Evaluating Identifier Readability Using CodeBERT Embeddings and Self-Attention Bi-LSTM with Explainable Modeling

Bharat Babaso Mane
Department of Computer Science and Engineering, Alliance University, Bengaluru, Karnataka, India
bharat.mane@gmail.com (corresponding author)

Rathnakar Achary
Department of Computer Science and Engineering, Alliance University, Bengaluru, Karnataka, India
rathnakar.achary@alliance.edu.in

Received: 5 February 2026 | Revised: 12 March 2026 and 20 March 2026 | Accepted: 21 March 2026

Licensed under a CC-BY 4.0 license | Copyright (c) by the authors | *DOI: https://doi.org/10.48084/etasr.17996*

ABSTRACT

Identifier names are natural language representations in source code that play a significant role in program understanding. Studies on the quality of identifier names generally focus on their role in program knowledge. Identifier names and naming conventions are important for programming understanding. Previous studies have shown a connection between software quality and the quality of identifier names. Recently, Deep Learning (DL) has been used to develop highly efficient models for identifying intelligibility challenges. DL models, such as transformer-based frameworks and attention mechanisms, can acquire contextual and sequential relations between identified tokens. This study proposes an Identifier Readability Analysis Framework using an Explainable Attention-Based Deep Learning (IRAF-XADL) approach, with the primary intention of assessing the quality of identifier names in Python and C++ source code. In the initial stage, a syntax-aware identifier preprocessing pipeline based on language-specific abstract syntax tree parsing is applied to extract identifiers and perform lexical normalization and semantic cleaning. From the normalized identifiers, this study computes ten linguistically and cognitively grounded readability parameters. Semantic and contextual representations are attained using CodeBERT embeddings, which are then processed by a self-attention-based bidirectional long short-term memory model to learn sequential and contextual dependencies. Furthermore, the model is optimized using the AdamW optimizer, enabling improved convergence and overall performance. In the last stage, SHAP-based explainability is integrated for interpreting the contribution of identifier tokens and features to readability predictions. The IRAF-XADL method was experimentally examined on the benchmark Code Snippets: Insights and Readability Dataset, and the results prove the improved performance over the existing approaches in terms of diverse metrics.

Keywords-identifier readability; code snippets; CodeBERT embeddings; model optimization; Explainable Artificial Intelligence (XAI); deep learning

I. INTRODUCTION

Unreadable code can impair program comprehension and lead to the introduction of bugs. Code mainly contains natural language text, both in comments and identifiers, in a specific form [1]. However, the methods presented to approximate code readability account for only organizational aspects and visual characteristics of source code, such as alignment of characters and line length. The source code should be understandable and readable [2]. These two features are frequently considered equivalent, but they are unique textual features. Readability relates to the difficulties of the text itself, while understandability is associated with the reader's capability to understand the meaning of the text [3]. Therefore, code assessed as readable might still not be understandable to a user without the essential knowledge or experience. Code comprehension and reading are regular activities for software developers. Identifier names interconnect program perceptions to the reader and are essential in program understanding and software maintenance.

Identifier names also constitute artifacts of the intellectual steps of the programmer and might be seen as a representation of his understanding of the concepts involved in the source code [4]. There is enough motivation to have good-quality identifiers. Low-quality identifier names could be seen as a safety guard, but can have adverse outcomes, as modern studies on source code readability validated a link between less readable source code and software flaws [5]. Current studies on source code readability focus on the involvement of source

code modules and the method in which the semantic content of identifiers contributes to readability and understandability [6]. Modern computing power, the convenience of data, and novel algorithms have led to significant achievements in Artificial Intelligence (AI). In this context, an additional potential application area for AI/ML is automated code analysis and intelligent software study with multiple motives such as vulnerability recognition, code classification, code summarization, and clone identification.

To enhance the accuracy of identifier readability assessment in source code, this paper presents an Identifier Readability Analysis Framework using an Explainable Attention-Based Deep Learning (IRAF-XADL) approach. The proposed model intends to evaluate the quality of identifiers in Python and C++ programs, with the following key contributions:

- Introduces a comprehensive set of ten linguistically and cognitively grounded readability parameters, allowing multidimensional assessment of identifier quality from semantic, structural, contextual, and cognitive perspectives.
- Utilizes CodeBERT embedding for capturing semantic and contextual representations of identifiers that efficiently model programming-specific language characteristics beyond surface-level token attributes.
- Leverages a Self-Attention-based Bidirectional Long Short-Term Memory (SA-BiLSTM) model to learn sequential and contextual relations among identifier tokens for practical identifier readability assessment.
- Integrates the AdamW optimizer for hyperparameter tuning to ensure better convergence behavior and superior model performance.
- Incorporates SHAP-based explainability to provide interpretable insights into the contribution of identifier tokens toward readability predictions.
- Performs a comprehensive experimental evaluation on the benchmark datasets, demonstrating that the proposed framework performs better than baseline techniques under diverse evaluation measures.

II. LITERATURE REVIEW ON IDENTIFIER READABILITY ASSESSMENT

RoFTCodeSum [7] is a new fine-tuning technique that improves the robustness of code summarization against poorly readable code. In [8], an experimental analysis on LLM-based restructuring utilized GPT-4o, applied to 100 Python classes drawn from the ClassEval benchmark. In [9], a DL-based method was trained and assessed to operate entirely inside protected corporate environments to categorize the quality of the source code. In [10], an initial survey for ML-driven PLI techniques offered details into the domain's current status and supervision toward an advanced tool growth for PLI. In [11], the efficiency of search-driven producers was combined with the readability of LLM-provided examinations. An assessment of 9 industrial and open source LLMs showed that author readability enhancement alterations are complete, semantically-keeping, and constant over various iterations. MALSIGHT [12] is a new code summarization architecture that can repeatedly produce binary malware descriptions by discovering malicious source code and harmless pseudocode. In the development steps, MalT5, a new LLM-driven code technique, is tuned on the MalS and harmless pseudocode datasets. This process eases the comprehension of pseudocode architecture and models complex interactions among functions, therefore, enhancing summaries' accuracy, usability, and completeness.

III. PROPOSED IDENTIFIER READABILITY FRAMEWORK

This study introduces a novel method for assessing identifier readability in source code. Figure 1 presents the high-level workflow of the proposed identifier readability methodology.

A. Identifier Preprocessing and Normalization Pipeline

A syntax-aware identifier preprocessing module is utilized to enhance the quality of features extracted from code snippets and improve readability prediction. Identifier preprocessing uses lexical normalization methods such as split snakeCase and camelCase, lowercase normalization, and digit-letter separation for decomposing the identifiers with expressive semantic tokens and decreasing the vocabulary sparsity.

B. Feature Engineering

From the normalized identifiers, a set of ten linguistically and cognitively grounded readability parameters is computed for quantitatively modeling human-perceived identifier readability.

- Meaningful Clarity (MC) models how philologically meaningful the identifier tokens are.

$$MC = \frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(w_i \in V) \quad (1)$$

- Naming Conformance (NC) assesses observance of language-based naming conventions.

$$NC = \begin{cases} 1, & \text{if identifier follows naming conventions} \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

- Optimal Length (OL) estimates how near the identifier length is to an optimum distance.

$$OL = 1 - \frac{|len(name) - L_{opt}|}{L_{max}} \quad (3)$$

- Domain Relevance (DR) evaluates how closely identifier tokens associate with domain-based vocabulary.

$$DR = \frac{1}{n}\sum_{i=1}^{n}\cos\left(embedding(w_i), embedding(D)\right) \quad (4)$$

- Pronounceability (PR) estimates how simply an identifier can be defined depending on the ratio of vowels for the entire character set.

$$PR = \frac{\#vowels}{\#characters} \quad (5)$$

- Lexical Familiarity (LF) measures how acquainted identifier tokens are by averaging their utilization frequency in a large natural language corpus.

$$LF = \frac{1}{n} \log \sum \left(freq(w_i) \right) \quad (6)$$

- Context Consistency (CC) estimates the semantic constancy of an identifier along with closely related identifiers employing embedding-driven comparison.

$$CC = cosine\left(embedding(name), embedding(nearest_{k_{names}})\right) \quad (7)$$

- Scope Appropriateness (SA) measures whether an identifier's length is suitable for the scope size.

$$SA = \frac{OL}{\log(scope_{size}+1)} \quad (8)$$

- Cognitive Load Score (CLS) estimates the cognitive exertion needed for understanding an identifier by integrating familiarity, clarity, and ambiguity-relevant factors.

$$CLS = 0.4(1 - MC) + 0.3(1 - LF) + 0.3(AD) \quad (9)$$

- Predictability (PRED) estimates how probable an identifier is of producing its surrounding code context employing a stochastic language method.

$$PRED = P(name \mid surrounding_{code}) \quad (10)$$

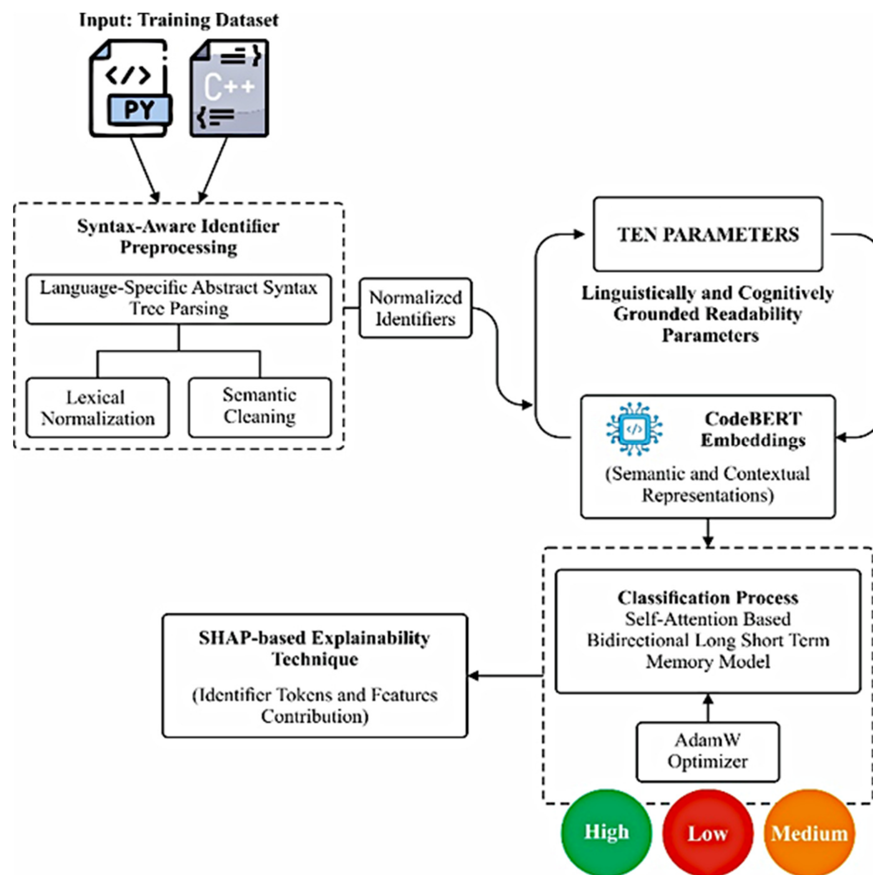

Fig. 1. Workflow of the proposed model.

C. *CodeBERT-Based Embeddings*

CodeBERT was used to extract the contextual and semantic information of identifiers and their surrounding code framework. CodeBERT [13] is a pre-trained algorithm that has been enhanced for handling programming-based contexts efficiently and trained on a vast corpus of programming languages. CodeBERT is intended for understanding the semantic source code and has been established as an effective technique in different software engineering applications. To acquire the embedding vector of the identifier token series, this method transforms the token order within three encoded vectors, such as (i) Segment Embeddings $[EA]$, (ii) Position embedding $[EP]$, and (iii) Token Embeddings $E[tk]$. Position embedding supports the encoder for embedding the token for its position. CodeBERT creates an embedding vector for a token that links to its position when the token appears twice.

D. *Attention-Driven DL Architecture for Identifier Readability*

The extracted representations are then passed into the SA-BiLSTM model for robust identifier readability classification.

BiLSTM [14] is an extension of traditional LSTMs. The BiLSTM layer encompasses dual separate LSTMs to analyze the data of an identifier series from dual directions. LSTM integrates a cell memory gate for monitoring values across alternating time intervals and 3 inherent gates, such as forget, input, and output gates, for handling the data flow. This technique estimates the output gate (o_t), forget gate (f_t), input gate (i_t), input modulation gate (g_t), cell memory state (c_t), and hidden state (h_t), for the present time-step employing identifier embedding (x_{t-1}) and hidden state vectors (h_{t-1}) from the previous time-step, and the engineered identifier attributes (r_t). The distinctive BiLSTM could not identify the features essential for determining the identifier readability, depending on related utilization. To acquire text features, which are further relevant to the contextual data, the Self-Attention Mechanism (SAM) is used along with the presented decoding algorithm. The SAM enables the given input for interacting with itself and extracts the significant attributes by allocating more weight to designate the importance of the feature. h_t acquired from the BiLSTM layer is given to the input for the SA layer. The SAM works as:

$$\alpha_t = \tanh(W_w * h_t + b_w),\quad \delta_t = \frac{\exp(\alpha_t * \alpha_w)}{\sum_t \exp(\alpha_t * \alpha_w)},\quad a = \sum_t \delta_t * h_t \tag{11}$$

where b_w and W_w represent the bias and weight, correspondingly, that is utilized for finding the new information of the hidden vector (α_t). The softmax layer in the decoding is employed for generating an identifier dependability forecast based on the consequential result from the SAM layer. At every time step t, it chooses the words with great possibility and refers them to the next time step for producing the whole text caption.

$$y_t = \text{softmax}\,(W_a * a + b_a) \tag{12}$$

where b_a, W_a, and y_t denote the bias and the weighted matrix, respectively.

E. Model Optimization Using AdamW

AdamW [15] is an optimized variant of the Adam optimizer that decouples weight decay from the gradient update, enhancing generalization in DL models. Contrasting the original Adam method that couples L_2 normalization to the gradient update, AdamW implements weight decay directly to the parameters. This clear decoupling separates the normalization impact from the dynamic update, resulting in further constant convergence and better simplification. Therefore, weight decay can be incorporated as an L2 normalization term with coefficient λ in the given amplified loss function:

$$\mathcal{L}_{total}(w) = \mathcal{L}_{data}(w) + \frac{\lambda}{2}\|w\|_2^2 \tag{13}$$

where $\mathcal{L}_{data}(w)$ is the original task loss. Taking the gradient of $\mathcal{L}_{total}(w)$ and implementing a learning-rate process submits the traditional shrinkage update,

$$w_{t+1} = (1 - \alpha\lambda)w_t - \alpha\nabla_w\mathcal{L}_{data}(w_t) \tag{14}$$

which shrinks every weight by a factor of $(1 - \alpha\lambda)$, before the data-based modification. In AdamW, the above decay term is clearly decoupled from the dynamic moment approximation, as illustrated in the last update. This decoupling assures that normalization does not interfere with adjustable learning dynamics, leading to further constant convergence and improved simplification.

F. Explainability and Feature Attribution via SHAP

SHAP-based explainability is utilized for interpreting the contribution of identifier tokens and features toward readability predictions. SHAP is a post-hoc explanatory tool, intended for assisting the clarification of the output produced by all DL models. SHAP depends on the impression of Shapley value, and it has a robust foundation in collaborative game theory. The estimation of Shapley values for every feature relies on a tentative expectation function that enables the interpretation of the feature's marginal involvement.

$$f(x^*) = \Phi_0 + \sum_{j=1}^{M} \Phi_j^* \tag{15}$$

Here, $f(x^*)$ signifies the ML forecasted value, Φ_0 represents the average estimation for the training datasets, and Φ_j^* denotes the Shapley value for feature j. SHAP enables both local and global explanations of DL models. Global explanations condense the effect of individual features over the whole dataset, while local explanations detect the impact of every attribute for a precise instance of examples. Moreover, XAI analysis enhances the comprehensibility of the DL models.

IV. PERFORMANCE EVALUATION AND EXPLAINABILITY INSIGHTS

A. Dataset Used

The performance of the proposed model was analyzed using the Code Snippets: Insights and Readability dataset [16], which is a collection of C++ and Python code snippets annotated with various code properties. The dataset is developed to support the examination of coding styles, complexity, and readability in Python and C++ programming.

- Python Data: This dataset offers an extensive collection of metrics and readability scores for Python code snippets. Every entry contains information, namely the Python solutions, problem title, number of lines, difficulty level, comments, code length, indents, cyclomatic complexity, loop count, identifiers, line length, and readability score.
- C++ Data: This dataset offers a comprehensive set of metrics and readability scores for C++ code snippets. Each record includes details such as the number of lines, the code itself, comments, code length, number of indents, cyclomatic complexity, line length, loop count, identifiers, and readability score.

TABLE I. DETAILS OF THE DATASET

Identifier readability level	Count	
	Python	CPP
Medium	560	500
High	560	502
Low	561	502
Total samples	**1681**	**1504**

B. Results on Python Data

Table II signifies the code identifier outcomes of the IRAF-XADL method on Python data at a 70:30 training/test split. In 70% training, the IRAF-XADL method achieves average $accur_y$ of 98.13%, $preci_n$ of 97.22%, $recal_l$ of 97.20%, $F1_{score}$ of 97.21% and AUC_{score} of 97.90%. At 30% testing, the IRAF-XADL method achieves average $accur_y$ of 97.36%, $preci_n$ of 96.14%, $recal_l$ of 96.01%, $F1_{score}$ of 96.03% and AUC_{score} of 97.02%.

TABLE II. CODE IDENTIFIER RESULT OF IRAF-XADL TECHNIQUE USING PYTHON DATA WITH 70:30 SPLIT

Classes	$Accur_y$	$Preci_n$	$Recal_l$	$F1_{score}$	AUC_{score}
Training Phase (70%)					
Medium	97.36	95.24	96.94	96.08	97.26
High	99.32	99.22	98.71	98.97	99.17
Low	97.70	97.19	95.96	96.57	97.27
Average	98.13	97.22	97.20	97.21	97.90
Testing Phase (30%)					
Medium	96.83	92.70	98.21	95.38	97.18
High	98.22	97.66	97.09	97.38	97.95
Low	97.03	98.08	92.73	95.33	95.92
Average	97.36	96.14	96.01	96.03	97.02

C. Results on CPP Data

Figure 2 shows the classifier results of the IRAF-XADL method on CPP data, demonstrating a confusion matrix with effective recognition of each class and the ROC curve, which signifies maximal efficiency on every class.

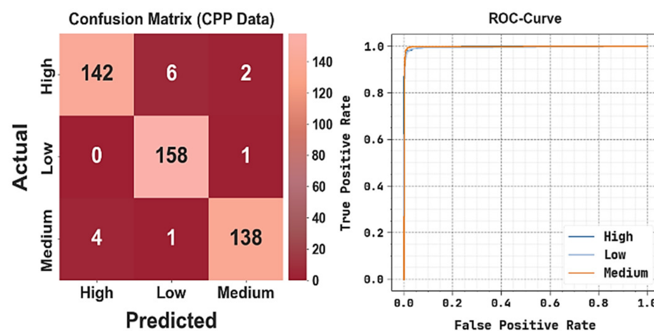

Fig. 2. Classifier outcome of IRAF-XADL method (a) Confusion matrices and (b) ROC curve using CPP data.

D. Comparative Discussions

Table III presents a brief comparison analysis with baseline models [17, 18]. On the Python dataset, the results show that the GNB-Isotonic, Perceptron, and LDA techniques demonstrated lower classification results, while the RF model obtained slightly closer results. The MLP, SMO, and LR models exhibited reasonable performance. However, IRAF-XADL establishes capable performance with $accur_y$ of 98.13%, $preci_n$ of 97.22%, $recal_l$ of 97.20%, and $F1_{score}$ of 97.21%, correspondingly. Similarly, on the CPP dataset, LDA and RF approaches demonstrated lower classification outcomes. Likewise, the RF model achieved the least performance with $accur_y$ of 65.00%. Additionally, the MLP, LR, GNB-Isotonic, Perceptron, and SMO methods displayed reasonable performance. The IRAF-XADL framework depicts promising outcomes with $accur_y$ of 98.42%, $preci_n$ of 97.62%, $recal_l$ of 97.61%, and $F1_{score}$ of 97.61%.

TABLE III. COMPARATIVE ANALYSIS OF IRAF-XADL WITH EXISTING MODELS ON THE PYTHON AND CPP DATASETS

Methods	Datasets	$Accur_y$	$Preci_n$	$Recal_l$	$F1_{score}$
MLP	Python	76.00	67.48	83.50	
	CPP	79.33	72.45	73.54	
SMO	Python	82.00	76.25	78.63	
	CPP	80.56	87.65	86.89	
LR	Python	81.00	87.02	90.58	
	CPP	75.23	87.55	87.05	
RF	Python	61.10	90.52	75.41	
	CPP	65.00	83.88	90.07	
GNB-Isotonic	Python	57.00	83.28	87.65	
	CPP	71.56	69.54	83.23	
Perceptron	Python	54.10	71.86	65.96	
	CPP	71.08	73.60	86.30	
LDA	Python	53.00	91.45	69.03	
	CPP	62.45	74.44	73.65	
IRAF-XADL	Python	98.13	97.22	97.20	97.21
	CPP	98.42	97.62	97.61	97.61

E. XAI Analysis

Figure 3 depicts the SHAP-based feature importance analysis for Python code classification at three levels: Low, Medium, and High.

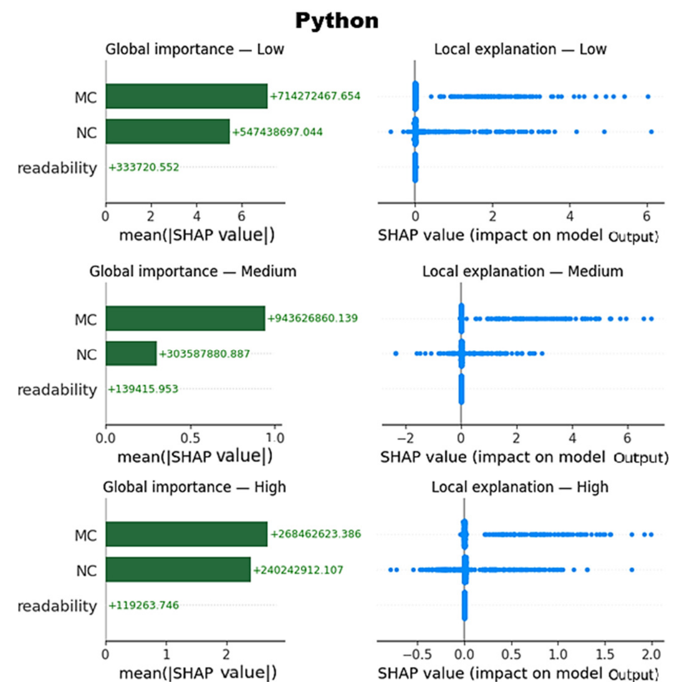

Fig. 3. SHAP values with Python data.

The left panels show global importance, signifying that MC and NC have the maximum influence on the model, while readability contributes minimally. The right panels display local explanations, in which every dot signifies the SHAP value for an individual prediction, which shows how features impact

the output. MC reliably pushes predictions positively across all levels, NC has a reasonable positive effect, and readability has a negligible influence. Figure 4 provides a SHAP-based feature importance analysis for C++ code classification at Low, Medium, and High levels. The left panels demonstrate global importance, representing that NC and MC are the dominant features, while readability has a lesser impact at every level. The right panels show local explanations, where each dot signifies the SHAP value for an individual prediction, displaying how attributes influence the model output.

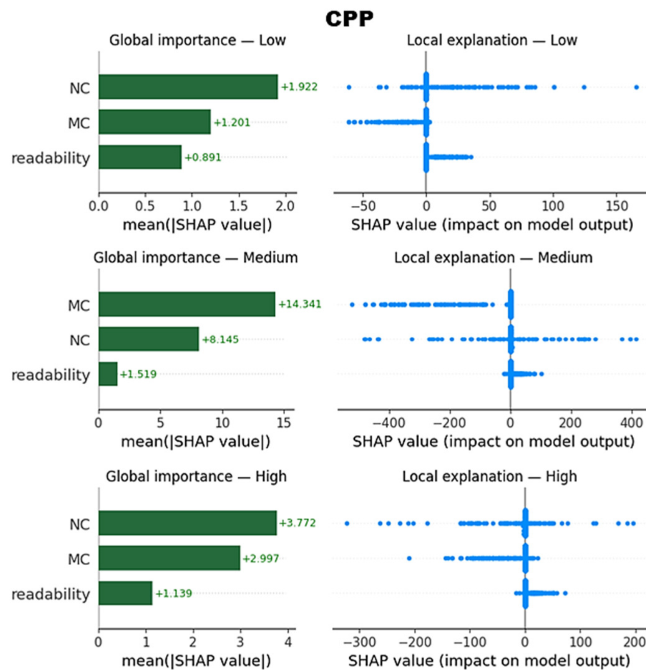

Fig. 4. SHAP values with CPP data.

V. CONCLUSION

This study presents the IRAF-XADL model for assessing the quality of identifier names in Python and C++ source code. This approach employs a syntax-aware identifier preprocessing module depending on language-specific AST parsing for identifier extraction, lexical normalization, and semantic cleaning. In addition, semantic and contextual representations are obtained through CodeBERT embeddings, which are processed by an SA-BiLSTM network for learning sequential and contextual dependencies. The model was optimized with the AdamW optimizer, allowing better convergence and enhanced overall performance. Finally, SHAP-based explainability is incorporated to interpret the contribution of identifier tokens and attributes to readability predictions. The comparative results highlight the supremacy of the proposed model compared to other techniques, with an accuracy of 98.13% and 98.42% on Python and CPP data. The proposed model can capture semantic and structural characteristics of identifiers, enabling automatic assessment of software quality. However, since the model was validated on a limited dataset of code snippets, it may not fully represent identifiers from different programming languages and large-scale repositories. Future work will incorporate larger datasets and integrate contextual code embeddings to further enhance identifier readability prediction and generalization performance.

DECLARATION ON COMPETING INTERESTS

The authors declare that they have no known relations or financial interests that could have influenced the results of this study.

ACKNOWLEDGMENT

Not applicable in this study.

DATA AVAILABILITY

The dataset used in this study is publicly available at [16].

REFERENCES

[1] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *Proceedings of the 26th Conference on Program Comprehension*, Feb. 2018, pp. 286–296, https://doi.org/10.1145/3196321.3196347.

[2] D. Oliveira, R. Santos, B. de Oliveira, M. Monperrus, F. Castor, and F. Madeiral, "Understanding Code Understandability Improvements in Code Reviews," *IEEE Transactions on Software Engineering*, vol. 51, no. 1, pp. 14–37, Jan. 2025, https://doi.org/10.1109/TSE.2024.3453783.

[3] H. Mestiri, I. Barraj, and M. Machhout, "AES High-Level SystemC Modeling using Aspect Oriented Programming Approach," *Engineering, Technology & Applied Science Research*, vol. 11, no. 1, pp. 6719–6723, Feb. 2021, https://doi.org/10.48084/etasr.3971.

[4] S. Butler, M. Wermelinger, Yijun Yu, and H. Sharp, "Exploring the Influence of Identifier Names on Code Quality: An Empirical Study," in *2010 14th European Conference on Software Maintenance and Reengineering*, Mar. 2010, pp. 156–165, https://doi.org/10.1109/CSMR.2010.27.

[5] E. D. Berger, C. Hollenbeck, P. Maj, O. Vitek, and J. Vitek, "On the Impact of Programming Languages on Code Quality: A Reproduction Study," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 41, no. 4, July 2019, Art. no. 21, https://doi.org/10.1145/3340571.

[6] R. A. Al-Msie'deen, "Tag Clouds for Object-Oriented Source Code Visualization," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4243–4248, June 2019, https://doi.org/10.48084/etasr.2706.

[7] W. Zeng, Y. Chai, H. Zhou, F. Meng, J. Zhou, and X. Gu, "Readability-Robust Code Summarization via Meta Curriculum Learning." arXiv, 2026, https://doi.org/10.48550/ARXIV.2601.05485.

[8] A. Midolo, E. Tramontana, and M. Di Penta, "From Human to Machine Refactoring: Assessing GPT-4's Impact on Python Class Quality and Readability." arXiv, 2026, https://doi.org/10.48550/ARXIV.2601.13139.

[9] S. Tokumoto, S. Kusumoto, and R. Imai, "Development and Evaluation of a Deep Learning-Based Model for Source Code Quality Classification Using Industrial Data," *Journal of Software Engineering Practice*, vol. 6, no. 1, pp. 1–19, June 2025.

[10] A. Verma, R. Saha, G. Kumar, A. Brighente, M. Conti, and T. H. Kim, "Exploring the Landscape of Programming Language Identification With Machine Learning Approaches," *IEEE Access*, vol. 13, pp. 23556–23579, 2025, https://doi.org/10.1109/ACCESS.2025.3538108.

[11] M. Biagiola, G. Ghislotti, and P. Tonella, "Improving the Readability of Automatically Generated Tests Using Large Language Models," in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*, Mar. 2025, pp. 162–173, https://doi.org/10.1109/ICST62969.2025.10989020.

[12] H. Lu *et al.*, "Malsight: Exploring Malicious Source Code and Benign Pseudocode for Iterative Binary Malware Summarization," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 6733–6747, 2025, https://doi.org/10.1109/TIFS.2025.3583552.

[13] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Aug. 2020, pp. 1536–1547, https://doi.org/10.18653/v1/2020.findings-emnlp.139.

[14] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, July 2005, https://doi.org/10.1016/j.neunet.2005.06.042.

[15] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization." arXiv, 2017, https://doi.org/10.48550/ARXIV.1711.05101.

[16] P. Maheshwari, "Code Snippets: Insights and Readability." Kaggle, [Online]. Available: https://www.kaggle.com/datasets/paakhim10/code-snippets-insights-and-readability.

[17] Q. Mi, J. Keung, Y. Xiao, S. Mensah, and X. Mei, "An Inception Architecture-Based Model for Improving Code Readability Classification," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, Mar. 2018, pp. 139–144, https://doi.org/10.1145/3210459.3210473.

[18] B. Susanto, R. Ferdiana, and T. B. Adji, "Predicting Multiclass Java Code Readability: A Comparative Study of Machine Learning Algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 16, no. 4, 2025, https://doi.org/10.14569/IJACSA.2025.01604102.