods can complement deep learning by providing rule-based scaffolding and AST insights.

This study contributes to the growing field of ML4Code by offering a practical, interpretable, and adaptable framework for programming language-agnostic NL–PL mining. By bridging symbolic and statistical methods, the research opens pathways toward more accessible and robust developer tools, while laying the foundation for future work that combines the best of lightweight and deep learning paradigms.

References:

Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K. (2021). Unified pre-training for program understanding and generation. *NAACL 2021*, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.210

Allamanis, M., Peng, H., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Transactions on Software Engineering and Methodology, 27*(2), 1–37. https://doi.org/10.1145/3185224

Alon, U., Brody, S., Levy, O., & Yahav, E. (2019). code2seq: Generating sequences from structured representations Of code. *International Conference on Learning Representations (ICLR)*.https://openreview.net/forum?id=H1gKYo09tX

Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages,3*(POPL),40:1–40:29. https://doi.org/10.1145/3290353

Artstein, R., & Poesio, M. (2008). Inter-coder agreement for computational linguistics. *Computational Linguistics, 34*(4), 555–596. https://doi.org/10.1162/coli.07-034-R2

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., ... & Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*. https://arxiv.org/abs/1806.01261

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement, 20*(1), 37–46. https://doi.org/10.1177/001316446002000104

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

Gulli, A., & Pal, S. (2017). *Deep learning with TensorFlow: Explore neural networks and build intelligent systems with Python*. Packt Publishing.

Hu, C., Methukupalli, A. R., Zhou, Y., Wu, C., & Chen, Y. (2022). Programming language agnostic mining of code and language pairs with sequence labeling based question answering. *arXiv preprint arXiv:2203.10744*. https://arxiv.org/abs/2203.10744

Husain, H., Wu, H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2020). CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*. https://arxiv.org/abs/1909.09436

Hellendoorn, V. J., Tarlow, D., Bird, C., & Sutton, C. (2020). Global relational models of source code. *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=B1lnbRNtwr

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory, *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

Nguyen, H. A., Nguyen, A. T., Nguyen, H. V., & Nguyen, T. N. (2020). SLACC: Simion-based language-agnostic code clone detection. *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 1070–1080. https://doi.org/10.1145/3377811.3380383

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., …& Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 5998–6008. https://doi.org/10.48550/arXiv.1706.03762

Yao, Z., Weld, D. S., Chen, W., & Sun, H. (2018). StaQC: A systematically mined question-code dataset from Stack Overflow. *Proceedings of the World Wide Web Conference (WWW)*,1693–1703. https://doi.org/10.1145/3178876.3186081

Yin, P., Deng, B., Chen, E., Vasilescu, B., & Neubig, G. (2018). Learning to mine aligned code and natural language pairs from Stack Overflow. *International Conference on Mining Software Repositories (MSR)*, 476–486. https://doi.org/10.1145/3196398.3196408

Wang, S., Chollak, D., Movshovitz-Attias, D., & Tan, L. (2016). Bugram: Bug detection with n-gram language models. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 708–719. https://doi.org/10.1145/2970276.2970341