An Optimized Hybrid Approach for Code Smell Detection Using Machine Learning and Attention-Based FNN

Taniz Fatema[1], Nazneen Akhter[2] and Md Musfique Anwar[3]
[1]Department of ICT Bangladesh University of Professionals, Dhaka, Bangladesh
[2]Department of ICT Bangladesh University of Professionals, Dhaka, Bangladesh
[3]Department of CSE Jahangirnagar University Dhaka, Bangladesh
Correspondence *e*-mail: taniz.fatema17@gmail.com, nazneen.akhter@bup.edu.bd, manwar@juniv.edu

***Abstract*—Code smells are structural flaws in the source code that have a detrimental effect on readability, maintainability, and dependability. To address these challenges we proposed a hybrid model using ML classifiers with attention-based Feed-forward Neural Network. In this work, we applied five Machine Learning classifiers are SVM (Support Vector Machine), KNN (K-Nearest Neighbors), LR (Logistic Regression), DT (Decision Tree), RF (Random Forest), two feature selection techniques are RFECV(Recursive Feature Elimination with Cross-Validation) and Information gain on different datasets namely DC(Data Class), FE(Feature Envy), GC(God Class), LM(Long Method). To enhance the performance and reduce the overfitting-underfitting issues we applied two optimization techniques are Bayesian Optimization and Grid Search while applying Information Gain feature selection technique. For deeply analyzing trained the model with attention-based FNN (Feed Forward Neural Network). Experimental outcome shown that our proposed model obtained the highest accuracy on Long method dataset, 99.69% by using SVM, RF classifiers and Information gain with Bayesian optimization technique. Also achieved the accuracy, 99.68% for RF, DT, LR classifiers with RFECV feature selection technique on the same dataset.**

***Index Terms*—Code smell, Attention mechanism, Machine-learning, Deep-learning, Feature selection technique, Optimization technique.**

I. Introduction

Software maintenance is one of the most expensive stages of software development. Bad design decisions and structural problems that are ingrained in the source code make maintenance much harder to perform and build up technical debt. Code smells, introduced by Fowler, are symptoms deriving from underlying design problems and they do not necessarily lead to failures but produce mediocre quality software that is hardly maintainable. Smell detectors based on machine learning (ML) frequently have large feature dimensionality and poor generalization since they rely on source-code metrics. Although explainability is a problem, deep learning (DL) models are able to grasp non-linear patterns. Thus, hybrid models that combine the representational capability of DL architectures with the advantages of deterministic ML classifiers are required. Prior studies have predominantly relied on tree-based machine learning classifiers, doesn't incorporate attention mechanisms which are well-suited for modeling dependencies and highlighting the most relevant features and focused primarily on predictive performance metrics such as accuracy, precision, and recall, while overlooking computational aspects such as timing efficiency. To overcome these challenges this, work proposed a hybrid model. The main contribution of this paper is -

- Developed a hybrid model based on machine learning and deep learning model
- Investigated the role of attention mechanisms in capturing feature dependencies in order to enhance predictive performance for code smell detection.
- Evaluated computational efficiency alongside predictive performance by analyzing time of each method, thereby assessing their suitability for real-world adoption in software engineering workflows.

This work provided three research questions to demonstrate this contribution, which will be assessed in the results section. This paper is assembled as follows: Section II provides a literature review. Section III illustrates a smarter hybrid model. Section IV evaluates the results and performances. Finally, allusions to upcoming works are included in the works conclusion.

II. Literature Review

The literature presents diverse approaches aimed and several methods that researchers have investigated at detecting code smells during software development. This study provides a brief overview of fifteen important works from 2021 to 2024. In the year of 2025, Anh et al. implemented two types of ensemble features are semantic features obtained from pre-trained programming language models and structural features. Relying on the type of code smell, the findings showed that the suggested model achieves cutting-edge results on MLCQ dataset with gained ranging from 5.98% to 28.26% [1]. In the year of 2024, Sharma et al. present a comprehensive survey of ML techniques conducted to source code analysis, highlighting how ML has transformed traditional software engineering tasks. The most popular classical machine learning approaches are SVM, RF, and DT [2]. Pervin et al. presents a number of algorithms based on decision trees with Binary Relevance, Label Powerset, and multi-label classification problem is addressed by Classifier Chain Methods. Proposed model achieved the accuracy of 99.54% [3]. In the year 2024, the authors proposed Federated Learning to detect the smell during development. Fl employed to collaboratively train machine learning algorithms. This study achieved the highest accuracy, 99% on Pecorelli et al. or Khalid et al. datasets [4]. Nasraldeen et al. applied KNN, SVM, DT, XGboost, and

MLP that integrates with random oversampling technique. XGboost obtained the highest accuracy, 100% on data class [5]. Pravin et al. analyzed that Ensemble Learners, Neural networks, Bayesian Learners, SVM, DT, Rule-Based Learning and Miscellaneous performed better to detect code smells [6]. Hamund et al. implemented greedy search and backward elimination in python and java which are simpler than full stacking ensemble to detect the code smells. According to this study, dynamic stacking ensembles which are less complex than complete stacking ensembles were able to enable the effective and consistent detection performance of Python and Java code smells over all base models [7]. The authors employed deep learning methods namely BLSTM and GRU. Two data balancing methods applied namely Tomek links, Random oversampling for GC, FE, DC, LM datasets. Both models achieved the highest accuracy,100% for long method dataset [8]. In the year of 2023, Puja et al. suggested a hybrid model that utilized two feature selection techniques are the wrapper approach and mutual information and used SVM and Grid search to identify code smells. This model achieved the accuracy, 99% for Azureus BLOB dataset [9]. Shivani et al. applied MI, fisher score, and univariate ROC–AUC feature selection technique with brute force and RF correlation strategies. Experimented on DC, LM, FE and GC code smells and J48, JRip, RF, NB, SMO, SVM, and kNN classifiers [10]. Seema et al. used the six Ml algorithm, grid search optimization, Chi-squared, and wrapper-based feature selection strategies to enhance the accuracy [11]. The authors implemented ensemble learning algorithms namely Adaboost, Gradient boosting, Bagging, XGboosting and Max voting and DL algorithms namely ANN and CNN on four datasets. The class was balanced using SMOTE. [12]. Nazneen et al. implemented four ML classifiers namely SVM, DT, NB, RF. In this proposed model SVM with SMOTE performed better [13]. Inderpreet et al. employed Bagging and Random Forest. Correlation based feature selection using BFS, Info Gain to identify code smells and measures accuracy, G-mean1, G-mean2, and F-measure. The best results are obtained using RF and Correlation-based Feature Selection [14]. Maiga et al. presented SVMDetect to identify code smell using SVM and this model performed better than DETEX [15].

III. SMARTER HYBRID MODEL

Code scent detection requires a combination of automated techniques, established design principles, teamwork, and in-tuition. Code smells are not errors; rather, they are signs of more serious design issues that might make code difficult to comprehend, update, or grow over time. To develop hybrid machine learning–deep learning models, investigate the role of attention mechanisms in capturing feature dependencies and enhancing predictive performance by applying attention layers within the hybrid framework and evaluate the computational efficiency alongside predictive performance by analyzing experimental time assessing their suitability for real-world adoption in software engineering workflows we proposed a smarter hybrid model. The proposed model is developed in three stages. Firstly, experiments are conducted using the complete feature set. Secondly, two feature selection techniques namely RFECV and Info Gain are applied to identify the most pertinent attributes. A 10-fold cross-validation strategy is employed to split the data into training, validation, and testing sets. Subsequently, five ML classifiers namely SVM, KNN, DT, LR, RF are trained and evaluated. To address overfitting and underfitting issues, Bayesian Optimization and Grid Search are applied, particularly in conjunction with the Information Gain based feature selection. In the final stage, the optimized classifier outputs are forwarded to a Feedforward Neural Network (FNN) enhanced with a self-attention mechanism. The proposed framework produces predictions evaluated using training, validation, and testing accuracy, classification reports, and computational time efficiency.

A. Illustration of Dataset and Code Smells

Datasets derived and analyzed from Fontana et al. [16]. Author calculated a wide range of object-oriented metrics after choosing 74 systems from 111 different dimensions, determined 61 class-level metrics for DC and GC code smells and 82 metrics for FE and LM for code smells at the method level. To identify code smells, they used a range of techniques and tools. Data Class hold data without offering sophisticated functionality and that other classes heavily rely on. God Class frequently has a lot of code, is complicated, uses a lot of data from other classes, and implements a variety of features. Feature Envy frequently uses a large number of qualities from a few other classes, as well as additional characteristics from other classes than from its own class. Long method typically uses a lot of data from other classes, have a lot of code, are complicated, and are hard to grasp [16].

TABLE I: Samples and Matrics of Code Smells

Code Smells	Samples	Selected Metrics
Data Class	420	61
God Class	420	61
Feature Envy	420	82
Long Method	420	82

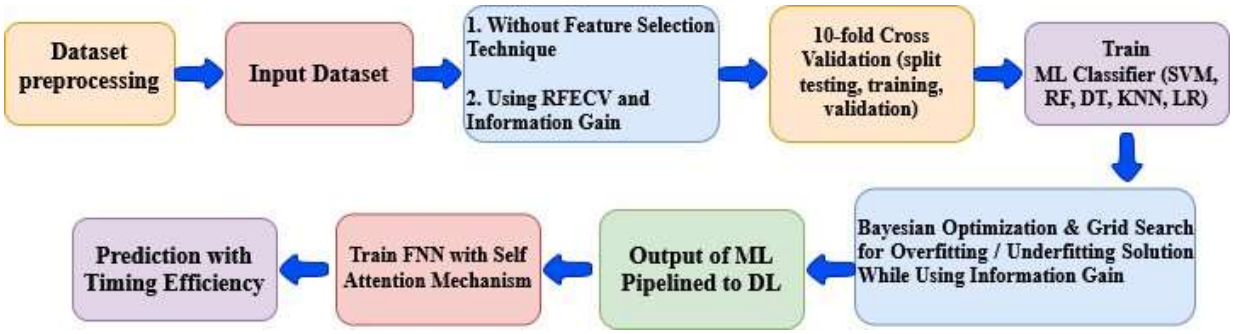

Fig. 1. Proposed Smarter Hybrid Model for Code Smell Detection

B. Method of Feature Selection

RFECV is a reliable and automatic feature selection technique that uses cross-validation and recursive feature elimination to determine the ideal number of features for a machine learning model [17]. Information Gain /Mutual Information is used to quantify the amount of knowledge that one variable offers about another. By lowering prediction uncertainty, they enhance model accuracy, divide decision boundaries, and optimize feature selection [18].Formula of Information Gain:

$$IG(S,A) = \sum_v \frac{|S_v|}{|S|} H(S_v) \quad (1)$$

C. ML Classifiers and FNN

SVM is used to determine the optimal line or decision boundary for dividing n-dimensional space so that the following data points may be recognized with ease. DT is a supervised learning method that may be applied to both regression and classification tasks, while it is most commonly used for classification. RF supervised ensemble learning that integrates the results of several decision trees to generate a final prediction that is more precise and reliable. KNN classifies a new data point according to the majority class, average value, and closeness of its ”K” nearest neighbors in the current dataset. LR is designed for binary classification problems. It uses a linear combination of input characteristics and the logistic (sigmoid) function to estimate the likelihood that a data instance belongs to a specific class. A Feedforward Neural Network (FNN) is a kind of artificial neural network in which there is only one direction of data flow from the input layer to the output layer via one or more hidden layers and no cycles or feedback connections are made. Each layer is made up of neurons that apply a nonlinear activation function after applying a weighted linear combination of inputs. Because of its ease of use, good approximation ability, and computational economy, FNNs are frequently employed for classification and regression problems. The FNN architecture consists of an output layer with one neuron after two dense hidden layers, the first of which has 64 neurons and the second has 32. The Adam optimizer was used to train the model for 30 epochs with a learning rate of 0.001. To guarantee reproducibility, a random seed of 42 was employed. Python was used to implement each experiment.

D. 10-Fold Cross Validation

10-fold Stratified Cross-Validation strategy was employed, where the dataset was divided into ten equal folds while preserving class distribution. In each iteration, nine folds were used for training and one fold for validation. This process was repeated ten times, ensuring that every sample served as validation data exactly once, and the results were averaged to obtain robust performance estimates.

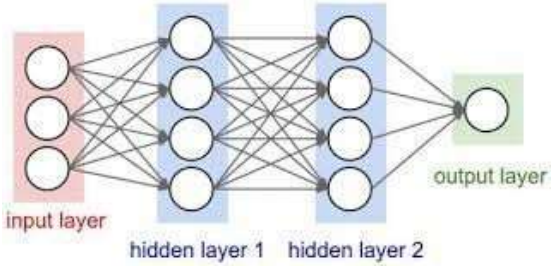

Fig. 2. Feed-forward Neural Network

E. Optimization Technique

Bayesian optimization is a potent optimization method that effectively determines the minimum (or maximum) of an objective function by utilizing the concepts of Bayesian inference. When handling costly, loud, or black-box functions, it works well. Grid search is an optimization technique that involves experimenting with several variables before selecting the one that yields the highest score. It optimizes the hyperparameters by first defining a list of values for each hyperparameter that has to be optimized, and then training a model for every possible combination of these numbers.

F. Attention Mechanism

Attention mechanism enables models to flexibly focus on the most pertinent portions of the input data by giving weights to various input pieces. This improves performance by generating context-aware representations and overcomes the shortcomings of previous models, such as FNN ignoring early information. To calculate attention scores, the input is subjected to a tanh activation after a linear transformation (xW+b). Normalized attention weights are produced by passing the scores through a softmax function. The significance of each input element is reflected in these attention weights. The input probabilities are multiplied by

the weights element by element. A weighted context vector (c) is created by adding the weighted values. The final classification procedure then makes use of the context vector, which highlights the most crucial data.

IV. Evaluation of results and performances

The hybrid model implemented on four datasets are Long Method, Data Class, God Class and Feature Envy. To complete the full experiment used processor Intel core i3, windows 11 version, RAM 4GB and operating system 64-bit. Performance evaluated by obtaining training accuracy, validation, and testing accuracy, classification reports, and computational time efficiency.

A. Dataset Preprocessing

The raw and processed versions of the experimental dataset are taken from the Fontana et al. code smell repository. The dataset goes through a number of preparation stages before the model is trained. To enable supervised learning, the class labels are first converted into numerical values using label encoding. The best discriminative software metrics are then found by performing feature selection using two complementary methods: RFECV and Info Gain (mutual information).

B. Research Question

The suggested model is assessed using the three research questions, RQ1, RQ2 and RQ3. Describing the performance of these questions below:

A. RQ1: How can hybrid machine learning–deep learning models improve anti-pattern detection compared to traditional approaches?

ML / Tree-based classifiers are simple but cannot always capture complex relationships in data. By combining machine learning with an attention-based neural network (deep learning) create hybrid systems that are better at recognizing non linear patterns in software metrics. This makes detection more accurate and flexible than using tree-based models / ML alone.

B. RQ2: How does the attention mechanism improve the ability of the detection model to capture feature dependencies and enhance predictive performance in code smell detection?

Attention mechanism incorporated with Feed- Forward Neural Network to enable dynamic weighting of static code metrics. This improves representation learning and code smell detection performance by enabling the model to prioritize informative features while suppressing less relevant ones.

TABLE II: Accuracy of ML-DL Final Output for Long Method Dataset

ML-DL	All Features	RFECV	INF.G + B.O	INF.G + G.S
SVM+FNN	97%	99% (Selected Feature 59)	**99.69**% (Selected Feature 20)	99.65% (Selected Feature 20)
RF + FNN	**99%**	**99.68%** (S.F 9)	**99.69%** (S.F 20)	98% (S.F 20)
DT + FNN	98%	**99.68%** (S.F 3)	99% (S.F 20)	97.65%(S.F 20)
KNN+FNN	90%	92% (S.F 59)	90% (S.F 20)	88%(S.F 20)
LR + FNN	99%	**99.68%** (S.F 10)	99% (S.F 20)	98.66% (S.F 20)

TABLE III: Accuracy of ML-DL Final Output for Feature Envy Dataset

ML-DL	All Features	RFECV	INF.G + B.O	INF.G + G.S
SVM+FNN	92%	97% (Selected Feature 5)	**99.65%** (Selected Feature 20)	99.58% Se-lected Fea-ture 20)
RF + FNN	**96%**	**98%** (S.F 6)	95% (S.F 20)	96% (S.F 20)
DT + FNN	95%	96% (S.F 3)	97% (S.F 20)	96% (S.F 20)
KNN+FNN	86%	95% (S.F 5)	79% (S.F 20)	83% (S.F 20)
LR + FNN	94%	**98%** (S.F 9)	96% (S.F 20)	92% (S.F 20)

TABLE IV: Accuracy of ML-DL Final Output for God Class Dataset

ML-DL	All Features	RFECV	INF.G + B.O	INF.G+ G.S
SVM+FNN	96%	98% (Selected Feature 13)	**99.63%** (Selected Feature 20)	99.58% (Selected Feature 20)
RF + FNN	**98%**	98% (S.F 4)	98% (S.F 20)	98% (S.F 20)
DT + FNN	97%	97% (S.F 40)	97% (S.F 20)	97% (S.F 20)
KNN+FNN	95%	97% (S.F 13)	98% (S.F 20)	98% (S.F 20)
LR + FNN	97%	**99%** (S.F 9)	97% (S.F 20)	98% (S.F 20)

TABLE V: Accuracy of ML-DL Final Output for Data Class Dataset

ML-DL	All Features	RFECV	INF.G + B.O	INF.G + G.S
SVM+FNN	97%	96% (Selected Feature 16)	**99.64%** (Selected Feature 20)	99.61 % (Selected Feature 20)
RF + FNN	**99%**	97% (S.F 12)	97% (S.F 20)	97% (S.F 20)
DT + FNN	98%	**99.64%** (S.F 3)	94% (S.F 20)	95% (S.F 20)
KNN+FNN	94%	91% (S.F 16)	97% (S.F 20)	96% (S.F 20)
LR + FNN	95%	95% (S.F 4)	96% (S.F 20)	96% (S.F 20)

Attention acts like a spotlight. It highlights the most important

TABLE VI: Classification Report (Precision, Recall, F1-Score, Support) of Feature Envy Dataset

ML-DL	All Features	RFECV	INF.G+B.O	INF.G+ G.S
SVM+FNN	P-88%, R-86%, F1-87%, S-141	P-99.65%, R-89%, F1-94%, S-28	**P-99.69%, R-99.68%, F1-99.63%, S-28**	P-99.69%, R-99.68%, F1-99.63%, S-28
RF + FNN	**P-94%, R-93%, F1-93%, S-141**	P-93%, R-99.62%, F1-97%, S-14	P-93%, R-89%, F1-91%, S-28	P-93%, R-93%, F1-93%, S-28
DT + FNN	P-90%, R-93%, F1-91%, S-141	P-93%, R-93%, F1-93%, S-28	P-92%, R-86%, F1-89%, S-28	P-93%, R-93%, F1-93%, S-28
KNN+FNN	P-85%, R-70%, F1-77%, S-141	P-93%, R-89%, F1-91%, S-28	P-64%, R-82%, F1-72%, S-28	P-69%, R-86%, F1-76%, S-28
LR + FNN	P-90%, R-89%, F1-90%, S-141	**P-99.66%, R-93%, F1-96%, S-28**	P-96%, R-79%, F1-86%, S-28	P-96%, R-79%, F1-86%, S-28

TABLE VII: Classification Report (Precision, Recall, F1-Score, Support) of God Class Dataset

ML-DL	All Features	RFECV	INF.G+B.O	INF.G + G.S
SVM+FNN	P-95%, R-93%, F1-94%, S-140	P-96%, R-96%, F1-96%, S-28	P-99.67%, R-99.65%, F1-99.63%, S-28	P-99.65%, R-99.63%, F1-99.58%, S-28
RF + FNN	P-96%, R-97%, F1-97%, S-140	P-96%, R-96%, F1-96%, S-28	P-96%, R-96%, F1-96%, S-28	P-96%, R-96%, F1-96%, S-28
DT + FNN	P-96%, R-93%, F1-94%, S-140	P-93%, R-96%, F1-95%, S-28	P-96%, R-93%, F1-95%, S-28	P-93%, R-96%, F1-95%, S-28
KNN+FNN	P-97%, R-86%, F1-91%, S-140	P-96%, R-93%, F1-95%, S-28	P-96%, R-96%, F1-96%, S-28	P-96%, R-96%, F1-96%, S-28
LR + FNN	P-95%, R-94%, F1-95%, S-140	P-99.69%, R-96%, F1-98%, S-28	P-96%, R-93%, F1-95%, S-28	P-99.69%, R-93%, F1-96%, S-28

features and reduces the influence of less useful ones. In this work, attention helps the neural network focus on the most important signals from the machine learning classifier. It works on the probability outputs of an optimized ML, capturing more complex feature relationships. In both cases, attention improves prediction by letting the model "pay more attention" to what matters most. Working steps of the attention mechanism:

1. **Calculate Attention Scores:** Score=tanh (input*w+b)

Softmax of Scores (Attention Weights)

2. **Softmax of Scores (Attention Weights)**

$$Softmax(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}} \quad (2)$$

3. **Weighted Inputs:** Multiply each probability by its attention weight

TABLE VIII: Classification Report (Precision, Recall, F1-Score, Support) of Long Method Dataset

ML-DL	All Features	RFECV	INF.G+B.O	INF.G + G.S
SVM+FNN	P-92%, R-96%, F1-94%, S-140	P-97%, R-99.69%, F1-98%, S-28	P-99.67%, R-99.65%, F1-99.67%, S-84	P-99.65%, R-99.63%, F1-99.58%, S-84
RF + FNN	P-99%, R-99%, F1-99%, S-140	P-99.63%, R-99.65%, F1-99.58%, S-14	P-99.57%, R-99.59%, F1-99.67%, S-28	P-99.69%, R-99.63%, F1-99.58%, S-28
DT + FNN	P-99%, R-98%, F1-98%, S-140	P-99.67%, R-99.65%, F1-99.57%, S-28	P-97%, R-99.63%, F1-98%, S-28	P-96%, R-96%, F1-96%, S-28
KNN+FNN	P-91%, R-76%, F1-83%, S-140	P-84%, R-93%, F1-88%, S-28	P-79%, R-93%, F1-85%, S-28	P-80%, R-86%, F1-83%, S-28
LR + FNN	P-98%, R-99%, F1-99%, S-140	P-99.69%, R-99.65%, F1-99.67%, S-28	P-97%, R-99.65%, F1-98%, S-28	P-97%, R-99.58%, F1-98%, S-28

TABLE IX: Classification Report (Precision, Recall, F1-Score, Support) of Data Class Dataset

ML-DL	All Features	RFECV	INF.G+B.O	INF.G + G.S
SVM+FNN	P-96%, R-94%, F1-95%, S-140	P-93%, R-96%, F1-95%, S-28	P-93%, R-99.65%, F1-97%, S-28	P-93%, R-99.69%, F1-97%, S-28
RF + FNN	P-99.57%, R-97%, F1-99%, S-140	P-99.69%, R-93%, F1-96%, S-14	P-93%, R-99.59%, F1-97%, S-28	P-93%, R-99.63%, F1-97%, S-28
DT + FNN	P-97%, R-99%, F1-98%, S-140	P-99.67%, R-99.65%, F1-99.57%, S-28	P-87%, R-96%, F1-92%, S-28	P-90%, R-96%, F1-93%, S-28
KNN+FNN	P-90%, R-93%, F1-92%, S-140	P-84%, R-93%, F1-88%, S-28	P-96%, R-96%, F1-96%, S-28	P-93%, R-96%, F1-95%, S-28
LR + FNN	P-93%, R-94%, F1-94%, S-140	P-90%, R-96%, F1-96%, S-84	P-93%, R-96%, F1-95%, S-28	P-93%, R-96%, F1-95%, S-28

4. **Compute Context Vector:** Sum of weighted inputs

RQ3. How does computational efficiency analysis contribute to real-world adoption?

A model is not useful in real life if it is too slow. Developers need quick feedback when writing or reviewing code. This hybrid model reports timing for every step (scaling, feature selection, training, testing), showing it can run efficiently and also measures time while using more advanced optimization (Bayesian tuning). This proves that the models are not just accurate, but also fast enough to be practical in real-world tools like IDEs or CI/CD systems. Table X and fig: 4 shows the computational efficiency of proposed model. To prove a model accurate, efficient, scalable, practical, its need to know timing efficiency. From fig 4, time complexity was not observed here, even after the heavy weighted model was developed.

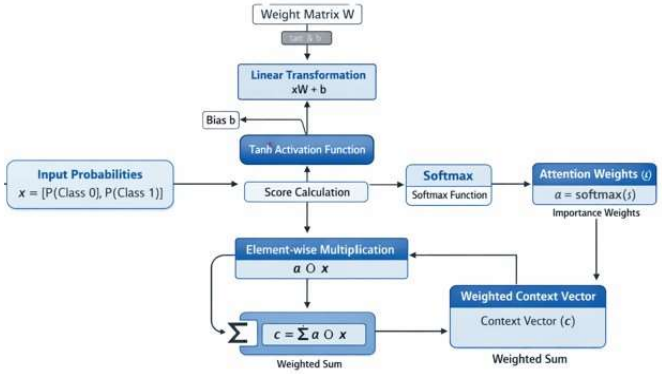

Fig. 3. Working steps of Attention Mechanism

TIMING SUMMARY	
Total Pipeline Time	: 16.17s
Scaling Time	: 0.00s
RFECV Feature Selection	: 8.47s
10-Fold CV Time	: 0.23s
Data Split Time	: 0.02s
SVM Training Time	: 0.04s
FNN Training Time	: 6.39s
Prediction Time	: 0.96s

Fig. 4. Timing Summary of Data Class by using RFECV

TABLE X: Experimental Training Time of This Model for Feature Envy Dataset.

Feature Envy	All Features (train time)	RFECV (train time)	Information gain with B.O (train time)
SVM	7.96s	6.49s	5.08s
RF	6.39s	7.3s	8.58s
DT	3.47s	4.87s	31.7s
KNN	4.63s	5.43s	5.73s
LR	5.4s	5.81s	5.9s

TABLE XI: Comparison between Proposed Model and Previous Existing Model

Authors	Dataset	Method	Accuracy (%)
Seema et al.	FE	Max Voting	91.45%
Proposed Model	FE	SVM+FNN	**99.65%**

V. Conclusion

This paper presented a smarter hybrid model by using five machine learning classifiers are namely Support Vector Machine (SVM), Logistic Regression (LR), Decision Tree (DT), K-Nearest Neighbor (KNN), and Random Forest (RF), two feature selection techniques are namely Recursive Feature Elimination with Cross-Validation and Information gain on Fontana et al. datasets are Data Class, God Class, Feature Envy, Long Method. Applied two optimization techniques: Bayesian Optimization and Grid Search. Self Attention Mechanism applied in Feedforward Neural Network. This model performs better on Long Method dataset by achieving highest accuracy, 99.69% by using SVM, RF classifiers and Information gain with Bayesian optimization technique. This framework offers a promising direction for automated software quality assessment. Future research will focus on incorporating source code embedding, multi-label detection, and automated refactoring recommendations.

References

[1] Anh Ho, Anh MT Bui, Phuong T Nguyen, Amleto Di Salle, and Bach Le. Ensesmells: Deep ensemble and programming language models for automated code smells detection. *Journal of Systems and Software*, 224:112375, 2025.

[2] Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari, Indira Vats, Hadi Moazen, and Federica Sarro. A survey on machine learning techniques applied to source code. *Journal of Systems and Software*, 209:111934, 2024.

[3] Pravin Singh Yadav, Rajwant Singh Rao, and Alok Mishra. An evaluation of multi-label classification approaches for method-level code smells detection. *IEEE Access*, 12:53664–53676, 2024.

[4] Sadi Alawadi, Khalid Alkharabsheh, Fahed Alkhabbas, Victor R Kebande, Feras M Awaysheh, Fabio Palomba, and Mohammed Awad. Fedcsd: A federated learning based approach for code-smell detection. *IEEE Access*, 12:44888–44904, 2024.

[5] Nasraldeen Alnor Adam Khleel and Ka´roly Nehe´z. Detection of code smells using machine learning techniques combined with data-balancing methods. *International Journal of Advances in Intelligent Informatics*, 9(3):402–417, 2023.

[6] Pravin Singh Yadav, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. Machine learning-based methods for code smell detection: a survey. *Applied Sciences*, 14(14):6149, 2024.

[7] Hamoud Aljamaan. Dynamic stacking ensemble for cross-language code smell detection. *PeerJ Computer Science*, 10:e2254, 2024.

[8] Nasraldeen Alnor Adam Khleel and Ka´roly Nehe´z. Improving accuracy of code smells detection using machine learning with data balancing techniques. *The Journal of Supercomputing*, 80(14):21048–21093, 2024.

[9] Ritu Sarker Puja, Taniz Fatema, Nazneen Akhter, and Afrina Khatun. Prediction of code smell from source code: A hybrid approach. In *2023*

international conference on information and communication technology for sustainable development (ICICT4SD), pages 315–319. IEEE, 2023.

[10] Shivani Jain and Anju Saha. Rank-based univariate feature selection methods on machine learning classifiers for code smell detection. *Evolutionary Intelligence*, 15(1):609–638, 2022.

[11] Seema Dewangan, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. A novel approach for code smell detection: an empirical study. *IEEE Access*, 9:162869–162883, 2021.

[12] Seema Dewangan, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. Code smell detection using ensemble machine learning algorithms. *Applied sciences*, 12(20):10321, 2022.

[13] Nazneen Akhter, Shanto Rahman, and Kazi Abu Taher. An anti-pattern detection technique using machine learning to improve code quality. In *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, pages 356–360. IEEE, 2021.

[14] Inderpreet Kaur and Arvinder Kaur. A novel four-way approach designed with ensemble feature selection for code smell detection. *IEEE Access*, 9:8695–8707, 2021.

[15] Abdou Maiga, Nasir Ali, Neelesh Bhattacharya, Aminata Sabane´, Yann-Gae¨l Gue´he´neuc, Giuliano Antoniol, and Esma Aimeur. Support vector machines for anti-pattern detection. In *Proceedings of the 27th IEEE/ACM international conference on automated software engineering*, pages 278–281, 2012.

[16] Francesca Arcelli Fontana, Mika V Ma¨ntyla¨, Marco Zanoni, and Alessandro Marino. Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3):1143–1191, 2016.

[17] scikit-learn developers. RFECV — Recursive Feature Elimination with Cross-Validation. https://scikit-learn.org/stable/modules/generated/ sklearn.feature_ selection.RFECV.html, 2025. Accessed: Dec. 20, 2025.

[18] GeeksforGeeks. Information Gain and Mutual Information for Machine Learning. https://www.geeksforgeeks.org/machine-learning/ information-gain-and-mutual-information-for-machine-learning/, 2025. Accessed: Dec. 20, 2025.

Biography

Taniz Fatema completed her B.Sc. in Information and Communication Engineering from department of Information and communication Technology of Bangladesh University of Professionals (BUP) and actively contributed to several university clubs, holding key leadership and creative roles. She is currently pursuing her M.Sc. degree in Information and Communication Engineering in the same department at BUP. She had earlier worked as a Research Intern in the Department of Applied Science and Technology Research at Timerni and involved with the 'STAR' project. Her research interests include the application of machine learning and deep learning techniques in software engineering, with a particular focus on intelligent software analysis, code quality assessment, and software defect prediction.

Nazneen Akhter completed her B.Sc. in Information and Communication Engineering and M.Sc. in Information and Communication Engineering from the Department of Information and Communication Technology of Bangladesh University of Professionals, besides several professional skills. She joined the Department of Information and Communication Technology as a lecturer in 2021. She is currently serving as an assistant professor in the Department of Information and Communication Technology at Bangladesh University of Professionals, Dhaka, Bangladesh. Her research areas of interest include software engineering, artificial intelligence and applications, and natural language processing.

Dr. Md Musfique Anwar has awarded PhD degree from Swinburne University of Technology, Melbourne, Australia in 2018. He has received M.Sc. degree from Kyoto University, Japan in 2013 and B.Sc. degree in Computer Science and Engineering from Jahangirnagar University, Bangladesh in 2006. Since 2008, he is a faculty member having current designation Professor in the Department of Computer Science and Engineering of Jahangirnagar University. Currently his research focuses on Data Mining, Social Network Analysis, Natural Language Processing and Software Engineering.