Journal of Computing Theories and Applications ISSN:3024-9104

Research Article

Evaluating Open-Source Machine Learning Project Quality Using SMOTE-Enhanced and Explainable ML/DL Models

Ali Hamza [1,*], Wahid Hussain [2], Hassan Iftikhar [3], Aziz Ahmad [1], and Alamgir Md Shamim [1]

[1] National Research University Higher School of Economics, Moscow 101000, Russian Federation; e-mail : alihamza.369.2@gmail.com, aakhmad@edu.hse.ru, malamgir@edu.hse.ru
[2] National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan; e-mail : wahidhussain@gmail.com
[3] Skolkovo Institute of Science and Technology, Moscow 121205, Russian Federation; e-mail : hassan.iftikhar@skoltech.ru
* Corresponding Author : Ali Hamza

Abstract: The rapid growth of open-source software (OSS) in machine learning (ML) has intensified the need for reliable, automated methods to assess project quality, particularly as OSS increasingly underpins critical applications in science, industry, and public infrastructure. This study evaluates the effectiveness of a diverse set of machine learning and deep learning (ML/DL) algorithms for classifying GitHub OSS ML projects as engineered or non-engineered using a SMOTE-enhanced and explainable modeling pipeline. The dataset used in this research includes both numerical and categorical attributes representing documentation, testing, architecture, community engagement, popularity, and repository activity. After handling missing values, standardizing numerical features, encoding categorical variables, and addressing the inherent class imbalance using the Synthetic Minority Oversampling Technique (SMOTE), seven different classifiers—K-Nearest Neighbors (KNN), Decision Tree (DT), Random Forest (RF), XGBoost (XGB), Logistic Regression (LR), Support Vector Machine (SVM), and a Deep Neural Network (DNN)—were trained and evaluated. Results show that LR (84%) and DNN (85%) outperform all other models, indicating that both linear and moderately deep non-linear architectures can effectively capture key quality indicators in OSS ML projects. Additional explainability analysis using SHAP reveals consistent feature importance across models, with documentation quality, unit testing practices, architectural clarity, and repository dynamics emerging as the strongest predictors. These findings demonstrate that automated, explainable ML/DL-based quality assessment is both feasible and effective, offering a practical pathway for improving OSS sustainability, guiding contributor decisions, and enhancing trust in ML-based systems that depend on open-source components.

Keywords: Explainable AI; GitHub Projects; Machine Learning; Open-Source Software; Quality Assessment; Software Engineering; SMOTE; Deep Neural Networks.

Received: September, 28th 2025
Revised: November, 10th 2025
Accepted: November, 15th 2025
Published: November, 16th 2025

1. Introduction

Machine Learning (ML) has transformed modern technology, permeating numerous industries and driving profound societal changes [1], [2]. Ensuring that ML solutions are reliable, effective, and scalable therefore requires strong software engineering standards [3]. GitHub, as the leading platform for collaborative software development, hosts a vast ecosystem of repositories across diverse fields, including a rapidly growing body of ML and DL projects [4]. It also functions as a central hub for open-source collaboration, providing developers and researchers with rich codebases, documentation, and tools that facilitate contribution and innovation. The platform's collaborative ecosystem encourages knowledge sharing, accelerates idea generation, and supports the development of robust software systems [5], [6]. Consequently, there is an increasing need for systematic and automated approaches to evaluate the quality of ML projects on GitHub. To address this need, the present study employs the NICHE dataset [7], which contains 572 ML repositories categorized as engineered or

non-engineered based on well-established software engineering principles, enabling an evidence-based assessment of OSS ML project quality.

ML and DL, two key subdivisions of artificial intelligence, have demonstrated remarkable progress in recent years [8]. ML algorithms—from simple classifiers to advanced ensemble methods—can leverage dataset features to effectively distinguish between engineered and non-engineered projects. The comprehensiveness of the NICHE dataset also enables the application of DL techniques, such as deep neural networks (DNNs), to capture complex patterns [9]. Using these approaches allows researchers and contributors to derive insights, develop systematic evaluation strategies, and ultimately advance best practices in high-quality ML project development. These models can support the open-source community by providing structured frameworks for project assessment, guiding contributors in identifying high-quality ML projects, and enabling more informed decision-making in collaboration and project selection [10], [11]. By integrating ML- and DL-based evaluation mechanisms, this research fosters an environment of technological excellence and collective progress within the open-source ecosystem.

As open-source ML projects increasingly become embedded in high-stakes real-world applications, their reliability has become a crucial concern. While platforms like GitHub democratize access to powerful tools, low-quality ML components can lead to severe consequences—for example, incorrect predictions in healthcare systems or economic losses in financial automation. The sheer volume of OSS projects makes manual vetting impractical, creating an urgent demand for automated and dependable frameworks to identify high-quality repositories and mitigate risk. This research addresses that challenge by providing a structured, data-driven approach for evaluating ML projects, reducing reliance on popularity indicators and enabling more accurate judgments regarding software quality.

The objective of this study is to bridge the gap between the large number of OSS ML projects and the need for systematic evaluation and categorization. By training ML and DL models on the NICHE dataset, the research aims to accelerate progress in project quality assessment and contribute to the advancement of sustainable software ecosystems. The findings of this work can benefit developers, researchers, and organizations seeking high-quality collaboration opportunities and reliable open-source components, supporting a culture of excellence and innovation within the ML community.

This study makes several key contributions to the automated quality assessment of OSS. First, it presents a rigorous empirical evaluation of seven ML models, demonstrating that Logistic Regression and a Deep Neural Network achieve high accuracy (84–85%) on the NICHE dataset. Second, it introduces a comprehensive preprocessing pipeline that effectively handles class imbalance, missing data, and mixed variable types to ensure reliable model training. Finally, the research provides a dependable framework that helps developers and organizations identify high-quality projects by illuminating the underlying factors that influence OSS ML project quality.

The remainder of this paper is structured as follows. Section II reviews related literature on ML project quality assessment. Section III presents the research methodology, including data collection, preprocessing, and the ML/DL modeling pipeline. Section IV reports the experimental results and insights gained from the predictive analysis. Section V discusses the implications of the findings, practical applications, and potential future directions. Section VI concludes the study.

2. Related Work

The landscape of ML project evaluation and categorization within open-source platforms has amassed significant research interest, particularly given the rapid spread of ML projects. This section reviews the relevant literature, focusing on the methodologies and datasets previously used to identify high-quality ML projects and the role of software engineering practices in these efforts. The assessment of ML project quality has been approached through various methodologies, emphasizing that "engineered" software projects must adhere to stringent software engineering practices as a core criterion for quality assessment. Popularity metrics such as GitHub stars, however, do not necessarily correlate with high-quality engineering practices, challenging the assumption that highly starred projects are well-engineered. Several subsequent studies have attempted to distinguish between popular and well-engineered projects, advocating for a more detailed understanding of project quality.

The mining of software repositories (MSR) has been useful in extracting valuable insights from data available on platforms like GitHub [12]. Researchers have utilized MSR techniques to study various aspects of software development, including bug fixing, developer collaboration, and project evolution, highlighting the potential of MSR for understanding software project dynamics and improving software quality [13], [14]. However, the quality of repositories themselves often poses a significant challenge, as many GitHub projects are of low quality, which can bias MSR findings. OSS has also become integral to commercial software products, services, and corporate processes, supporting open innovation among companies [15]. Research in this area has examined the motivations and behaviors of individual developers contributing to OSS projects, highlighting factors such as personal interest, skill development, and community recognition [16], [17].

Significant research has also explored the challenges faced by developers in using available tools to make technical contributions to OSS projects [18], [19]. These studies underscore the complexity of navigating OSS environments, which often require an understanding of both technical and social dynamics. Corporate engagement with OSS projects is another well-researched area, focusing on the strategic motivations behind corporate contributions and the methodologies used by companies to integrate OSS into their workflows [20]. Some research has highlighted software projects where companies have a controlling influence [21], while in other cases, professionals from different companies collaborate within community OSS initiatives controlled by foundations or organizations. The governance and licensing frameworks of these projects, in conjunction with the strategic interests of participating corporations, significantly influence how contributions are made [22].

Previous studies have highlighted the importance of following established community practices in collaborative environments [23], [24]. Similarly, the governance and licensing of OSS projects, as well as the strategic interests of various participants, influence engagement and contribution decisions [25], [26]. The GitHub Archive, for example, provides a comprehensive collection of GitHub data and has been widely used in MSR research [27]. However, the generality of such datasets often limits their effectiveness in specialized domains like ML. The NICHE dataset helps address this gap by providing a focused collection of ML projects labeled as engineered or non-engineered based on established software engineering practices [7]. Quality metrics are crucial for evaluating software projects, and previous studies have proposed various metrics—code complexity, documentation quality, community engagement, and continuous integration practices—to assess project quality from both technical and social perspectives [28], [29]. For instance, complexity metrics have been shown to predict software defects, highlighting the importance of code quality in overall project assessment [30].

The application of ML to predict software quality has been explored in several studies [31], [32]. Research has demonstrated that Logistic Regression can predict software defects, confirming the potential of ML models in quality assessment [33]. Similarly, other work has conducted comprehensive surveys on ML techniques for software defect prediction, concluding that ensemble methods and neural networks often yield superior performance [34]. DL has also increasingly contributed to software engineering research tools [35], significantly enhancing performance across various software engineering tasks. Software effort estimation has likewise been extensively studied through parametric models such as COCOMO, Function Points, and SLIM, which rely on historical data and predefined formulas to estimate development effort [36]. These models consider factors like team experience, software reliability, programming language, and estimated lines of code, but may not fully capture the unique characteristics of each project. In contrast, ML approaches make fewer assumptions about the function being studied and use historical data to construct rules that generalize to unseen data [37]. Techniques such as regression trees and neural networks, specifically back-propagation, have shown advantages over traditional models in estimating software development effort [38], though these methods also present limitations requiring further research.

DL has also contributed to testing and debugging of DL systems themselves through tools like DeepXplore, DeepTest, DeepGauge, DLFuzz, and DeepHunter, which enhance effectiveness and reliability [39]. ML techniques have been used to classify issue reports in software engineering [40]. For example, a study using issue reports from open-source projects such as HTTPCLIENT, LUCENE, and JACKRABBIT applied various ML algorithms—including Naive Bayes, linear discriminant analysis, KNN, SVM, DT, and RF—to classify reports as bugs or non-bugs [41]. The results showed F-measures between 0.69 and 0.76 and

average classification accuracy between 0.75 and 0.83, indicating that ML can effectively filter issue reports before they reach developers. The study also emphasized the importance of manually classified reports in achieving high-quality automated classification, further highlighting ML's value in improving software quality and efficiency.

Class imbalance is a prevalent issue in software quality prediction studies, where high-quality projects are often greatly outnumbered by low-quality ones. Various techniques, including oversampling, undersampling, and synthetic data generation, have been employed to address this. The use of SMOTE has been particularly effective in improving ML model performance on imbalanced datasets [42], as models trained on imbalanced data tend to favor the majority class, resulting in poor predictive performance for the minority class they aim to identify. SMOTE mitigates this bias by creating synthetic minority instances between existing minority samples and their neighbors [43], resulting in a more balanced distribution and enabling models to learn more accurate decision boundaries [44], [45].

While prior studies have applied ML to OSS quality assessment, many rely on traditional models, overlook class imbalance, or use datasets that are too general to capture the characteristics of ML-specific repositories. A comparative analysis incorporating modern deep learning architectures, trained on a specialized dataset like NICHE, and interpreted through advanced explainability techniques remains notably absent in the literature. This gap highlights the urgency for a balanced, comprehensive, and interpretable evaluation framework capable of accurately assessing OSS ML project quality.

3. Methodology

This section describes the methodology employed in the study, including dataset characteristics, preprocessing procedures, feature engineering steps, balancing techniques, and the machine learning and deep learning models used. The experiments were conducted using the NICHE open-source dataset, which contains 572 ML projects, each labeled as either engineered or non-engineered according to established software engineering principles. A total of 441 projects (77%) are categorized as engineered and 131 (23%) as non-engineered, resulting in a significant class imbalance with a ratio of approximately 3.3:1. Table I summarizes the distribution of project quality.

Table 1. Summary of engineered vs non-engineered projects.

Category	Count	Percentage
Engineered	441	77%
Non-Engineered	131	23%
Total	572	100%

3.1. Dataset Characteristics and Feature Distributions

The dataset contains essential metadata such as the number of stars, commits, and lines of code, representing project popularity, development activity, and project size. Correlation analysis was conducted to understand feature dependencies and identify predictors strongly associated with project quality. Figure 1 presents the correlation matrix, showing that Unit Testing ($r = 0.57$), Architecture ($r = 0.40$), and Documentation ($r = 0.36$) exhibit the strongest associations with engineered status. In contrast, popularity indicators such as Stars ($r = 0.03$) show negligible correlation, indicating that project popularity alone cannot reliably indicate project quality.

A distributional analysis of numerical features further supports these observations. As shown in Figure 2, engineered projects tend to have higher median values of both stars and commits, yet the distributions are heavily right-skewed, suggesting non-linear relationships that simple linear models may fail to capture.

3.2. Preprocessing and Textual Normalization

A comprehensive preprocessing pipeline was applied to ensure data consistency and modeling reliability. The dataset contained multiple text-based attributes exhibiting inconsistent patterns. These attributes were normalized using a combination of string similarity matching and manual verification.

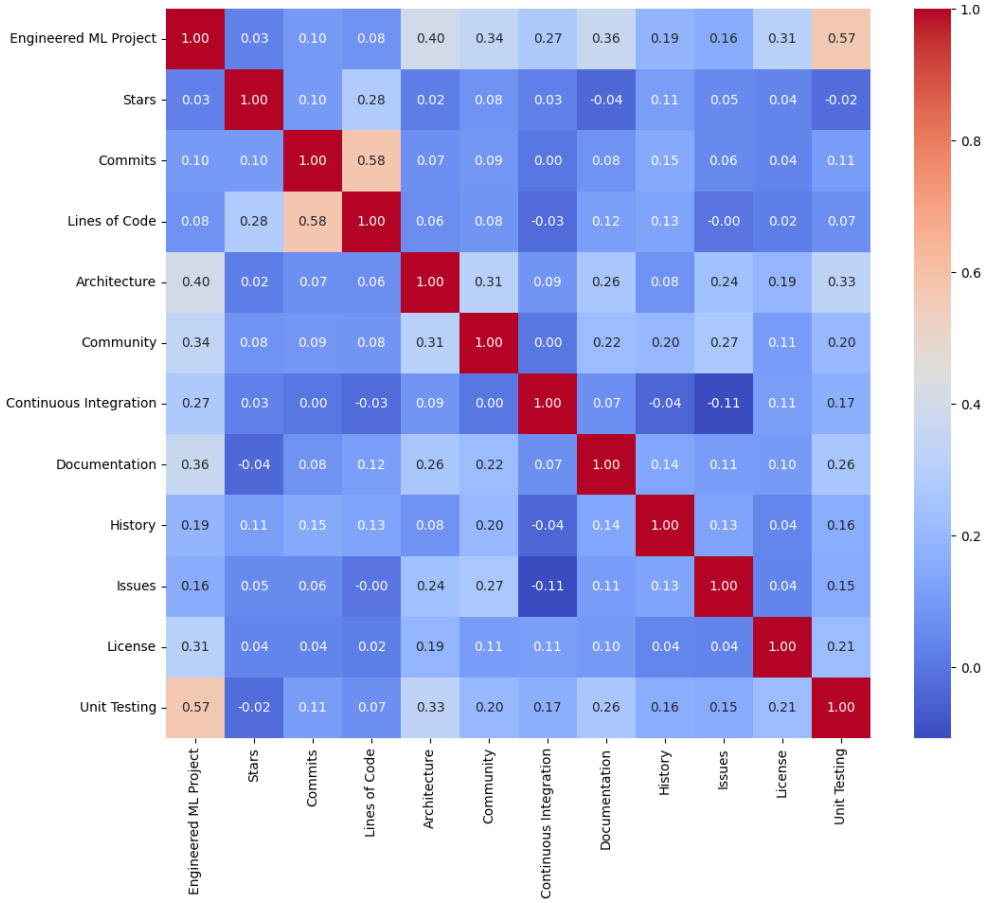


Figure 1. Correlation Matrix of Dataset Features

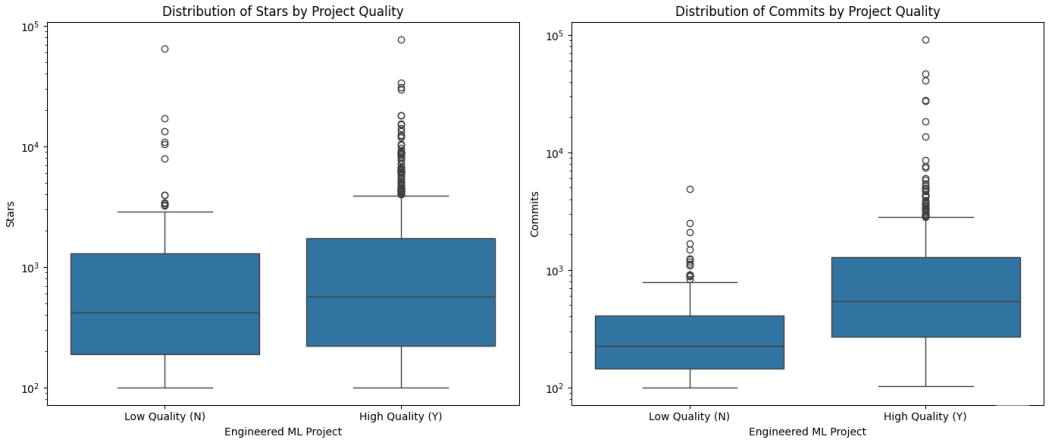


Figure 2. Distribution of Stars and Commits by Project Quality

To group semantically similar entries, the Levenshtein Distance algorithm was used:

$$\text{lev}(a,b) = \begin{cases} |a| & b = \emptyset \\ |b| & a = \emptyset \\ \text{lev}\big(\text{tail}(a), \text{tail}(b)\big) & \text{if head}(a) = \text{head}(b) \\ 1 + \min \begin{cases} \text{lev}(tail(a), b) \\ \text{lev}\big(a, tail(b)\big) \\ \text{lev}\big(tail(a), tail(b)\big) \end{cases} & \text{otherwise} \end{cases} \quad (1)$$

After automated grouping, a manual verification step was performed to correct classification errors. Textual fields—such as architecture, documentation, community, license,

testing, and issue handling—were categorized into meaningful labels (e.g., Well-maintained, Uncertain, Inactive, Active). Missing values were removed due to their small proportion. Categorical variables were encoded using Label Encoding or One-Hot Encoding, depending on their semantic structure.

3.3. Feature Engineering

To enhance representation quality, two additional engineered features were created:

$$\text{Commits per Star} = \frac{\text{Commits}}{\text{Stars}} \tag{2}$$

$$\text{Line of Code per Commits} = \frac{\text{Line\_of\_Code}}{\text{Commits}} \tag{3}$$

These derived metrics capture deeper project dynamics, such as development intensity relative to popularity and average contribution size. Continuous variables (Stars, Commits, Lines of Code, and engineered ratios) were subsequently standardized using:

$$x' = \frac{x - \mu}{\sigma} \tag{4}$$

Where μ is the feature mean and σ its standard deviation.

3.4. Handling Class Imbalance Using SMOTE

Given the imbalance between engineered and non-engineered classes, the Synthetic Minority Oversampling Technique (SMOTE) was employed to generate synthetic examples for the minority class using the default parameter $k = 5$ neighbors.

$$x_{\text{new}} = x_{\text{minority}} + \delta \cdot (x_{\text{neigbor}} - x_{\text{minority}}) \tag{5}$$

Where $\delta \in [0,1]$ is a random interpolation weight.

SMOTE was applied after scaling, but before the train–test split to avoid data leakage.

3.5. Overall Processing Pipeline

The complete methodology is illustrated in Figure 3, covering data collection, preprocessing, SMOTE balancing, model training, and evaluation.

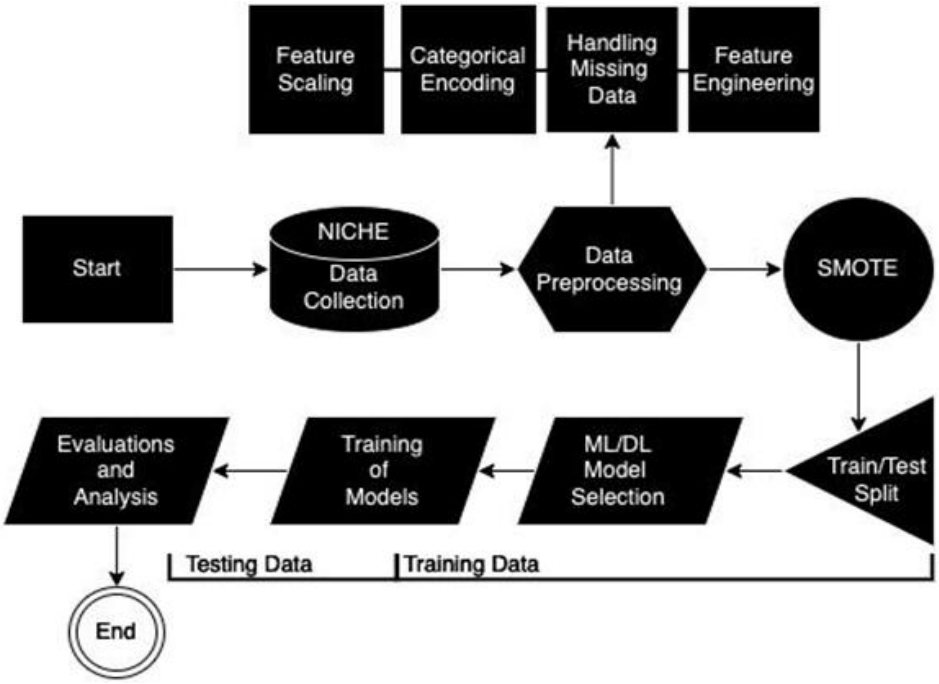


Figure 3. Overall methodology pipeline

3.6. Machine Learning and Deep Learning Models

Seven classification algorithms were evaluated, i.e. Logistic Regression (LR), K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), XGBoost (XGB), Deep Neural Network (DNN). Each model was trained using the same preprocessed dataset to ensure fair comparison. Traditional models (LR, KNN, SVM) serve as baselines, while tree-based models (RF, XGB) capture complex nonlinear relations. A feed-forward DNN was implemented to investigate the capacity of deep models to learn feature interactions automatically. The full ML workflow is formalized in Algorithm 1, which provides a structured description of each step, from preprocessing through evaluation.

Algorithm 1. Project Classification Using Machine Learning (ML)

INPUT: NICHE dataset D, preprocessing procedures P, set of ML models M
OUTPUT: performance metrics for each model
Data Gathering:
    Collect dataset D from NICHE (available on GitHub).
Data Preprocessing:
    Handle missing values using appropriate imputation techniques.
    Encode categorical attributes into numerical format.
    Normalize numerical features using standard scaling, Equation (4).
Feature Engineering:
    Create additional derived features and bin numerical attributes where appropriate.
Handling Class Imbalance:
    Apply SMOTE to generate synthetic samples for the minority class, Equation (5).
Model Selection:
    Initialize model set $M = \{LR, KNN, SVM, DT, RF, XGB\}$
Training and Evaluation:
    Split dataset into training and testing sets:
    $D = D_{\text{train}} \cup D_{\text{test}}$
For each model $m \in M$
    Train m using D_{train}
    Evaluate using D_{test} with accuracy, precision, recall, and F1-score.
End For
Results Analysis:
    Compare model performance and discuss implications for project quality assessment.

For the deep learning model, a separate training routine was required due to the iterative nature of neural network optimization. The DNN was trained using mini-batch gradient descent, backpropagation, and the Adam optimizer, with binary cross-entropy serving as the loss function. The training loop included forward propagation, gradient computation, weight updates, and validation monitoring. The complete DNN workflow is summarized in Algorithm 2.

Algorithm 2. Deep Neural Network (DNN) Training and Validation

INPUT: Training data $(X_{\text{train}}, y_{\text{train}})$, validation data $(X_{\text{val}}, y_{\text{val}})$, hyperparameters (E, B, η)
OUTPUT: Trained DNN model M
Initialize model M with random weights.
Initialize Adam optimizer with learning rate η
Define Binary Cross-Entropy loss function.

Training Phase:
For epoch $e = 1$ to E
    Set model M to training mode.
    For each batch $(X_{\text{b}}, y_{\text{b}})$, in training data:
        Zero optimizer gradients.
        Compute predictions $y_{\text{pred}} = M(X_{\text{b}})$

Algorithm 2. Deep Neural Network (DNN) Training and Validation

```
        Compute loss L = BCE(y_pred, y_b)
        Perform backpropagation: L.backward()
        Update weights using optimizer.
    End for

Validation phase:
    Set model M to validation mode.
    Compute validation loss and validation accuracy on (X_val, y_val)
    Optionally apply early stopping.
End for
Return trained model M
```

Table 2 reports the final hyperparameter configurations used for all classifiers.

Table 2. Classifier hyperparameters.

Classifier	Parameter	Value
DNN	Units	64
	Activation	ReLU
	Loss	Binary Crossentropy
	Optimizer	Adam
	Learning Rate	0.001
	Batch Size	32
	Epochs	50
	Final Activation	Sigmoid
KNN	n_neighbors	5
	Metric	Minkowski
	Weights	Uniform
SVM	Kernel	RBF
	C	0.9
	Gamma	Scale
	Probability	False
Decision Tree	Criterion	Gini
	Min Samples Split	2
	Max Depth	None
Logistic Regression	Penalty	L2
	Fit Intercept	True
	Tolerance	0.0001
XGBoost	Learning Rate	0.3
	n_estimators	110
	Objective	binary: logistic
Random Forest	Criterion	Gini
	n_estimators	110
	Max Features	sqrt
	Max Features	sqrt

4. Results and Discussion

This section presents a comprehensive evaluation of the machine learning and deep learning models used to classify GitHub ML projects as engineered or non-engineered. The assessed models include KNN, DT, XGBoost, Random Forest, Logistic Regression, SVM, and DNN. For each model, performance was measured using accuracy, precision, recall, and F1-score. These metrics collectively provide insight into the predictive capabilities of the models across both classes.

4.1. K-Nearest Neighbors (KNN)

KNN showed strong performance, with an accuracy of 0.83. The precision for class 0 (non-engineered) was 0.63, with a recall of 0.70 and an F1-score of 0.67. For class 1 (engineered), the precision was 0.91, the recall was 0.88, and the F1 score was 0.89. The confusion matrix results were as follows: 77 true positives, 19 true negatives, 8 false positives, and 11 false negatives. This indicates that the model is good at minimizing false positives. These results indicate that KNN is effective in predicting engineered projects but may need improvement in accurately predicting non-engineered projects. Table 3 shows the classification report for KNN.

Table 3. Classification report for KNN.

Class	Precision	Recall	F1-Score	Support
0	0.63	0.70	0.67	27
1	0.91	0.88	0.89	88
Accuracy			0.83	
Macro Avg	0.77	0.79	0.78	115
Weighted Avg	0.84	0.83	0.84	115

4.2. Decision Tree (DT)

The DT achieved an accuracy of 0.73. The precision for class 0 was 0.44, with a recall of 0.52 and an F1 score of 0.47. For class 1, the precision was 0.84, the recall was 0.80, and the F1 score was 0.82. The confusion matrix showed 70 true positives, 14 true negatives, 13 false positives, and 18 false negatives. This shows a tendency towards higher precision at the cost of lower recall. The model performed well for engineered projects but needed to improve in accurately classifying non-engineered projects. See Table 4 for the classification report.

Table 4. Classification report for DT.

Class	Precision	Recall	F1-Score	Support
0	0.44	0.52	0.47	27
1	0.84	0.80	0.82	88
Accuracy			0.73	
Macro Avg	0.64	0.66	0.65	115
Weighted Avg	0.75	0.73	0.74	115

4.3. XGBoost (XGB)

XGBoost exhibited a balanced performance with an accuracy of 0.79. The precision for class 0 was 0.56, with a recall of 0.56 and an F1 score of 0.56. For Class 1, the precision, recall, and F1 score were all 0.86. The confusion matrix showed 76 true positives, 15 true negatives, 12 false positives, and 12 false negatives. XGBoost showed consistent performance across both classes, although its ability to identify non-engineered projects was moderate. Table 5 shows the classification Report for XGB.

Table 5. Classification report for XGB.

Class	Precision	Recall	F1-Score	Support
0	0.56	0.56	0.56	27
1	0.86	0.86	0.86	88
Accuracy			0.79	
Macro Avg	0.71	0.71	0.71	115
Weighted Avg	0.79	0.79	0.79	115

4.4. Random Forest (RF)

Random Forest achieved an accuracy of 80%. The precision for class 0 was 0.58, with a recall of 0.52 and an F1 score of 0.55. For class 1, the precision was 0.86, the recall was 0.89,

and the F1 score was 0.87. The confusion matrix showed 78 true positives, 14 true negatives, 13 false positives, and 10 false negatives. Random Forest was effective in classifying engineered projects, with slightly better performance for non-engineered projects compared to XGBoost. Table 6 shows the classification Report for RF.

Table 6. Classification report for RF.

Class	Precision	Recall	F1-Score	Support
0	0.58	0.52	0.55	27
1	0.86	0.89	0.87	88
Accuracy			0.80	
Macro Avg	0.72	0.70	0.71	115
Weighted Avg	0.79	0.80	0.80	115

4.5. Logistic Regression (LR)

LR demonstrated excellent performance, with an accuracy of 0.84. For class 0, recall was particularly high (0.85), and precision was 0.62, resulting in an F1-score of 0.72. For class 1, precision and F1-score were 0.95 and 0.89, respectively. The confusion matrix (74 TP, 23 TN, 4 FP, 14 FN) indicates that LR balances both classes more effectively than tree-based models.

Table 7. Classification report for LR.

Class	Precision	Recall	F1-Score	Support
0	0.62	0.85	0.72	27
1	0.95	0.84	0.89	88
Accuracy			0.84	
Macro Avg	0.79	0.85	0.81	115
Weighted Avg	0.87	0.84	0.85	115

4.6. Support Vector Machine (SVM)

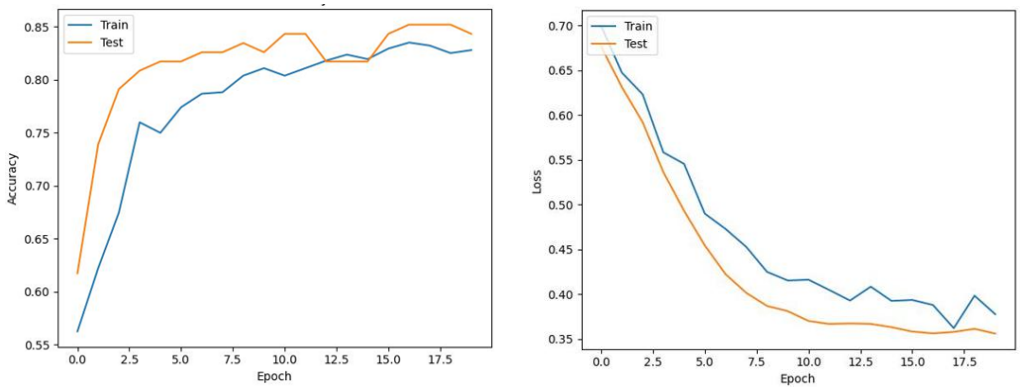
SVM achieved an accuracy of 83%. The precision for class 0 was 0.61, with a recall of 0.70 and an F1 score of 0.66. For class 1, the precision was 0.90, the recall was 0.86, and the F1 score was 0.88. The confusion matrix showed 76 true positives, 19 true negatives, 8 false positives, and 12 false negatives. SVM performed well for both classes, with high precision and recall for engineered projects, similar to KNN. Table 8 shows the classification Report for SVM.

Table 8. Classification report for SVM.

Class	Precision	Recall	F1-Score	Support
0	0.61	0.70	0.66	27
1	0.90	0.86	0.88	88
Accuracy			0.83	
Macro Avg	0.76	0.78	0.77	115
Weighted Avg	0.84	0.83	0.83	115

4.7. Deep Neural Network (DNN)

DNN also showed strong performance, with an accuracy of 0.84. The precision for class 0 was 0.62, with a recall of 0.85 and an F1 score of 0.72. For class 1, the precision was 0.95, the recall was 0.84, and the F1 score was 0.89. The confusion matrix showed 74 true positives, 23 true negatives, 4 false positives, and 14 false negatives. DNN's performance metrics were comparable to those of LR, highlighting its effectiveness in classifying both engineered and non-engineered projects. Table 9 shows the classification report for DNN. Training behaviour is illustrated in Fig. 4, showing smooth convergence of accuracy and loss without overfitting.



(a) (b)

Figure 4. Train and testing behaviour of DNN (**a**) Accuracy; (**b**) Loss.

Table 9. Classification report for DNN.

Class	Precision	Recall	F1-Score	Support
0	0.62	0.85	0.72	27
1	0.95	0.84	0.89	88
Accuracy			0.84	
Macro Avg	0.79	0.85	0.81	115
Weighted Avg	0.87	0.84	0.85	115

4.8. Impact of SMOTE

Given the substantial class imbalance in the NICHE dataset, where engineered projects account for roughly 77% of all samples, the application of SMOTE played a crucial role in improving model performance, particularly for the minority class. SMOTE generates synthetic examples by interpolating between existing minority samples and their nearest neighbours, enabling the classifier to learn the characteristics of underrepresented non-engineered projects better. The results summarized in Table 10 show that the impact of SMOTE varies significantly across models.

Table 10. SMOTE's impact on model performance.

Model	Treatment	Recall (Class 0,1)	Precision (Class 0,1)	F1-Score (Class 0,1)	Overall Accuracy
LR	Without SMOTE	0.37, 0.88	0.77, 0.83	0.50, 0.89	0.82
	With SMOTE	0.85, 0.84	0.62, 0.95	0.72, 0.89	0.84
KNN	Without SMOTE	0.79, 0.92	0.88, 0.88	0.68, 0.92	0.79
	With SMOTE	0.70, 0.86	0.63, 0.91	0.67, 0.89	0.83
DT	Without SMOTE	0.44, 0.91	0.55, 0.85	0.46, 0.87	0.73
	With SMOTE	0.44, 0.80	0.52, 0.84	0.47, 0.82	0.73
RF	Without SMOTE	0.33, 0.83	0.82, 0.93	0.47, 0.90	0.83
	With SMOTE	0.52, 0.89	0.58, 0.88	0.55, 0.87	0.80
XGB	Without SMOTE	0.30, 0.98	0.80, 0.82	0.43, 0.85	0.82
	With SMOTE	0.56, 0.86	0.56, 0.86	0.56, 0.86	0.80
SVM	Without SMOTE	0.52, 0.95	0.78, 0.87	0.62, 0.91	0.81
	With SMOTE	0.70, 0.86	0.61, 0.90	0.66, 0.88	0.83
DNN	Without SMOTE	0.52, 0.94	0.74, 0.86	0.61, 0.90	0.83
	With SMOTE	0.85, 0.84	0.62, 0.95	0.72, 0.89	0.84

LR and DNN benefited the most, particularly in improving the recall and F1-score of class 0. For instance, LR's recall for class 0 increased dramatically from 0.37 to 0.85, while its

F1-score rose from 0.50 to 0.72. A similar improvement occurred in the DNN model, where the F1-score for the minority class increased from 0.61 to 0.72. These improvements suggest that oversampling helped these linear and deep models capture patterns that were previously overshadowed by the dominant class.

Models like KNN and SVM also experienced moderate improvements, especially in recall. However, this came at the cost of a slight reduction in precision—an expected trade-off when synthetic samples increase the likelihood of false positives. In contrast, tree-based models, such as Decision Trees, Random Forests, and XGBoost, displayed mixed behaviour. The Decision Tree showed virtually no change, indicating that synthetic data had a lesser influence on its rule-based structure. Meanwhile, Random Forest and XGBoost experienced marginal declines in accuracy after applying SMOTE, suggesting that these ensemble models may rely more on natural feature distributions and may not always benefit from oversampling. Overall, the results demonstrate that SMOTE can significantly enhance performance for specific types of models, particularly LR, SVM, and DNN; however, its effects are not universally positive across all algorithms. These findings underscore the importance of matching imbalance-handling strategies to the specific modelling approach rather than assuming a one-size-fits-all solution.

4.9. Model Visualizations

To complement the numerical evaluations, a series of visual analyses was conducted to understand model behaviour better and interpret predictive patterns. The confusion matrices in Figure 5 provide a clear illustration of how each model differentiates between engineered and non-engineered projects. LR and DNN show balanced and reliable classification patterns with relatively few false predictions across both classes, reinforcing their strong performance in earlier metrics. KNN and SVM also demonstrate stable classification behavior, especially for engineered projects, although both exhibit slightly weaker detection of non-engineered samples. Meanwhile, tree-based models reveal higher false negatives, suggesting that they may be more sensitive to borderline or ambiguous cases.

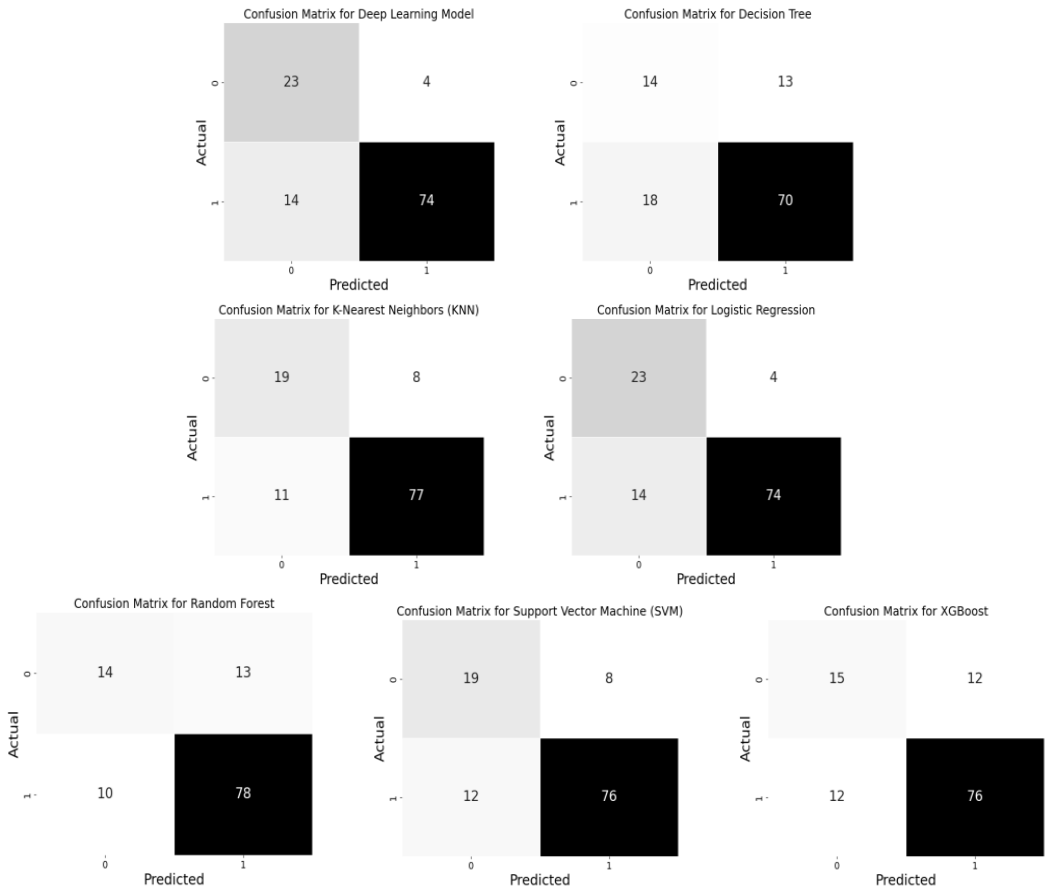


Figure 5. Confusion matrix of each model.

The ROC curves in Figure 6 further illustrate the models' discriminative abilities. LR, DNN, KNN, and Random Forest achieve high ROC-AUC values, indicating strong performance across a range of decision thresholds. SVM likewise performs well, with smooth, convex ROC behavior. In contrast, the Decision Tree exhibits a less favourable ROC curve, consistent with its lower accuracy and greater susceptibility to overfitting. These results highlight that while accuracy offers a single measure of performance, ROC analysis provides wider insight into how models behave under different decision boundary settings.

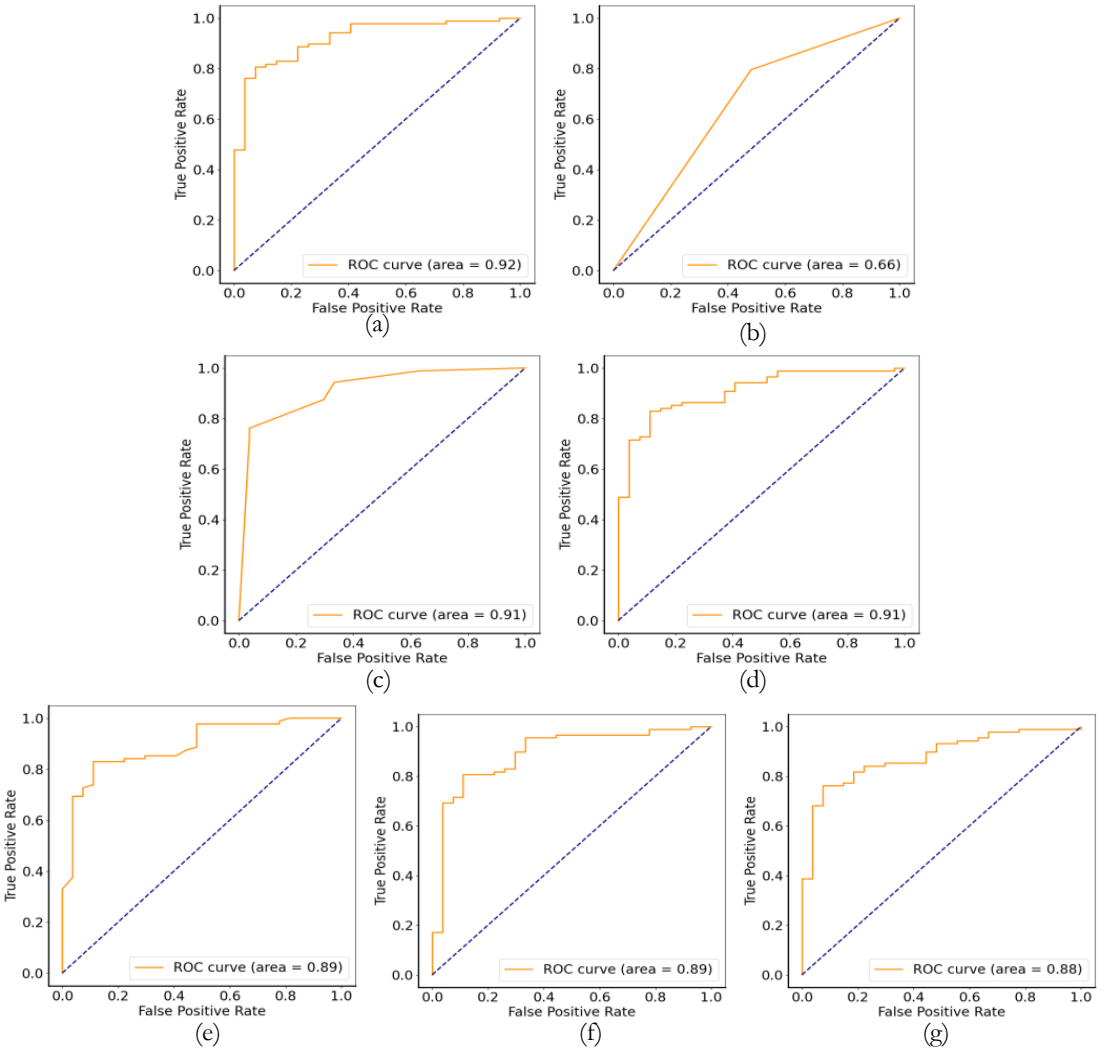


Figure 6. ROC curves of each models (**a**) DNN; (**b**) DT; (**c**) KNN; (d) LR; (**e**) RF; (**f**) SVM; (**g**) XGBoost.

To explore interpretability and feature contributions, SHAP decision plots were generated for LR, DNN, DT, RF, and XGB, as shown in Figure 7. Across all models, features such as Unit Testing, Documentation, Architecture quality, and Issue handling consistently emerge as the most important contributors to prediction outcomes. The decision paths differ significantly across model families: LR shows straight, parallel patterns characteristic of additive linear effects, whereas DNN exhibits smooth and intersecting trajectories that reflect its capacity to model non-linear feature interactions. Ensemble trees, such as RF and XGB, display smoothed averaging of multiple decision paths, illustrating how they combine numerous weak rules to form more robust predictions. These interpretability results reinforce that OSS project quality involves both linear and non-linear relationships, and different models—especially LR and DNN—capture complementary aspects of these patterns.

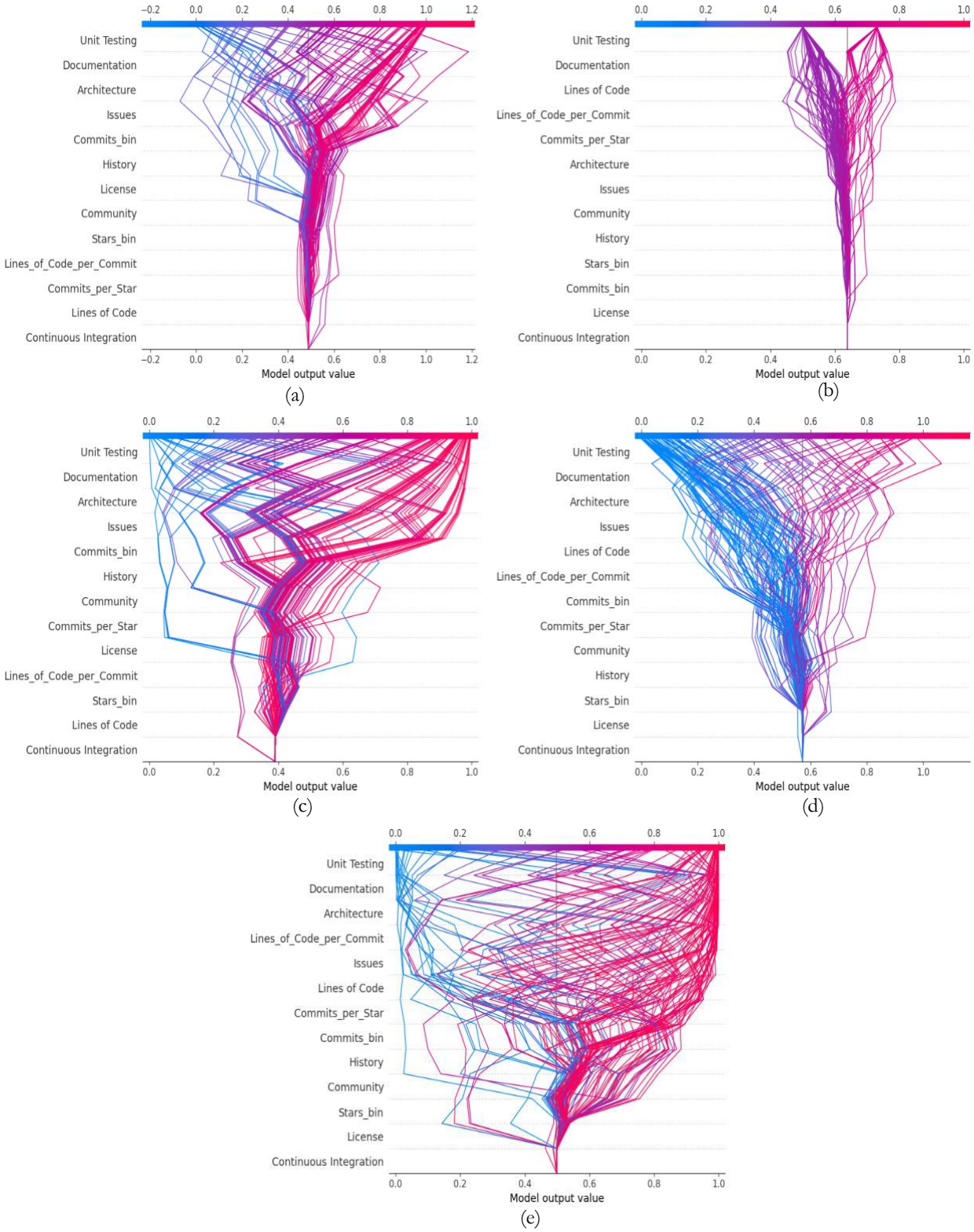


Figure 7. SHAP plot results (**a**) DNN; (**b**) DT; (**c**) LR; (**d**) RF; (**e**) XGBoost

4.10. Top-10 Feature Ablation Study

To further validate the feature importance results derived from the SHAP analysis, an ablation study was performed using the Deep Neural Network (DNN) model, which had previously exhibited the best overall performance. The objective of this experiment was to determine whether the features ranked highest by SHAP truly represent the primary drivers of predictive performance. Based on the mean absolute SHAP values, the ten most influential

features were selected, primarily dominated by factors such as Unit Testing, Documentation quality, Architecture, Issue management, and Project History.

A new DNN model was then trained exclusively on these ten features while maintaining the same preprocessing pipeline, including the application of SMOTE to handle class imbalance. The performance of this reduced-feature model was compared directly with the full-feature DNN model. As summarized in Table 11, the ablation results show only a marginal reduction in performance: accuracy decreased slightly from 0.84 to 0.83, and the F1-score from 0.81 to 0.79. Precision and recall also remained highly comparable between the two model configurations.

Table 11. Performance comparison DNN for top-10 feature ablation study.

Model Configuration	Accuracy	F1-Score	Precision	Recall
Full Feature Set	84%	0.81	0.79	0.85
Top-10 Features Only	83%	0.79	0.78	0.83

These findings reinforce two important conclusions. First, the top SHAP-ranked features indeed capture most of the predictive signal needed for distinguishing between engineered and non-engineered ML projects, confirming the reliability of the SHAP importance estimates. Second, the minimal performance degradation suggests that a more compact, computationally efficient model—using only the ten most critical features—could be deployed without significant loss in predictive capability. This provides practical value for real-world applications where model simplicity, explainability, and reduced data requirements are advantageous.

5. Conclusions

This study investigated the application of various machine learning (ML) and deep learning (DL) models for classifying GitHub ML projects as engineered or non-engineered. The models evaluated included KNN, DT, XGB, RF, LR, SVM, and DNN. Among these, Logistic Regression, SVM, and DNN demonstrated the highest performance, each achieving an accuracy of 0.84. These findings suggest that ML and DL models have strong potential for systematically evaluating and categorizing open-source ML projects. The approach presented in this study provides a practical framework that can assist researchers and practitioners in identifying high-quality projects, thereby supporting more informed collaboration and decision-making within the open-source ecosystem.

Despite these promising results, several limitations should be acknowledged. Although the NICHE dataset provides useful indicators of project quality, it does not encompass the full range of characteristics needed for a comprehensive assessment. Important attributes such as code complexity, technologies used in the projects, their adoption levels, and long-term relevance are not captured. Furthermore, while SMOTE effectively addressed class imbalance, it relies on synthetic samples that may not fully represent real-world distributions, which could influence the generalizability of the models. Another limitation is that the evaluation was conducted using a static snapshot of the dataset. Open-source projects evolve continuously, with changes in code, contributors, and activity over time. Consequently, models trained on static data may not fully capture these temporal dynamics, which can potentially reduce their long-term applicability.

Future research should address these limitations by incorporating a broader set of quality indicators and developing methods to dynamically update datasets to reflect the ongoing evolution of the project. Investigating more advanced techniques for handling class imbalance—beyond synthetic oversampling—may also improve model robustness. Additionally, integrating explainable AI methods could enhance interpretability, offering deeper insight into model decisions and increasing user trust in automated quality assessment tools. Efforts should also be directed toward designing automated and objective metrics for features such as documentation quality and testing practices, reducing subjectivity and improving consistency in labelling. Conducting longitudinal studies that track project quality over time would yield richer insights into the factors contributing to sustained excellence in open-source ML projects. Moreover, future work may explore more transparent and inherently interpretable deep learning architectures, alongside incorporating finer-grained code quality metrics to capture project

dynamics more accurately. By addressing these directions, future research can build upon the foundation laid in this study, enabling more accurate evaluation and stronger support for high-quality ML projects in the open-source community.

Author Contributions: Conceptualization: A.H. and W.H.; Methodology: A.H.; Software: A.H., W.H, and H.I.; Validation: A.H., and A.A.; Formal analysis: A.H., H.I. and A.M.S.; Investigation: A.H. and W.H.; Resources: A.H. and A.M.S; Data curation: H.I. and A.M.S.; Writing—original draft preparation: A.H. and W.H.; Writing—review and editing: A.H, H.I. and A.A.; Visualization: A.A. and A.M.S; Supervision: A.H.; Project administration: A.H. and W.H.; Funding acquisition: A.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data supporting the findings of this study are openly available in the NICHE repository at: https://github.com/soarsmu/NICHE/blob/main/NICHE.csv DOI: https://doi.org/10.48550/arXiv.2303.06286

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] R. Sharma, "The Transformative Power of AI as Future GPTs in Propelling Society Into a New Era of Advancement," *IEEE Eng. Manag. Rev.*, vol. 51, no. 4, pp. 215–224, Dec. 2023, doi: 10.1109/EMR.2023.3315191.

[2] T. V. N. Rao, A. Gaddam, M. Kurni, and K. Saritha, "Reliance on Artificial Intelligence, Machine Learning and Deep Learning in the Era of Industry 4.0," in *Smart Healthcare System Design*, Wiley, 2022, pp. 281–299. doi: 10.1002/9781119792253.ch12.

[3] M. N. Chaudhry, S. S. U. Din, Z. U. R. Zia, M. K. Abid, and N. Aslam, "Achieving Scalable and Secure Systems: The Confluence of ML, AI, Iot, Block-chain, and Software Engineering," *J. Comput. Biomed. Informatics*, 2024, [Online]. Available: https://jcbi.org/index.php/Main/article/view/359

[4] V. Cosentino, J. L. Canovas Izquierdo, and J. Cabot, "A Systematic Mapping Study of Software Development With GitHub," *IEEE Access*, vol. 5, pp. 7173–7192, 2017, doi: 10.1109/ACCESS.2017.2682323.

[5] N. McDonald and S. Goggins, "Performance and participation in open source software on GitHub," in *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, Apr. 2013, pp. 139–144. doi: 10.1145/2468356.2468382.

[6] E. Kalliamvakou, D. Damian, K. Blincoe, L. Singer, and D. M. German, "Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, May 2015, pp. 574–585. doi: 10.1109/ICSE.2015.74.

[7] R. Widyasari *et al.*, "NICHE: A Curated Dataset of Engineered Machine Learning Projects in Python," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, May 2023, pp. 62–66. doi: 10.1109/MSR59073.2023.00022.

[8] S. Fahle, C. Prinz, and B. Kuhlenkötter, "Systematic review on machine learning (ML) methods for manufacturing processes – Identifying artificial intelligence (AI) methods for field application," *Procedia CIRP*, vol. 93, pp. 413–418, 2020, doi: 10.1016/j.procir.2020.04.109.

[9] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-Driven Off-Loading for DNN-Based Applications Over Cloud, Edge, and End Devices," *IEEE Trans. Ind. Informatics*, vol. 16, no. 8, pp. 5456–5466, Aug. 2020, doi: 10.1109/TII.2019.2961237.

[10] O. Lock, M. Bain, and C. Pettit, "Towards the collaborative development of machine learning techniques in planning support systems – a Sydney example," *Environ. Plan. B Urban Anal. City Sci.*, vol. 48, no. 3, pp. 484–502, Mar. 2021, doi: 10.1177/2399808320939974.

[11] F. de Arriba-Pérez, S. García-Méndez, J. Otero-Mosquera, F. J. González-Castaño, and F. Gil-Castiñeira, "Automatic Generation of Insights From Workers' Actions in Industrial Workflows With Explainable Machine Learning: A Proposed Architecture With Validation," *IEEE Ind. Electron. Mag.*, vol. 18, no. 2, pp. 17–29, Jun. 2024, doi: 10.1109/MIE.2023.3284203.

[12] S. Kourtzanidis, A. Chatzigeorgiou, and A. Ampatzoglou, "RepoSkillMiner: identifying software expertise from GitHub repositories using natural language processing," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, Dec. 2020, pp. 1353–1357. doi: 10.1145/3324884.3415305.

[13] F. Wen, C. Nagy, M. Lanza, and G. Bavota, "Quick remedy commits and their impact on mining software repositories," *Empir. Softw. Eng.*, vol. 27, no. 1, p. 14, Jan. 2022, doi: 10.1007/s10664-021-10051-z.

[14] D. Spadini, M. Aniche, and A. Bacchelli, "PyDriller: Python framework for mining software repositories," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Oct. 2018, pp. 908–911. doi: 10.1145/3236024.3264598.

[15] D. Nagy, A. M. Yassin, and A. Bhattacherjee, "Organizational adoption of open source software," *Commun. ACM*, vol. 53, no. 3, pp. 148–151, Mar. 2010, doi: 10.1145/1666420.1666457.

[16] A. Bonaccorsi and C. Rossi, "Why Open Source software can succeed," *Res. Policy*, vol. 32, no. 7, pp. 1243–1258, Jul. 2003, doi: 10.1016/S0048-7333(03)00051-9.

[17] J. Feller, *Perspectives on Free and Open Source Software*. The MIT Press, 2005. doi: 10.7551/mitpress/5326.001.0001.

[18] S. Butler *et al.*, "An investigation of work practices used by companies making contributions to established OSS projects," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, May 2018, pp. 201–210. doi: 10.1145/3183519.3183531.
[19] W. Scacchi, "Socio-Technical Interaction Networks in Free/Open Source Software Development Processes," in *Software Process Modeling*, New York: Springer-Verlag, 2005, pp. 1–27. doi: 10.1007/0-387-24262-7_1.
[20] V. K. Gurbani, A. Garvert, and J. D. Herbsleb, "A case study of a corporate open source development model," in *Proceedings of the 28th international conference on Software engineering*, May 2006, pp. 472–481. doi: 10.1145/1134285.1134352.
[21] S. Butler *et al.*, "On Company Contributions to Community Open Source Software Projects," *IEEE Trans. Softw. Eng.*, vol. 47, no. 7, pp. 1381–1401, Jul. 2021, doi: 10.1109/TSE.2019.2919305.
[22] L. Dahlander and M. G. Magnusson, "Relationships between open source software companies and communities: Observations from Nordic firms," *Res. Policy*, vol. 34, no. 4, pp. 481–493, May 2005, doi: 10.1016/j.respol.2005.02.003.
[23] S. Arto, "Open Source in Finnish Software Companies," *ETLA Econ. Res.*, 2006, [Online]. Available: https://www.etla.fi/en/publications/dp1002-en/
[24] K. Mouakhar and A. Tellier, "How do Open Source software companies respond to institutional pressures? A business model perspective," *J. Enterp. Inf. Manag.*, vol. 30, no. 4, pp. 534–554, Jul. 2017, doi: 10.1108/JEIM-05-2015-0041.
[25] R. Sen, C. Subramaniam, and M. L. Nelson, "Determinants of the Choice of Open Source Software License," *J. Manag. Inf. Syst.*, vol. 25, no. 3, pp. 207–240, Dec. 2008, doi: 10.2753/MIS0742-1222250306.
[26] T. August, W. Chen, and K. Zhu, "Competition Among Proprietary and Open-Source Software Firms: The Role of Licensing in Strategic Contribution," *Manage. Sci.*, vol. 67, no. 5, pp. 3041–3066, May 2021, doi: 10.1287/mnsc.2020.3674.
[27] V. Markovtsev and W. Long, "Public git archive," in *Proceedings of the 15th International Conference on Mining Software Repositories*, May 2018, pp. 34–37. doi: 10.1145/3196398.3196464.
[28] M. Shahin, M. Ali Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017, doi: 10.1109/ACCESS.2017.2685629.
[29] C. Anderson, "Quality assurance practices in open-source projects: Nurturing excellence in collaborative development," *J. Sci. Technol.*, vol. 3, no. 4, pp. 23–36, 2022, [Online]. Available: https://thesciencebrigade.com/jst/article/view/61
[30] S. Omri, P. Montag, and C. Sinz, "Static Analysis and Code Complexity Metrics as Early Indicators of Software Defects," *J. Softw. Eng. Appl.*, vol. 11, no. 04, pp. 153–166, 2018, doi: 10.4236/jsea.2018.114010.
[31] K. Srinivasan and D. Fisher, "Machine learning approaches to estimating software development effort," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 126–137, 1995, doi: 10.1109/32.345828.
[32] S. Goyal and P. K. Bhatia, "Comparison of Machine Learning Techniques for Software Quality Prediction," *Int. J. Knowl. Syst. Sci.*, vol. 11, no. 2, pp. 20–40, Apr. 2020, doi: 10.4018/IJKSS.2020040102.
[33] J. Goyal and R. Ranjan Sinha, "Software Defect-Based Prediction Using Logistic Regression: Review and Challenges," in *Second International Conference on Sustainable Technologies for Computational Intelligence*, Springer, 2022, pp. 233–248. doi: 10.1007/978-981-16-4641-6_20.
[34] A. Alsaeedi and M. Z. Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study," *J. Softw. Eng. Appl.*, vol. 12, no. 05, pp. 85–100, 2019, doi: 10.4236/jsea.2019.125007.
[35] P. Devanbu *et al.*, "Deep Learning & Software Engineering: State of Research and Future Directions," *ArXiv*. Sep. 17, 2020. [Online]. Available: http://arxiv.org/abs/2009.08525
[36] P. Singal, A. C. Kumari, and P. Sharma, "Estimation of Software Development Effort: A Differential Evolution Approach," *Procedia Comput. Sci.*, vol. 167, pp. 2643–2652, 2020, doi: 10.1016/j.procs.2020.03.343.
[37] B. Mahesh, "Machine Learning Algorithms - A Review," *Int. J. Sci. Res.*, vol. 9, no. 1, pp. 381–386, 2020, doi: 10.21275/ART20203995.
[38] C. López-Martín, "Predictive accuracy comparison between neural networks and statistical regression for development effort of software projects," *Appl. Soft Comput.*, vol. 27, pp. 434–449, Feb. 2015, doi: 10.1016/j.asoc.2014.10.033.
[39] P. Zhang, B. Ren, H. Dong, and Q. Dai, "CAGFuzz: Coverage-Guided Adversarial Generative Fuzzing Testing for Image-Based Deep Learning Systems," *IEEE Trans. Softw. Eng.*, vol. 48, no. 11, pp. 4630–4646, Nov. 2022, doi: 10.1109/TSE.2021.3124006.
[40] N. Pandey, D. K. Sanyal, A. Hudait, and A. Sen, "Automated classification of software issue reports using machine learning techniques: an empirical study," *Innov. Syst. Softw. Eng.*, vol. 13, no. 4, pp. 279–297, Dec. 2017, doi: 10.1007/s11334-017-0294-1.
[41] M. D. Ali and A. A. Abusnaina, "Classifying bug reports to bugs and other requests: an approach using topic modelling and fuzzy set theory," *Int. J. Adv. Comput. Res.*, vol. 11, no. 56, p. 103, Sep. 2021, doi: 10.19101/IJACR.2021.1152031.
[42] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Inf. Sci. (Ny).*, vol. 259, pp. 571–595, Feb. 2014, doi: 10.1016/j.ins.2010.12.016.
[43] F. O. Aghware *et al.*, "Enhancing the Random Forest Model via Synthetic Minority Oversampling Technique for Credit-Card Fraud Detection," *J. Comput. Theor. Appl.*, vol. 1, no. 4, pp. 407–420, Mar. 2024, doi: 10.62411/jcta.10323.
[44] J. T. Iorzua, D. K. Kwaghtyo, T. P. Hule, A. T. Ibrahim, and A. D. Nongu, "AI-Driven Approach to Crop Recommendation: Tackling Class Imbalance and Feature Selection in Precision Agriculture," *J. Futur. Artif. Intell. Technol.*, vol. 2, no. 2, pp. 269–281, Jul. 2025, doi: 10.62411/faith.3048-3719-118.
[45] D. R. I. M. Setiadi, K. Nugroho, A. R. Muslikh, S. W. Irianandа, and A. A. Ojugo, "Integrating SMOTE-Tomek and Fusion Learning with XGBoost Meta-Learner for Robust Diabetes Recognition," *J. Futur. Artif. Intell. Technol.*, vol. 1, no. 1, pp. 23–38, May 2024, doi: 10.62411/faith.2024-11.