SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V12I1P104

Volume 12 Issue 1, 44-71, January 2025

Original Article

An Efficient CoRXGB Approach to Estimate Effort of Scrum Projects

Shivali Chopra[1], Arun Malik[2]

[1,2]*Lovely Professional University, Punjab, India.*

[1]*Corresponding Author : shivalichopra100@gmail.com*

Received: 08 November 2024 Revised: 14 December 2024 Accepted: 05 January 2025 Published: 30 January 2025

Abstract *- In agile software development, Story Point Estimation (SPE) is at the core of project planning, resource planning, and project timeline management. During the last ten years, many researchers have reasonably attempted to propose methods for estimating story points or tasks in agile projects. Expert judgment, planning poker, and analogy are some traditional approaches that have been widely applied but criticized because of the inherent subjectivity, vulnerability to biases, and inability to handle intrinsic complexities in user stories. Eventually, these lead to inaccurate estimates, misaligned stakeholder expectations, and suboptimal sprint outcomes. The research direction has also shifted more in recent times to machine learning-based and deep learning-based approaches that try to present more systematic estimation models driven by the data itself. However, these also face difficulties while fully capturing the nuances involved with the multifaceted nature of user stories. This paper proposes a new hybrid model for software effort estimation entitled CoRXGB. This will help through synergistically combine CNN, RNN, and XGBoost and take the strength of all: CNN for extracting contextual and textual features, the Bi-LSTM for extracting sequential and temporal relations, and XGBoost is superior at classifications. Among the key originalities of the research approach, the most important may become the strategy for hyperparameter optimization that involves Bayesian Optimization integrated with the Learning Gain Matrix. This strategy thus systematically analyzes and optimizes performance gains from various configurations of hyperparameters and hence effectively removes inefficiencies that are associated traditionally with the tuning process. This indeed lets one make better-informed and selective adjustments in reaching high performance. Then, the resultant CoRXGB model has been applied extensively to a wide array of data sourced from different Agile projects that included user stories amounting to more than 23,000 stories. The results showed a significant improvement in the accuracy after hyperparameter tuning, with Appcelerator Studio increasing its accuracy to 90.55% from 82.47% and Aptana Studio increasing from 82.57% to 90.82%, reflecting an increase of 6.44%. In different data sets, CoRXGB outperformed traditional classifiers like Logistic Regression, Support Vector Classifier, and K-Nearest Neighbors, and also outperformed advanced models like RNN-CNN and DEEP-SE. These results underpin the efficiency of the CoRXGB model in story point estimation. It not only outperforms baselines by substantial margins in precision, recall, and F1-score but also holds immense promise to improve Agile project planning processes toward more reliable and efficient software development practices.*

Keywords *- Agile, CNN, Deep Learning, Effort Estimation, Machine Learning, RNN, Scrum, XGB.*

1. Introduction

Story points serve as a significant metric for estimating the effort required to implement a user story in agile project management, focusing on its complexity instead of its size [1-2]. Traditional estimation methods, including expert judgement, planning poker, and analogy-based techniques [3], often result in inconsistencies and inaccuracies due to heavy dependencies on the subjective judgement of team members, which impact the project timelines and resource allocation [4]. To overcome these challenges, there is a need for a more objective and reliable technique to estimate story point complexity. Over the past decade, several Machine Learning (ML) [5-13], deep learning [14-16], fuzzy [17-18] and regression techniques [19] have been explored to overcome these limitations and enhance the story point estimation accuracy. Earlier, Abrahamsson et al. (2011) [20] implemented regression models and Support Vector Machines using priority ad-specific keywords as features which were extracted from user stories. The SVM models show significant potential to address human bias and errors in manual estimations. Porru et al. (2016) [21] further expanded this approach using textual features and metadata taken from Jira issue reports to classify user stories, confirming the importance of user story text and length as key predictors for estimating story points. Moving further, developer-related features (e.g., developer reputation and

workload) alongside textual features were included by Scott and Pfahl (2018) [22] for the estimation of story points using SVM models, which outperformed traditional textual analysis. The incorporation of deep learning resulted in the development of models like Deep-SE (Choetkiertikul et al., 2019) [23] and GPT2SP [24]. Deep-SE leverages word embeddings and deep learning architectures to map raw user story text into a representation that aids story point estimation.

The CoRXGB model performs better than various other algorithms like TF/IDF, SVM, etc. The model can work on the semantic relationships between user stories. A transformer-based model, GPT2SP, has also been developed in recent times, which offers improvements in cross-project scenarios. However, the existing methods still face challenges in capturing user stories' intricate and sequential nature. These research gaps have been addressed by CoRXGB, combining CNN, RNN and XGBoost [25]. The CNN layer extracts features from user stories by detecting local patterns and identifying key phrases and combinations of words frequently associated with specific story points. This layer captures the spatial hierarchies in data, deriving higher-level features from lower-level ones. The output from the CNN is fed into the RNN layer, which is used for processing sequences and, therefore, handles the contextual nature of user stories. The RNN layer understands the complexity embedded in the sequence of words and preserves the context of the story points. Finally, the extracted features are sent to an XGBoost classifier, which utilizes these refined features to predict story points.

The CorXGB model incorporates an innovative hyperparameter tuning strategy that combines Bayesian Optimization [26] with a Learning Gain Matrix (LGM). This approach captures performance gains while exploring the hyperparameter space [27]. The proposed CoRXGB model was trained and evaluated on a dataset containing more than 23000 user stories taken from open-source projects and repositories. This performance demonstrates the model's efficacy in delivering a consistent estimation process. It also aids in better sprint planning and resource management. Section 2 reviews related work in story point estimation, highlighting current trends and gaps in the field. Section 3 details the methodology of the CoRXGB model. Section 4 depicts the setup, datasets, and performance metrics. Section 5 presents the results, comparing the CoRXGB model with existing approaches. Section 6 outlines the conclusion and future research.

2. Related Work

The estimation of SPs in Agile software development has increasingly drawn attention from researchers due to the dynamic nature of Agile projects, where the estimation of effort with good accuracy. The current research landscape of Agile story point estimation indeed reflects a considerable evolution in methodologies, focusing on enhancing estimation accuracy with new machine learning and NLP techniques [33-34]. Vali Towasi et al. (2022) [35] extended the DEEP-SE framework by incorporating a larger dataset from open-source projects. However, the expanded dataset did not significantly improve estimation accuracy, which may indicate that greater volume does not necessarily lead to higher-quality predictions in this context.

On the other hand, Hung Phan et al. (2022) [36] applied the Text-Level Graph Neural Network with an accuracy of 80%. Due to the above approach beating or attaining traditional approaches such as TF-IDF [37], its demonstration has opened up newer sets with a greater potential GNN and in extracting complex associations presented between user stories than most previous traditional methods may stand in the way. Michael Fu et al. applied (Deep) transformer models in support of an Agile story point estimates' implementation and were certain about accuracy improvements with estimations of 6 percent up to 47 percent on the DEEP-SE method. This also underscores a significant need that already leading-edge deep learning methodologies could play in estimating story points.

SLRs also contribute to this domain. Indra Kharisma Raharjana et al. (2021) [40] performed an SLR on user stories and NLP, analyzing 38 studies out of a pool of 718 papers published between 2009 and 2020. Their work provides an overall overview of the intersection between NLP and user story analysis, highlighting trends and gaps in the literature. Julliano Trindade Pintas et al. (2021) [41] presented a related SLR on feature selection methods in text classification; hence, effective feature selection must also be performed to achieve better performance. Morakot Choetkiertikul et al. (2019) [23] presented a model of LSTM [42] networks with RHNs for story point prediction. Their end-to-end DEEP-SE system, training raw data without any feature engineering, tended to outperform methods such as Doc2Vec and Bag-ofWords.

Most recent studies have combined deep learning models with other machine learning approaches, such as SentenceBERT with gradient-boosted trees. The work of Burcu Yalçıner et al. (2024) [43] is a very good example of this. Their model outperformed the best-performing state-of-theart models by about 18%, demonstrating how effectively DLbased feature extraction works coupled with advanced ML algorithms. These studies together point to the direction that Agile story point estimation research has taken so far—from using traditional NLP techniques to more advanced deep learning models. The challenge is how these models are further refined, which can achieve high accuracy and be interpretable and practical in real-world Agile environments. A summary is provided in Table 1.

Table 1. Summary of the related work

Author(s)	Year	Methodology	Key contributions
Burcu Yalçıner et al. [43]	2024	Deep Learning, Machine Learning, NLP (SBERT, Gradient Boosting)	Introduces an integrated approach using SBERT and GBT, improving story point estimation by 18% over state-of-the-art models. Addresses subjectivity and variability in traditional methods.
Sánchez, Santacruz [44]	2023	Autoencoders, Deep Neural Networks, Ensemble Learning	Combines traditional Agile estimation with advanced ML models to improve estimation accuracy.
Younisse, Azzeh [45]	2023	NLP and Machine Learning	Surveys the application of NLP in Agile SPE, highlighting the benefits of integrating ML and DL techniques for more accurate estimations.
Vali Towasi et al. [35]	2022	Deep Learning (DEEP-SE Framework)	Extended DEEP-SE framework with a dataset of thousands of user stories. Found that merely increasing dataset size does not necessarily improve estimation accuracy.
Hung Phan et al. [36]	2022	Graph Neural Network (GNN)	Used Text-Level GNN to estimate story points, achieving 80% accuracy. Demonstrated the superiority of GNNs in capturing complex relationships within user stories over traditional methods.
Michael Fu et al. [24]	2022	Transformer Models (GPT2SP)	Applied transformer models to Agile story point estimation, significantly improving accuracy (6% to 47%) over DEEP-SE. Highlighted the potential of advanced deep learning architectures.
Haithem Kassem et al. [46]	2022	Hierarchical Attention Neural Network (HAN)	Proposed Hierarchical Attention Neural Network (HAN), focusing on the hierarchical nature of textual information in user stories. The authors selected 7 projects out of 16 and formed a new dataset of 7459 issues. The proposed HAN model was tested on this subset, achieving an accuracy of 87%.
Indra Kharisma Raharjana et al. [40]	2021	Systematic Literature Review (SLR)	Conducted an SLR on user stories and NLP, providing an extensive overview of trends and gaps in the intersection of NLP and user story analysis. Reviewed 38 studies out of 718 papers published from 2009 to 2020.
Julliano Trindade Pintas et al. [41]	2021	Systematic Literature Review (SLR)	Conducted an SLR on feature selection methods in text classification, emphasizing the importance of effective feature selection in improving accuracy, including Agile story point estimation.
Morakot Choetkiertikul et al. [23]	2019	Deep Learning	Proposed the DEEP-SE model, an end-to-end system combining LSTM and RHN. Trained on raw data without feature engineering, outperforming methods like Doc2Vec and Bag-of-Words.
Porru et al. [21]	2018	Textual Features, Metadata, Classification	Confirmed the significance of user story text and length in story point classification, highlighting the feasibility of automated tools in Agile estimation.
Abrahamsson et al. [20]	2011	Regression Models and Neural Networks (NN)	Explored various models for story point estimation, finding SVM to be the most effective in reducing human bias in manual estimation.

3. Methodology

The CoRXGB model is a hybrid deep learning model that integrates CNN, RNN (Bi-LSTM), and XGB to classify story points based on the complexity of user stories. It predicts the complexity of the given user story based on historical data of previously estimated tasks. Our model classifies the user stories into 3 categories – easy, medium and complex. The steps of the proposed research methodology are given below.

3.1. Data Preprocessing

- Data Loading: Load the dataset containing user stories, their corresponding descriptions, and assigned story points.
- Class Balancing: Group the user stories into three classes—easy, medium, and complex—to manage class imbalance in the dataset.
- Feature Creation: Concatenate the attributes of each user story to form an input feature representing the textual data.
- Data Cleaning: The stop words, HTML tags, and null values have been removed from the user stories.
- Text Normalization: A Porter stemmer has been used to reduce dimensionality.
- Tokenization: A Count Vectorizer is used on cleaned user stories, which further splits the text into tokens.
- TF-IDF Vectorization: A TF-IDF is used to convert the textual information into a numerical format. It captures the relevance of words in the user stories.
- Padding: It has been ensured that the input sequences are of uniform length for better model training and that padding is used for this task.
- Normalization: A standard scaler has been used to standardize numerical features. It ensured consistency in scaling across all data points.

3.2. CoRXGB Model Architecture

- Input Layer: In this layer, the shape of the TF-IDF vector and padded sequences have been taken together.
- Embedding Layer: This layer captures the semantic meaning of the user stories. It transforms the word indices into dense vectors of a fixed size.

  It is defined with input dimensions corresponding to the vocabulary size and output dimensions suitable for the subsequent convolutional layer.
- Convolutional Layer: This layer incorporates a 1D Convolutional Layer from user stories. It is defined in Equation (1) with input size (N, C_{in}, L_{in}), output size (N, C_{out}, L_{out}), and ReLU activation function using Equation (2):

$$out\left(N_i, C_{out_j}\right) = bias\left(C_{out_j}\right) + \sum_{k=0}^{C_{in}-1} weight\left(C_{out_j}, k\right) \star input(N_i, k) \quad (1)$$

$$f(x) = max(0, x) \quad (2)$$

The output size of the Convolution 1D Layer can be calculated using Equation (3):

$$L_{out} = \left\lfloor \frac{L_{in} - K + 2*P}{S} + 1 \right\rfloor \quad (3)$$

- Max-Pooling Layer: Use a MaxPooling1D layer to simplify the model and reduce computational complexity. Define the Max Pooling 1D Layer using Equation (4) with input size (N, C_{in}, L_{in}) and output size (N, C_{out}, L_{out}):

$$out(N_i, C_j, k) = max_{m=0,\ldots,K-1} input(N_i, C_j, S * k + m) \quad (4)$$

The output size of the Max Pooling 1D Layer can be calculated using Equation (4):

- Bidirectional LSTM Layers: Add Bidirectional LSTM layers to capture long-term dependencies, increasing the model's understanding of the context within the user stories. Define the Bidirectional LSTM Layer using Equations (5) and (6)

$$\overrightarrow{h_t} = LSTM(x_t, \overrightarrow{h_{t-1}}) \quad (5)$$

$$\overleftarrow{h_t} = LSTM(x_t, \overleftarrow{h_{t+1}}) \quad (6)$$

Each cell in the LSTM is calculated using Equations (7)-(12):

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \quad (7)$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \quad (8)$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \quad (9)$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \quad (10)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (11)$$

$$h_t = o_t \odot tanh(c_t) \quad (12)$$

- Dropout Layer: Apply the Dropout layer using Equation (13):

$$Dropout(x) = \begin{cases} \frac{x_i}{1-p}, with\ probability\ 1-p \\ 0, with\ probability\ p \end{cases} \quad (13)$$

- Dense Layer: Introduce a dense layer with ReLU activation followed by batch normalization to improve non-linearity and stabilize the training process. Define the Dense layer using Equation (14) with ReLU activation function Equation (2):

$$z_t = f(\Sigma w_i x_i + b) \quad (14)$$

Where w_i is the weight of neuron i, x_i is the input to neuron i and b is the bias. Also, Define the Batch Normalization layer using Eq. (15):

$$y_i = \gamma\left(\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}\right) + \beta \quad (15)$$

Where x_i is the input feature, μ_B is the mean of the input features, σ_B^2 is the variance of the input features, ε is a small constant, and γ & β are the scalable parameters for selecting and shifting the normalized input.

- XGBoost Layer: Integrate an XGBoost classification layer to utilize the extracted features from the CNN-Bi-LSTM part of the model for final classification. Define the XGBoost classifier using Eq. (16) – Eq. (18):

$$\hat{y} = \sum_{k=1}^{K} f_k(x) \quad (16)$$

Where $\hat{y}$ is the predicted class, K is the number of trees, f_k is the k[th] tree in the ensemble, which is part of the model, and x is the input feature vector.

Each tree f_k is defined as follows –

$$f_k(x) = w_{q(x)} \quad (17)$$

Where $q(x)$ is a function that assigns the input x to one of the leaves and $w_{q(x)}$ is the weight of the leaf to which x is assigned.

The objective function for the XGBoost is defined as follows:

$$\text{Obj}(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \Omega(f_k) \quad (18)$$

Where l is the multiclass cross-entropy loss, and $\Omega(f_k)$ is the regularization term for the k[th] tree, which controls the complexity of the model.

3.3. Model Training and Configuration

- Data Splitting: Divide the dataset into training, validation, and testing sets. Stratified k-fold cross-validation has also been used.
- Model Compilation: Compile the CoRXGB model with sparse categorical cross-entropy and an ADAM optimizer.
- Training: Train the model using the training set, evaluating performance on the validation set to tune parameters and prevent overfitting.

3.4. Hyperparameter Tuning via Bayesian Optimization

- Hyperparameter Selection: Identify key hyperparameters (e.g., learning rate, number of layers, and XGBoost-specific parameters) for optimization.
- Objective Function: Define the objective function to maximize accuracy while also taking into account the Learning Gain Matrix (LGM) to ensure model improvements.
- Optimization Process: Use Bayesian Optimization to find the optimal hyperparameter settings over multiple iterations.
- Learning Gain Matrix (LGM) Evaluation: Apply LGM to evaluate how changes in hyperparameters affect model performance, optimizing for both accuracy and efficiency.

3.5. Model Evaluation

- Performance Metrics: Evaluate the CoRXGB model using evaluation metrics across all Agile projects.
- Confusion Matrix and ROC Curve: Construct confusion matrices and AUC-ROC curves before and after hyperparameter tuning to assess improvements in classification.
- Gain Analysis: Plot line graphs showing the relationship between CoRXGB gains and score over iterations for each Agile project, highlighting areas of improvement.
- Hyperparameter Impact Visualization: Visualize the effects of various hyperparameters using scatter matrices, heat maps, and bar charts to understand their influence on the model's performance.
- Stability Assessment: Use radar graphs to assess the stability of the model across multiple iterations, ensuring that the model remains robust.
- Statistical Comparison: Perform statistical hypothesis testing to compare CoRXGB's performance against

other baseline classifiers, ensuring that improvements are statistically significant.

- Comparative Analysis: Conduct a comparative analysis of CoRXGB to showcase its effectiveness in the context of Agile software development.

The flowchart of the CoRXGB model is given in Figure 1.

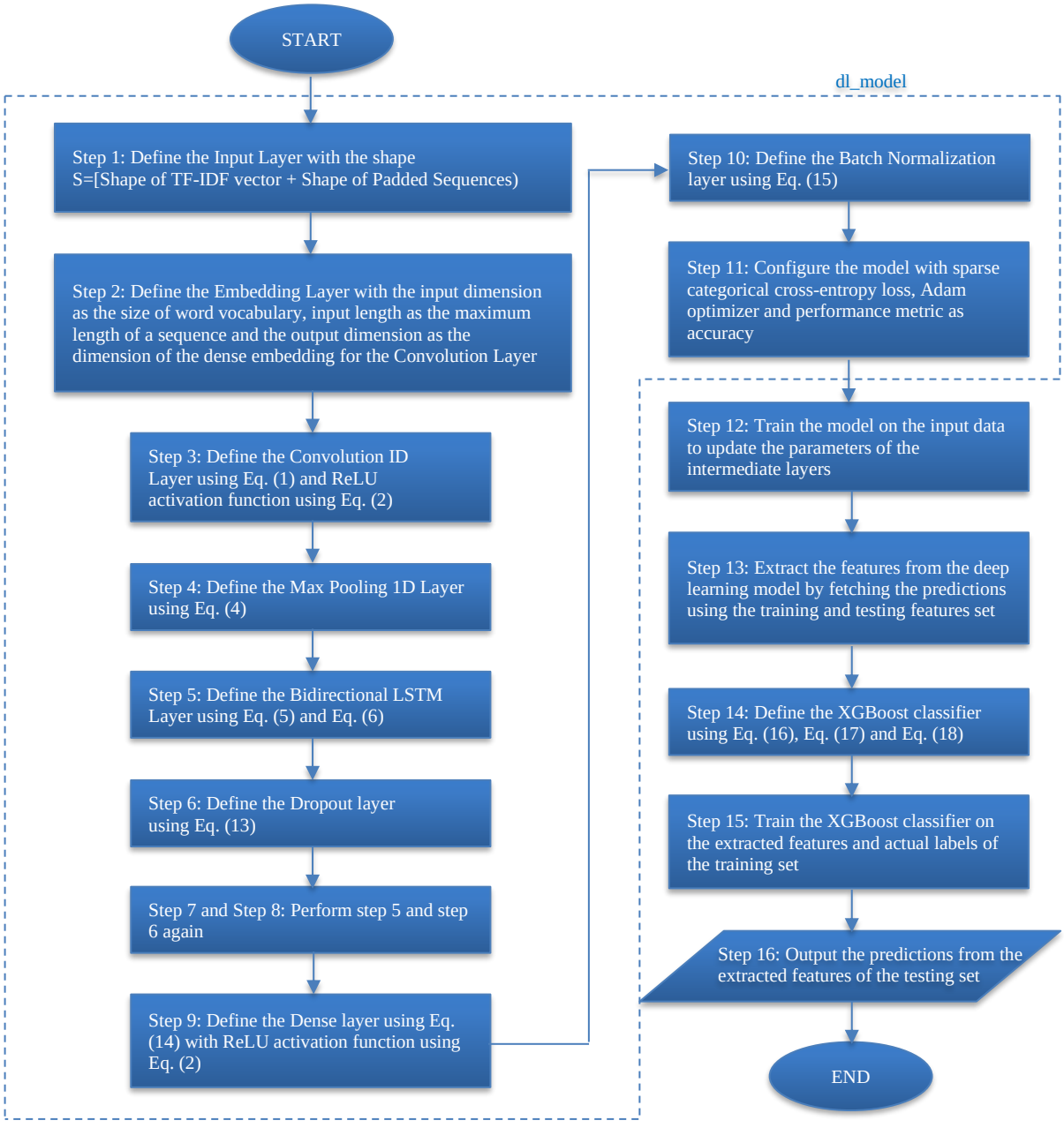

Fig. 1 Flowchart of CoRXGB

4. Results and Discussions

4.1. The Dataset

The CoRXGB model has taken datasets from various open-source projects [23]. A typical issue consists of an issue key, issue title, issue description, and actual story points. A total of 23313 issues have been used. The descriptive characteristics of the dataset have been given in Table 2. It contains the name of the open-source repository, open-source project name, number of issues, minimum SPs, maximum SPs, mean SPs, median SPs, mode SPs, SP variance, SP standard deviation, and mean TD length of the project.

Table 2. Descriptive characteristics of the dataset

Repo.	Project	No. of Issues	min SP	max SP	mean SP	median SP	mode SP	var SP	std SP	Mean TD length
Apache	Mesos	1,680	1	40	3.09	3	3	5.87	2.42	181.12
	Usergrid	482	1	8	2.85	3	3	1.97	1.4	108.6
Appcelerator	Appcelerator Studio	2,919	1	40	5.64	5	5	11.07	3.33	124.61
	Aptana Studio	829	1	40	8.02	8	8	35.46	5.95	124.61
	Titanium SDK/CLI	2,251	1	34	6.32	5	5	25.97	5.1	205.9
DuraSpace	DuraCloud	666	1	16	2.13	1	1	4.12	2.03	70.91
Atlassian	Bamboo	521	1	20	2.42	2	1	4.6	2.14	133.28
	Clover	384	1	40	4.59	2	1	42.95	6.55	124.48
	JIRA Software	352	1	20	4.43	3	5	12.35	3.51	114.57
Moodle	Moodle	1,166	1	100	15.54	8	5	468.5	21.6	88.86
Lsstcorp	Data Management	4,667	1	100	9.57	4	1	275.7	16.6	69.41
Mulesoft	Mule	889	1	21	5.08	5	5	12.24	3.5	81.16
	Mule Studio	732	1	34	6.4	5	5	29.01	5.39	70.99
Spring	Spring XD	3,526	1	40	3.7	3	1	10.42	3.23	78.47
Talendforge	Talend Data Quality	1,381	1	40	5.92	5	8	26.96	5.19	104.86
	Talend ESB	868	1	13	2.16	2	1	2.24	1.5	128.97
Total		23313								

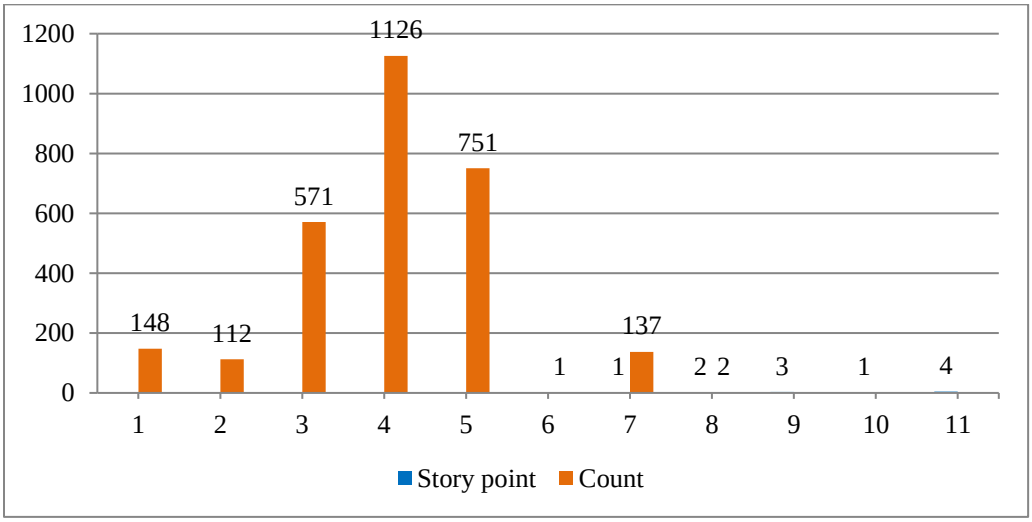

Fig. 2 Distribution of story points in Appcelerator Studio

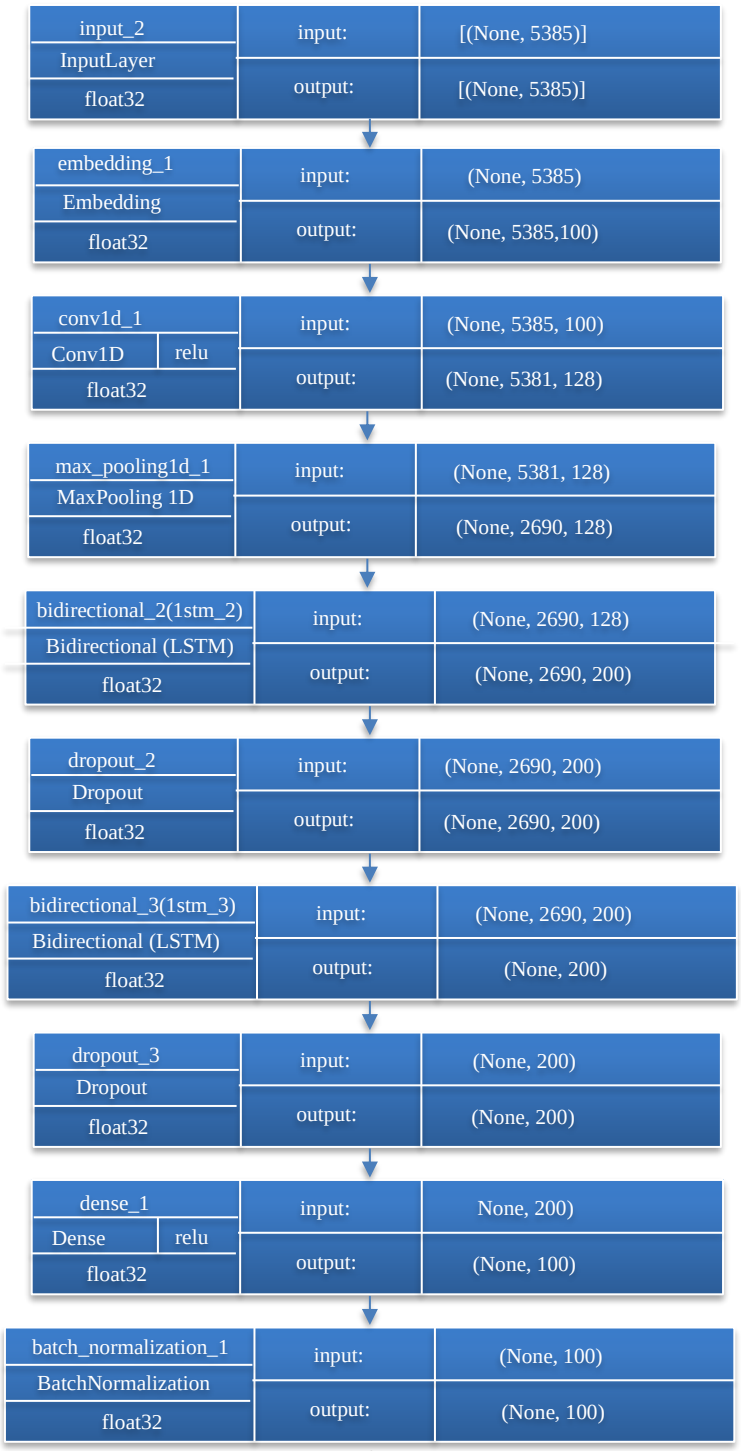

Fig. 3 Architecture of the CoRXGB model

The actual story points in the Appcelerator Studio dataset before pre-processing are given in Figure 2. It can be inferred that the number of occurrences is not uniform throughout the different story point categories. We have resolved the problem of the imbalanced dataset by generating new artificial entries using SMOTE (Synthetic Minority Oversampling Technique) [47] as defined in Equation (19). Before oversampling, the story points have been grouped together into 3 different categories – easy, medium and complex, to reduce the imbalanced data.

$$x_{new} = x_i + \delta . (x_{neighbor} - x_i) \quad (19)$$

Where x_{new} is the synthetic sample, x_i is the original minority class sample, $x_{neighbour}$ is k-nearest neighbors of x_i, δ is the random number between 0 and 1.

Figure 3 illustrates the architecture of the CoRXGB model used for story point classification in Agile projects. The model employs a hybrid deep learning approach, starting with an input layer that accepts the concatenated TF-IDF vector and padded sequence of the user stories. These inputs are then processed through an embedding layer that transforms word indices (with a vocabulary size of 5385) into dense vectors of size 100, capturing the semantic meaning of the text. Following this, a Conv1D layer extracts important local patterns. Dropout layers are introduced between the LSTMs to prevent overfitting during training. A dense layer with 100 neurons applies a non-linear transformation, and a batch normalization layer ensures stable and efficient training by normalizing the input to the dense layer. Although not shown in the figure, the final step involves passing the extracted features to an XGBoost classifier, which performs the classification of user stories into categories such as "easy," "medium," or "complex." This architecture effectively combines the strengths of CNNs for feature extraction and Bidirectional LSTMs for capturing sequential relationships, followed by XGBoost for improved classification performance. Initial Hyperparameter setting of CoRXGB: Table 3 defines the hyperparameters and their values for the CoRXGB model. These hyperparameters are carefully chosen to balance the model's complexity and performance.

Table 3. Initial Hyperparameters setting of the CoRXGB model

Hyperparameter Name	Value
Filters	32
kernel_size	5
Conv 1D Activation (Input)	ReLU
Dropout	0.5
LSTM input	128 neurons
Optimizer	Adam
Number of epochs	10
Early stopping	Monitor – val_accuracy Min_delta = 1e-2 Patience = 5
Loss type	Categorical cross-entropy
min_lr	0.001
No. of folds	5
Seed	250

The hyperparameters selected for the XGB model are given in Table 4.

The specified ranges for these hyperparameters allow the Bayesian Optimization process to explore a wide space of possible model configurations. In the context of story point classification using the CoRXGB algorithm, this helps find an optimal balance between model complexity, accuracy, and generalization capability.

Table 4. Hyperparameters of the XGBoost model

Hyperparameter	Significance
max_depth (MD)	Controls the maximum depth of trees. A range of (3, 10) balances complexity and generalization, capturing intricate dependencies between story attributes without overfitting.
learning_rate(LR)	Controls the step size during each iteration. A range of (0.01, 0.3) ensures subtle learning and avoids aggressive updates, which is essential for precise story point classification.
n_estimators(NE)	Specifies the number of trees. The range (100, 500) ensures accurate classification by capturing data patterns while maintaining training efficiency.
Gamma(G)	Minimum loss reduction for partitioning leaf nodes. The range (0, 0.5) balances, making enough splits for pattern capture while avoiding over-complex models.
Subsample(SS)	Controls the fraction of samples per tree. The range (0.5, 1.0) introduces randomness, improving robustness and reducing overfitting.
colsample_bytree (CB)	Controls the fraction of features used for each tree. A range of (0.5, 1.0) ensures the model captures relevant features without over-reliance on any single attribute.
reg_lambda(RL)	Specifies the L2 regularization term to prevent overfitting. The range (0, 10) applies varying degrees of regularization for better generalization.
reg_alpha(RA)	Specifies the L1 regularization term. A range of (0, 10) helps reduce overfitting, encouraging simpler models that generalize better.

Table 5. CoRXGB performance on the Appcelerator Studio dataset

Class/Metrics	precision	Recall	f1-score	support
Class 0 (Easy)	0.941657	0.95109	0.94635	1697
Class 1 (Medium)	0.755199	0.748969	0.752071	1697
Class 2 (Complex)	0.775679	0.774308	0.774993	1697
macro avg	0.824178	0.824789	0.824471	5091

4.2. Performance Evaluation of CoRXGB on Appcelerator Studio Project (without Hyperparameter Tuning)

The evaluation of the "Appcelerator Studio" project was conducted using four algorithms: CoRXGB, KNN, Logistic Regression (LR), and Support Vector Classifier (SVCLd). This analys is included several performance metrics. Figure 4 illustrates the confusion matrix of the CoRXGB model, which performed exceptionally well in classifying user stories into Easy, Medium, and Complex categories. The model's overall classification accuracy on the Appcelerator Studio dataset was 83%, as summarized in Table 5 for each class.

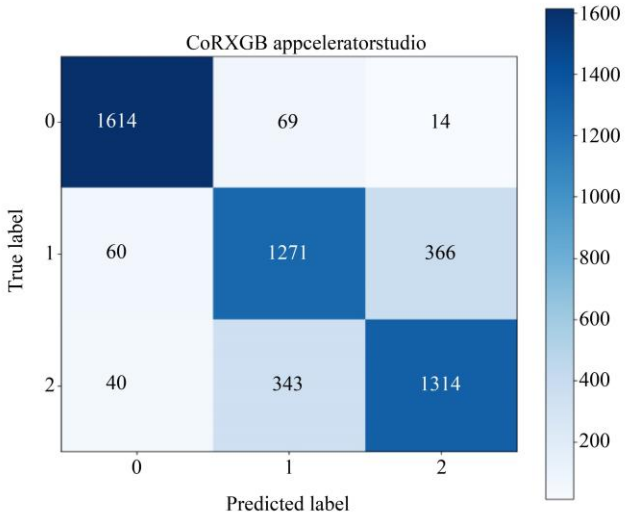

Fig. 4 Confusion matrix of CoRXGB on Appcelerator Studio

4.3. Training and Validation Accuracy

The CoRXGB algorithm demonstrates high accuracy across training and validation phases. The slight drop in validation accuracy is due to the nature of the US, but overall, the model performs well on the Appcelerator Studio dataset. This high performance across multiple phases indicates a robust model with good generalization capabilities. The "loss plot" in Figure 5 shows the training and validation loss.

The training loss and the validation loss decreased, showing significant improvement in both cases. The gap between training and validation loss is relatively small, indicating good generalization performance. Table 6 shows the accuracy comparison of the CoRXGB for the appcelerator studio project.

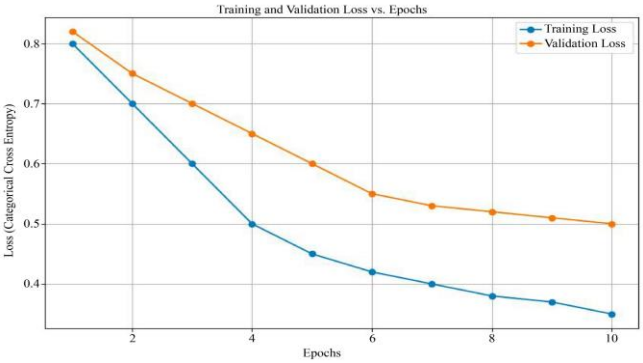

Fig. 5 Training and validation loss vs epochs plot of CoRXGB on appcelerator studio project

Table 6. CoRXGB accuracy comparison for Appcelerator studio

Metric	Accuracy
Training Accuracy	0.905
Validation Accuracy	0.845

4.4. ROC Curve for CoRXGB Model on Appcelerator Studio Dataset

The ROC curve for the CoRXGB model on the Appcelerator Studio dataset in Figure 6 indicates strong classification performance across all three classes: Easy, Medium, and Complex. The curve for Class 0 (Easy) is the most prominent. Class 1 (Medium) also shows good performance but with slightly lower accuracy compared to Class 0. Class 2 (Complex) has the lowest curve among the three, indicating relatively more challenges in accurately classifying complex tasks. The AUC values for the appcelerator studio project are given in Table 7.

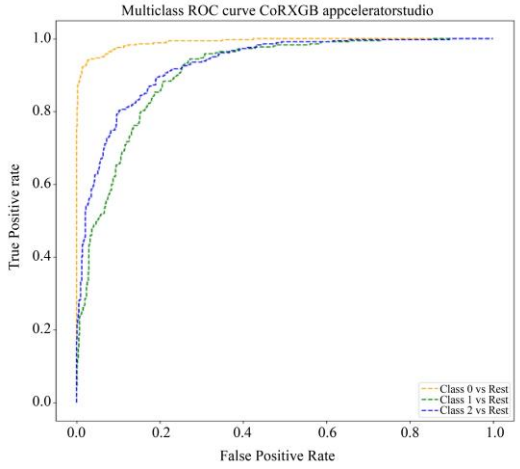

Fig. 6 ROC curve of CoRXGB on appcelerator studio

Table 7. AUC values table for appcelerator studio

Class	AUC value
0 (Easy)	0.98
1 (Medium)	0.90
2 (Complex)	0.87

These AUC values indicate the model's performance, with Class 0 having the highest accuracy.

4.5. Performance Evaluation of CoRXGB across Multiple Projects

A comprehensive analysis of the CoRXGB model's performance across multiple projects has been performed.

4.6. Training, Testing, and Validation Accuracy of CoRXGB across Different Projects

The various accuracies of the CoRXGB model across various projects are given in Table 8 and Figure 7.

It helps in understanding the model's generalizability to new data. It also ensures consistency across different datasets.

4.7. Performance Metrics of CoRXGB across Different Projects

The performance matrices for each class across different projects are given in Table 9.

Table 8. Training, Testing, and Validation Accuracy of CoRXGB across Different Projects

Project	Training Accuracy	Testing accuracy	Validation Accuracy
Appceleratorstudio	0.84	82.47	0.80
Aptanastudio	0.75	82.57	0.80
Bamboo	0.95	89.68	0.85
Clover	0.70	75.26	0.75
Datamanagement	0.74	62.20	0.70
Duracloud	0.95	90.63	0.80
Jirasoftware	0.85	79.22	0.75
Mesos	0.90	73.64	0.70
Moodle	0.65	82.03	0.80
Mule	0.79	65.11	0.70
Mulestudio	0.78	65.31	0.75
Springxd	0.76	67.2	0.70
Talenddataquality	0.70	60	0.65
Talendesb	0.85	87.05	0.85
Titanium	0.80	76.64	0.75
Usergrid	0.85	87.48	0.80

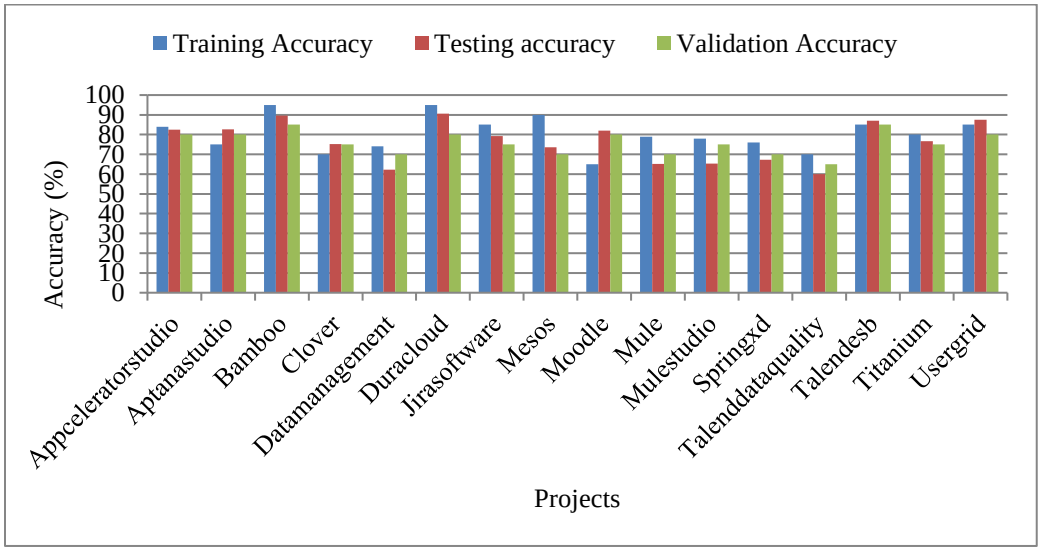

Fig. 7 CoRXGB accuracy comparisons across different projects

Table 9. Performance Metrics of CoRXGB across Different Projects

Projects	Class	Precision	Recall	F1-Score	Support
Appcelerator Studio	0 (Easy)	0.941657	0.95109	0.94635	1697
	1 (Medium)	0.755199	0.748969	0.752071	1697
	2 (Complex)	0.775679	0.774308	0.774993	1697
Aptana Studio	0 (Easy)	0.917927	0.925926	0.921909	459
	1 (Medium)	0.792735	0.808279	0.800431	459
	2 (Complex)	0.764574	0.742919	0.753591	459
Bamboo	0 (Easy)	0.848485	0.845283	0.846881	265
	1 (Medium)	0.843284	0.85283	0.84803	265
	2 (Complex)	1	0.992453	0.996212	265
Clover	0 (Easy)	0.697917	0.705263	0.701571	190
	1 (Medium)	0.676617	0.715789	0.695652	190
	2 (Complex)	0.898305	0.836842	0.866485	190
Moodle	0 (Easy)	0.917957	0.908116	0.91301	653
	1 (Medium)	0.77573	0.773354	0.77454	653
	2 (Complex)	0.768882	0.779479	0.774144	653
Mule	0 (Easy)	0.703349	0.744304	0.723247	395
	1 (Medium)	0.574026	0.559494	0.566667	395
	2 (Complex)	0.662304	0.640506	0.651223	395
Mulestudio	0 (Easy)	0.787466	0.840116	0.81294	344
	1 (Medium)	0.534884	0.534884	0.534884	344
	2 (Complex)	0.626168	0.584302	0.604511	344
SpringXD	0 (Easy)	0.616915	0.632653	0.624685	1372
	1 (Medium)	0.547282	0.535714	0.541436	1372
	2 (Complex)	0.848463	0.844752	0.846603	1372
Data Management	0 (Easy)	0.609394	0.63654	0.622671	1549
	1 (Medium)	0.621142	0.610717	0.615885	1549
	2 (Complex)	0.636122	0.618464	0.627169	1549
Duracloud	0 (Easy)	0.863636	0.869281	0.86645	459
	1 (Medium)	0.872807	0.867102	0.869945	459
	2 (Complex)	0.982571	0.982571	0.982571	459
Jirasoftware	0 (Easy)	0.72956	0.794521	0.760656	146
	1 (Medium)	0.796875	0.69863	0.744526	146
	2 (Complex)	0.854305	0.883562	0.868687	146
Mesos	0 (Easy)	0.640816	0.635628	0.638211	741
	1 (Medium)	0.621871	0.636977	0.629333	741
	2 (Complex)	0.951989	0.936572	0.944218	741
Talenddata quality	0 (Easy)	0.664488	0.682327	0.673289	447
	1 (Medium)	0.49537	0.478747	0.486917	447
	2 (Complex)	0.566667	0.57047	0.568562	447
Talendesb	0 (Easy)	0.80863	0.80863	0.80863	533

	1 (Medium)	0.80705	0.816135	0.811567	533
	2 (Complex)	0.998102	0.986867	0.992453	533
Titanium	0 (Easy)	0.868491	0.879357	0.87389	1119
	1 (Medium)	0.679715	0.682752	0.68123	1119
	2 (Complex)	0.75	0.737265	0.743578	1119
Usergrid	0 (Easy)	0.814516	0.852321	0.83299	237
	1 (Medium)	0.834821	0.78903	0.81128	237
	2 (Complex)	0.974895	0.983122	0.978992	237

4.8. AUC Values of CoRXGB across Different Projects

The AUC values for each class across different projects are given in Table 10. The higher value of the AUC indicates better performance. The classification of the complex user stories showed the best performance across multiple projects. It represents the model's effectiveness and reliability. It also provides a solid foundation for further improvements of CoRXGB in real-world agile project management scenarios.

Training, Testing, and Validation Accuracy: The results show consistency among training, testing, and validation accuracies. Here are some observations:

- Bamboo and Duracloud: These projects show high training and testing accuracies, depicting the good performance of the model on these datasets.
- Talend ESB and Usergrid: These projects also show balanced training and validation accuracies along with testing accuracies, depicting the model's reliability.
- Data management, Mule, Mulestudio, and SpringXD: These projects show lower testing accuracies, depicting a margin of improvements.

Table 10. AUC values of CoRXGB across different projects

Project	Class 0 AUC	Class 1 AUC	Class 2 AUC
Appceleratorstudio	0.98	0.9	0.87
Aptanastudio	0.97	0.88	0.84
Bamboo	0.99	0.92	0.85
Clover	0.98	0.9	0.92
Datamanagement	0.92	0.87	0.89
Duracloud	0.96	0.91	0.94
Jirasoftware	0.95	0.85	0.89
Mesos	0.97	0.88	0.93
Moodle	0.96	0.87	0.92
Mule	0.92	0.84	0.88
Mulestudio	0.9	0.83	0.86
Springxd	0.92	0.86	0.91
Talenddataquality	0.9	0.84	0.88
Talendesb	0.92	0.88	0.94
Titanium	0.94	0.89	0.91
Usergrid	0.99	0.96	0.98

Performance Metrics (Precision, Recall, F1-Score): The performance metrics across various projects show that CoRXGB consistently performs well, particularly for complex user stories:

- High Performance for Complex Stories: The model achieves high precision and recall values for complex stories across most projects, indicating its strong capability to handle the most challenging classifications.
- Medium and Easy Stories: While the model generally performs well for medium and easy stories, some projects show a slight drop in performance for medium complexity (e.g., Mesos and Talend Data Quality).

AUC Values: The AUC values across projects highlight the model's excellent ability to distinguish between different classes:

- High AUC for Complex Stories: Projects such as Duracloud, Talend ESB, and Usergrid exhibit very high AUC values for complex stories, demonstrating the model's superior discrimination capability in identifying these cases.
- Balanced Performance: Even for projects with relatively lower AUC values (e.g., Talend Data Quality and Mule), the model still maintains a reasonable level of performance, suggesting its overall robustness.

4.9. Comparative Analysis of CoRXGB and Other Classifiers

The CoRXGB model demonstrates robust performance across various projects when compared to other classifiers like KNN, LR, and SVCLd. The comparison is given in Table 11. The performance metrics indicate that CoRXGB excels, particularly in classifying complex user stories. An accuracy comparison with DEEP-SE [23] and RNN-CNN [15] is given in Table 12.

Table 11. Comparative Performance Metrics of CoRXGB and Other Classifiers across Different Projects

Projects	Algorithm	Class	Precision	Recall	F1-Score	Support
Appcelerator Studio	CoRXGB	0 (Easy)	0.941657	0.95109	0.94635	1697
		1 (Medium)	0.755199	0.748969	0.752071	1697
		2 (Complex)	0.775679	0.774308	0.774993	1697
	KNN	0 (Easy)	0.612845	0.995286	0.75859	1697
		1 (Medium)	0.737255	0.221567	0.340734	1697
		2 (Complex)	0.696438	0.748969	0.721749	1697
	LR	0 (Easy)	0.846071	0.926341	0.884388	1697
		1 (Medium)	0.702561	0.630524	0.664596	1697
		2 (Complex)	0.72807	0.733648	0.730848	1697
	SVCLd	0 (Easy)	0.875327	0.984679	0.926789	1697
		1 (Medium)	0.773707	0.634649	0.697313	1697
		2 (Complex)	0.751397	0.792575	0.771437	1697
Aptana Studio	CoRXGB	0 (Easy)	0.917927	0.925926	0.921909	459
		1 (Medium)	0.792735	0.808279	0.800431	459
		2 (Complex)	0.764574	0.742919	0.753591	459
	KNN	0 (Easy)	0.550602	0.995643	0.709077	459
		1 (Medium)	0.696325	0.784314	0.737705	459
		2 (Complex)	0.8	0.052288	0.09816	459
	LR	0 (Easy)	0.909278	0.960784	0.934322	459
		1 (Medium)	0.792735	0.808279	0.800431	459
		2 (Complex)	0.808962	0.747277	0.776897	459
	SVCLd	0 (Easy)	0.926829	0.993464	0.958991	459
		1 (Medium)	0.793587	0.862745	0.826722	459
		2 (Complex)	0.862694	0.72549	0.788166	459
Bamboo	CoRXGB	0 (Easy)	0.848485	0.845283	0.846881	265
		1 (Medium)	0.843284	0.85283	0.84803	265
		2 (Complex)	1	0.992453	0.996212	265
	KNN	0 (Easy)	1	0.101887	0.184932	265
		1 (Medium)	0.598916	0.833962	0.697161	265

		2 (Complex)	0.66416	1	0.798193	265
	LR	0 (Easy)	0.865854	0.803774	0.833659	265
		1 (Medium)	0.822064	0.871698	0.846154	265
		2 (Complex)	0.988806	1	0.994371	265
	SVCLd	0 (Easy)	0.88843	0.811321	0.848126	265
		1 (Medium)	0.828671	0.89434	0.860254	265
		2 (Complex)	0.992509	1	0.996241	265
Clover	CoRXGB	0 (Easy)	0.697917	0.705263	0.701571	190
		1 (Medium)	0.676617	0.715789	0.695652	190
		2 (Complex)	0.898305	0.836842	0.866485	190
	KNN	0 (Easy)	0.8	0.084211	0.152381	190
		1 (Medium)	0.572	0.752632	0.65	190
		2 (Complex)	0.593333	0.936842	0.726531	190
	LR	0 (Easy)	0.775148	0.689474	0.729805	190
		1 (Medium)	0.730769	0.8	0.763819	190
		2 (Complex)	0.891192	0.905263	0.898172	190
	SVCLd	0 (Easy)	0.828947	0.663158	0.736842	190
		1 (Medium)	0.738318	0.831579	0.782178	190
		2 (Complex)	0.897059	0.963158	0.928934	190
Moodle	CoRXGB	0 (Easy)	0.917957	0.908116	0.91301	653
		1 (Medium)	0.77573	0.773354	0.77454	653
		2 (Complex)	0.768882	0.779479	0.774144	653
	KNN	0 (Easy)	0.621822	0.898928	0.735128	653
		1 (Medium)	0.591837	0.799387	0.68013	653
		2 (Complex)	0.894737	0.182236	0.302799	653
	LR	0 (Easy)	0.863034	0.897397	0.87988	653
		1 (Medium)	0.752656	0.759571	0.756098	653
		2 (Complex)	0.79066	0.751914	0.770801	653
	SVCLd	0 (Easy)	0.887798	0.969372	0.926794	653
		1 (Medium)	0.799127	0.840735	0.819403	653
		2 (Complex)	0.862254	0.738132	0.79538	653
Mule	CoRXGB	0 (Easy)	0.703349	0.744304	0.723247	395
		1 (Medium)	0.574026	0.559494	0.566667	395
		2 (Complex)	0.662304	0.640506	0.651223	395
	KNN	0 (Easy)	0.491228	0.850633	0.622799	395
		1 (Medium)	0.573529	0.098734	0.168467	395
		2 (Complex)	0.600462	0.658228	0.628019	395
	LR	0 (Easy)	0.64657	0.787342	0.710046	395
		1 (Medium)	0.562112	0.458228	0.504881	395

		2 (Complex)	0.691099	0.668354	0.679537	395
	SVCLd	0 (Easy)	0.691824	0.835443	0.756881	395
		1 (Medium)	0.608247	0.448101	0.516035	395
		2 (Complex)	0.671463	0.708861	0.689655	395
Mule Studio	CoRXGB	0 (Easy)	0.787466	0.840116	0.81294	344
		1 (Medium)	0.534884	0.534884	0.534884	344
		2 (Complex)	0.626168	0.584302	0.604511	344
	KNN	0 (Easy)	0.435931	0.959302	0.599455	344
		1 (Medium)	0.769231	0.087209	0.156658	344
		2 (Complex)	0.690678	0.473837	0.562069	344
	LR	0 (Easy)	0.769231	0.843023	0.804438	344
		1 (Medium)	0.60066	0.52907	0.562597	344
		2 (Complex)	0.613636	0.627907	0.62069	344
	SVCLd	0 (Easy)	0.79198	0.918605	0.850606	344
		1 (Medium)	0.619863	0.526163	0.569182	344
		2 (Complex)	0.639296	0.633721	0.636496	344
SpringXD	CoRXGB	0 (Easy)	0.616915	0.632653	0.624685	1372
		1 (Medium)	0.547282	0.535714	0.541436	1372
		2 (Complex)	0.848463	0.844752	0.846603	1372
	KNN	0 (Easy)	0.573293	0.416181	0.482264	1372
		1 (Medium)	0.576923	0.153061	0.241935	1372
		2 (Complex)	0.458636	0.921283	0.612403	1372
	LR	0 (Easy)	0.622493	0.633382	0.62789	1372
		1 (Medium)	0.539405	0.44898	0.490056	1372
		2 (Complex)	0.752852	0.865889	0.805424	1372
	SVCLd	0 (Easy)	0.629078	0.646501	0.637671	1372
		1 (Medium)	0.569982	0.451166	0.503662	1372
		2 (Complex)	0.784568	0.926385	0.849599	1372
Data Management	CoRXGB	0 (Easy)	0.609394	0.63654	0.622671	1549
		1 (Medium)	0.621142	0.610717	0.615885	1549
		2 (Complex)	0.636122	0.618464	0.627169	1549
	KNN	0 (Easy)	0.55609	0.224015	0.319374	1549
		1 (Medium)	0.400238	0.868948	0.548046	1549
		2 (Complex)	0.719697	0.306649	0.430059	1549
	LR	0 (Easy)	0.645266	0.64235	0.643805	1549
		1 (Medium)	0.589454	0.606198	0.597708	1549
		2 (Complex)	0.659392	0.643641	0.651421	1549
	SVCLd	0 (Easy)	0.661806	0.615236	0.637671	1549
		1 (Medium)	0.616494	0.690123	0.651234	1549

		2 (Complex)	0.663951	0.631375	0.647253	1549
Duracloud	CoRXGB	0 (Easy)	0.863636	0.869281	0.86645	459
		1 (Medium)	0.872807	0.867102	0.869945	459
		2 (Complex)	0.982571	0.982571	0.982571	459
	KNN	0 (Easy)	0.578313	0.104575	0.177122	459
		1 (Medium)	0.633284	0.936819	0.755712	459
		2 (Complex)	0.708943	0.949891	0.811918	459
	LR	0 (Easy)	0.90799	0.816993	0.860092	459
		1 (Medium)	0.84413	0.908497	0.875131	459
		2 (Complex)	0.976596	1	0.988159	459
	SVCLd	0 (Easy)	0.961735	0.821351	0.886016	459
		1 (Medium)	0.853282	0.962963	0.904811	459
		2 (Complex)	0.982869	1	0.991361	459
Jirasoftware	CoRXGB	0 (Easy)	0.72956	0.794521	0.760656	146
		1 (Medium)	0.796875	0.69863	0.744526	146
		2 (Complex)	0.854305	0.883562	0.868687	146
	KNN	0 (Easy)	0.596154	0.849315	0.700565	146
		1 (Medium)	1	0.089041	0.163522	146
		2 (Complex)	0.640553	0.952055	0.76584	146
	LR	0 (Easy)	0.759259	0.842466	0.798701	146
		1 (Medium)	0.784	0.671233	0.723247	146
		2 (Complex)	0.89404	0.924658	0.909091	146
	SVCLd	0 (Easy)	0.79375	0.869863	0.830065	146
		1 (Medium)	0.803279	0.671233	0.731343	146
		2 (Complex)	0.884615	0.945205	0.913907	146
Mesos	CoRXGB	0 (Easy)	0.640816	0.635628	0.638211	741
		1 (Medium)	0.621871	0.636977	0.629333	741
		2 (Complex)	0.951989	0.936572	0.944218	741
	KNN	0 (Easy)	0.616725	0.238866	0.344358	741
		1 (Medium)	0.641791	0.232119	0.340932	741
		2 (Complex)	0.443046	0.997301	0.613533	741
	LR	0 (Easy)	0.659091	0.665317	0.662189	741
		1 (Medium)	0.655063	0.558704	0.603059	741
		2 (Complex)	0.856465	0.974359	0.911616	741
	SVCLd	0 (Easy)	0.663877	0.642375	0.652949	741
		1 (Medium)	0.656848	0.601889	0.628169	741
		2 (Complex)	0.8948	0.99865	0.943878	741
	CoRXGB	0 (Easy)	0.664488	0.682327	0.673289	447
		1 (Medium)	0.49537	0.478747	0.486917	447

Talenddata quality		2 (Complex)	0.566667	0.57047	0.568562	447
	KNN	0 (Easy)	0.395604	0.805369	0.530582	447
		1 (Medium)	0.424354	0.257271	0.320334	447
		2 (Complex)	0.45	0.161074	0.237232	447
	LR	0 (Easy)	0.634656	0.680089	0.656587	447
		1 (Medium)	0.476998	0.440716	0.45814	447
		2 (Complex)	0.550111	0.552573	0.551339	447
	SVCLd	0 (Easy)	0.654064	0.774049	0.709016	447
		1 (Medium)	0.529412	0.483221	0.505263	447
		2 (Complex)	0.576733	0.521253	0.547591	447
Talendesb	CoRXGB	0 (Easy)	0.80863	0.80863	0.80863	533
		1 (Medium)	0.80705	0.816135	0.811567	533
		2 (Complex)	0.998102	0.986867	0.992453	533
	KNN	0 (Easy)	0.653061	0.180113	0.282353	533
		1 (Medium)	0.658015	0.80863	0.725589	533
		2 (Complex)	0.636136	0.95122	0.762406	533
	LR	0 (Easy)	0.816929	0.778612	0.79731	533
		1 (Medium)	0.791289	0.818011	0.804428	533
		2 (Complex)	0.987037	1	0.993476	533
	SVCLd	0 (Easy)	0.876333	0.771107	0.820359	533
		1 (Medium)	0.799663	0.891182	0.842946	533
		2 (Complex)	0.994403	1	0.997194	533
Titanium	CoRXGB	0 (Easy)	0.868491	0.879357	0.87389	1119
		1 (Medium)	0.679715	0.682752	0.68123	1119
		2 (Complex)	0.75	0.737265	0.743578	1119
	KNN	0 (Easy)	0.509416	0.942806	0.661442	1119
		1 (Medium)	0.682609	0.140304	0.232765	1119
		2 (Complex)	0.703598	0.663986	0.683218	1119
	LR	0 (Easy)	0.778302	0.884718	0.828105	1119
		1 (Medium)	0.677149	0.577301	0.623251	1119
		2 (Complex)	0.717065	0.724754	0.720889	1119
	SVCLd	0 (Easy)	0.813777	0.960679	0.881148	1119
		1 (Medium)	0.737209	0.566577	0.640728	1119
		2 (Complex)	0.728741	0.765862	0.746841	1119
Usergrid	CoRXGB	0 (Easy)	0.814516	0.852321	0.83299	237
		1 (Medium)	0.834821	0.78903	0.81128	237
		2 (Complex)	0.974895	0.983122	0.978992	237
	KNN	0 (Easy)	0.632353	0.907173	0.745234	237
		1 (Medium)	0.84	0.088608	0.160305	237

		2 (Complex)	0.684971	1	0.813036	237
	LR	0 (Easy)	0.843882	0.843882	0.843882	237
		1 (Medium)	0.84322	0.839662	0.841438	237
		2 (Complex)	0.995798	1	0.997895	237
	SVCLd	0 (Easy)	0.841509	0.940928	0.888446	237
		1 (Medium)	0.932367	0.814346	0.869369	237
		2 (Complex)	0.991632	1	0.995798	237

Table 12. Accuracy Comparison of CoRXGB with DEEP-SE and RNN-CNN across Various Projects

Projects	Algorithms	Accuracy	Projects	Algorithms	Accuracy
Appcelerator Studio	CoRXGB	82.47	Data Management	CoRXGB	62.2
	RNN-CNN	62.29		RNN-CNN	49.92
	DEEP-SE	60.26		DEEP-SE	47.87
Aptana Studio	CoRXGB	82.57	Duracloud	CoRXGB	90.63
	RNN-CNN	45.62		RNN-CNN	70.94
	DEEP-SE	42.58		DEEP-SE	69.92
Bamboo	CoRXGB	89.68	Jirasoftware	CoRXGB	79.22
	RNN-CNN	74.28		RNN-CNN	59.64
	DEEP-SE	71.24		DEEP-SE	59.52
Clover	CoRXGB	75.26	Mesos	CoRXGB	73.64
	RNN-CNN	50.95		RNN-CNN	59.53
	DEEP-SE	50.45		DEEP-SE	59.84
Moodle	CoRXGB	82.03	Talenddata quality	CoRXGB	60
	RNN-CNN	51.32		RNN-CNN	49.06
	DEEP-SE	50.29		DEEP-SE	48.28
Mule	CoRXGB	65.11	Talendesb	CoRXGB	87.05
	RNN-CNN	43.17		RNN-CNN	70.57
	DEEP-SE	40.09		DEEP-SE	69.67
Mulestudio	CoRXGB	65.31	Titanium	CoRXGB	76.64
	RNN-CNN	18.26		RNN-CNN	58.95
	DEEP-SE	17.17		DEEP-SE	55.92
SpringXD	CoRXGB	67.2	Usergrid	CoRXGB	87.48
	RNN-CNN	46.93		RNN-CNN	55.16
	DEEP-SE	46.82		DEEP-SE	52.66

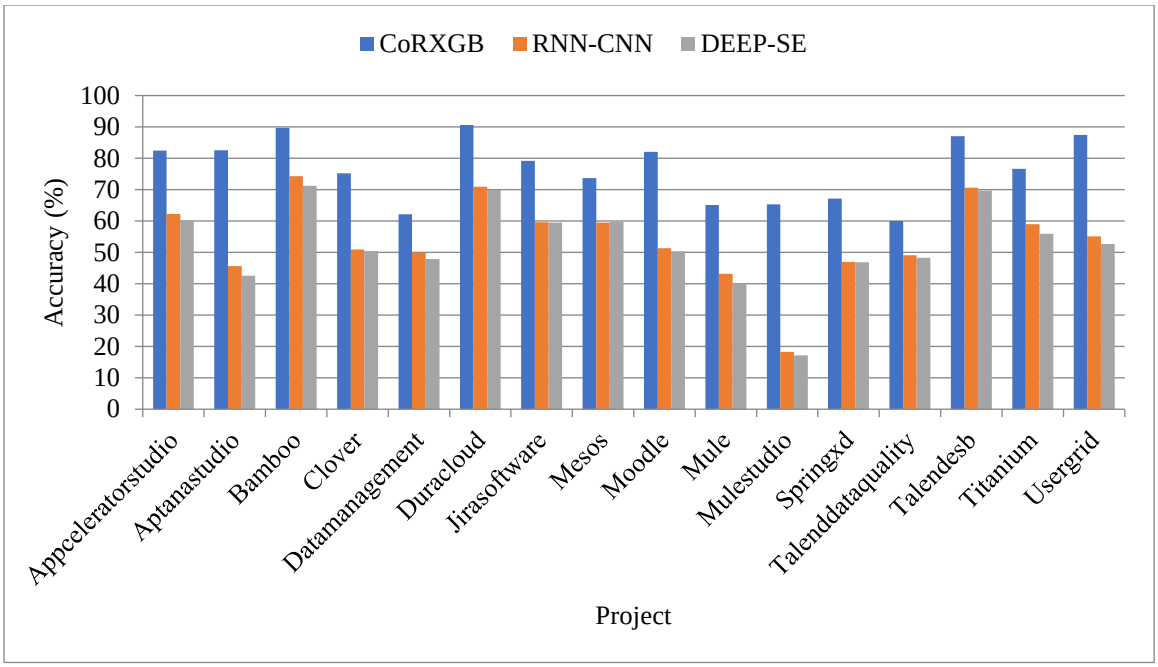

Fig. 8 Accuracy comparison – CoRXGB vs DEEP-SE vs RNN-CNN

Table 13. LGM Bayesian Optimization results for CoRXGB Hyperparameters - Appcelerator Studio Project

max_depth	learning_rate	n_estimators	gamma	subsample	colsample_bytree	reg_lambda	reg_alpha	score	Gain
7	0.22228	162	0.47536	0.9330881	0.6872701	0.58084	1.55995	83.94	0
9	0.01597	432	0.35404	0.5917023	0.8005575	1.81825	2.12339	88.24	5.118814
5	0.13526	344	0.26238	0.6831809	0.6521211	2.92145	1.39494	86.36	-2.12968
6	0.06791	336	0.39259	0.5852621	0.728035	6.07545	0.4645	90.55	2.615584
8	0.29003	221	0.47444	0.7200762	0.5325258	6.84233	0.97672	88.92	-1.80016
9	0.0599	300	0.48058	0.9229804	0.950709	9.91947	0	86.58	-4.38296
3	0.3	100	0.5	1	1	10	10	82.87	-8.48641
7	0.06443	299	0.46991	0.8955632	0.931528	9.3424	0.20465	82.98	-8.36044
10	0.01	315	0.5	1	1	10	0	87.19	-3.71303
8	0.24072	174	0.41956	0.6154358	0.6675653	0.35628	3.95756	86.94	-3.99074
5	0.22327	151	0	0.5	0.7153427	1.37257	0.37098	86.36	-4.62615
8	0.07783	159	0.27687	0.5217279	0.6532014	9.96212	8.0399	87.61	-3.24146
8	0.28949	341	0.02785	0.7688727	0.8629891	7.85035	8.742	84.47	-6.71929
3	0.01	336	0.5	0.5	0.5	0	0	90.14	-0.4578
6	0.24158	340	0.43896	0.9370519	0.5444044	8.48533	1.39253	83.35	-7.95131
9	0.01	335	0.18557	0.5	1	8.98963	0.44258	88.82	-1.91013
8	0.01	341	0.01562	0.5	1	4.58593	0.87025	84.88	-6.26427
10	0.03863	302	0.5	1	1	7.26337	1.73721	86.64	-4.31867
6	0.2248	158	0.28334	0.7614067	0.6993633	0.77253	0.98499	85.78	-5.27052
3	0.01112	160	0.16475	0.7065307	0.7504244	2.65962	0.92076	88	-2.81323

The comparison of CoRXGB with DEEP-SE and RNN-CNN models in Figure 8 across multiple projects reveals that CoRXGB consistently delivers higher accuracy. For instance, CoRXGB achieved 82.47% accuracy on the Appcelerator Studio dataset, significantly outperforming RNN-CNN (62.29%) and DEEP-SE (60.26%). Similarly, CoRXGB excelled in other projects, such as Aptana Studio and Bamboo, where it recorded accuracies of 82.57% and 89.68%, respectively, compared to much lower scores by the other models. Notable high-performing cases for CoRXGB include Duracloud (90.63%) and Usergrid (87.48%). These results underscore CoRXGB's robustness and effectiveness in classifying complex user stories, demonstrating its superior generalization and adaptability across diverse datasets in agile story point classification tasks.

4.10. Performance Evaluation of CoRXGB on Appceleratorstudio project (Post Tuning)

This section represents the results of the performance of the CoRXGB model using hyperparameter tuning for different projects. Each result consists of a Learning Gain Matrix (LGM), model score, gain over iterations, hyperparameters iterations, and confusion matrix. Performance metrics have also been updated based on these results. Interpretation of the same has been included, along with comparative analysis for various projects.

The Learning Gain Matrix (LGM) Bayesian Optimization results for CoRXGB Hyperparameters are given in Table 13 for the Appcelerator Studio Project. The highest score of 90.55 was achieved with a max_depth of 6, learning_rate of 0.06791, n_estimators of 336, gamma of 0.39259, subsample of 0.5852621, colsample_bytree of 0.728035, reg_lambda of 6.07545, and reg_alpha of 0.4645. This result indicates that the model benefits from a moderate depth, a balanced learning rate, and a sufficient number of estimators. Table 13 shows that lower learning rates and higher numbers of estimators generally lead to higher accuracy scores.

The combination of CNN and RNN is used to extract the spatial and sequential features not done by other models. The use of Bayesian optimization helps in achieving the optimal configurations for the CoRXGB model.

The Appcelerator Studio project exhibited a substantial improvement in accuracy from 82.47% to 90.55% after hyperparameter tuning. The confusion matrix after Hyperparameter tuning is given in Figure 9.

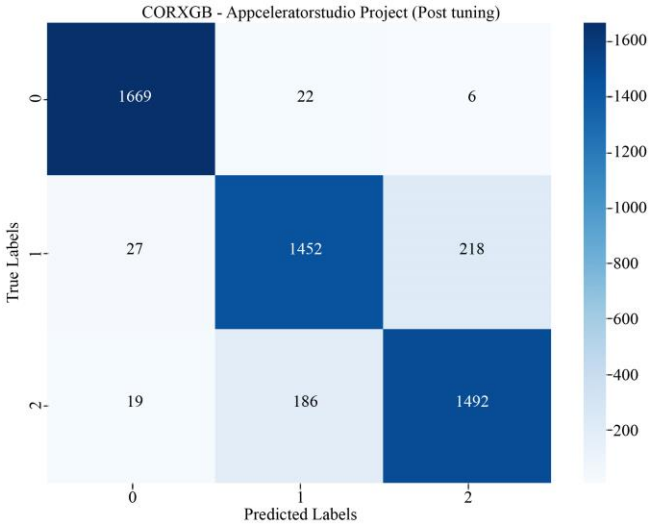

Fig. 9 Post-tuning confusion matrix of CoRXGB for Appcelerator studio project

The improved performance metrics for the Appcelerator studio project are given in Table 14.

Table 14. CoRXGB Performance Metrics after Tuning for Appcelerator Studio Project

Class/Metrics	Precision	Recall	F1 Score	Support
Class 0 (Easy)	0.973177	0.982518	0.977822	1697
Class 1 (Medium)	0.866742	0.855562	0.861114	1697
Class 2 (Complex)	0.867135	0.879141	0.873091	1697

A "Model score and gain over iterations" plot is given in Figure 10. The Blue Line (Score) represents the model's score over different iterations. It indicates the performance of the model as the iterations progress. The score fluctuates significantly for initial iterations, indicating instability in the model's performance initially. There is a significant drop in the score around iteration 5 in the middle iterations, followed by a recovery, suggesting the model may be adjusting to the parameters. In later iterations, the score remains somewhat volatile but shows overall improvement, indicating potential fine-tuning and optimization of the model. The Red Line (Gain%) represents the gain percentage over different iterations. It measures the change in performance relative to the previous iteration.

The initial iterations show high fluctuations, indicating significant changes in the model's performance, whereas, in middle iterations, a noticeable drop in gain around iteration 5 corresponds to the drop in the score. The gains become more stable in later iterations, with smaller fluctuations indicating more consistent model performance.

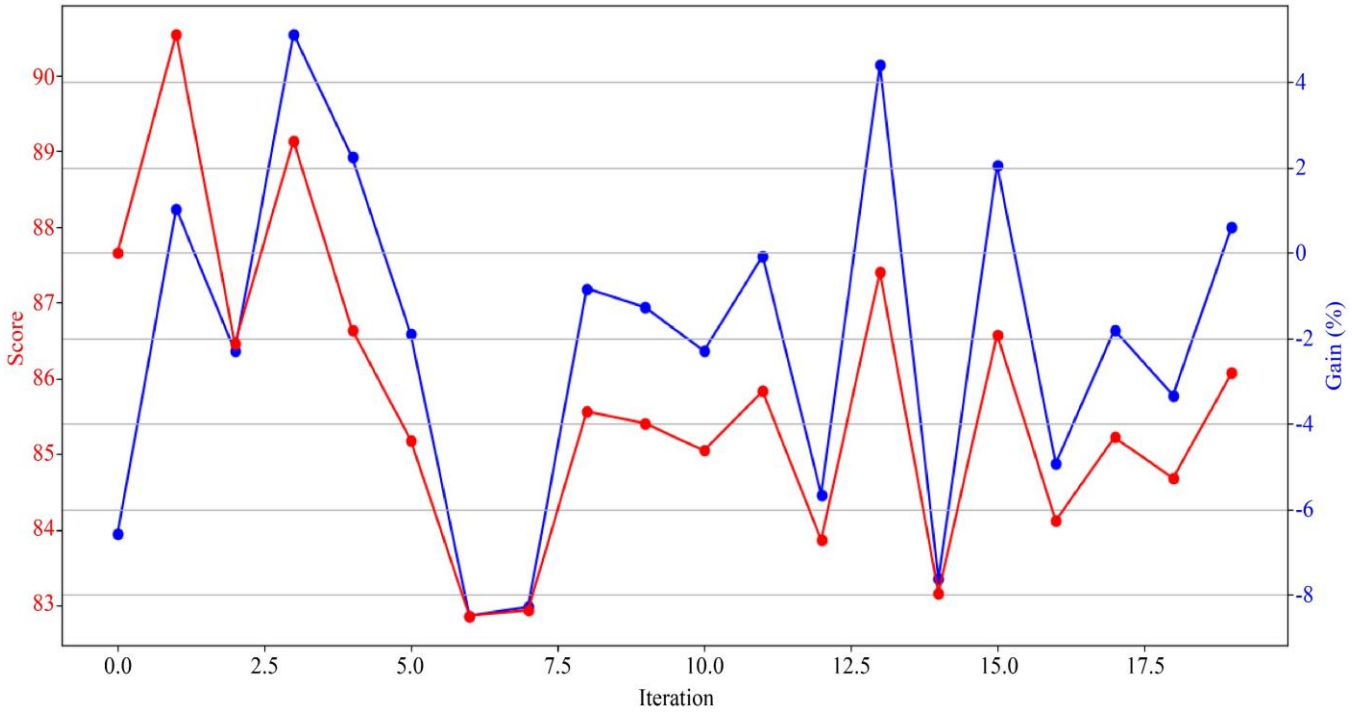

Fig. 10 Model score and gain over iterations – appcelerator studio

In Figure 11, a series of subplots shows how different hyperparameters impact the model's score across various iterations.

- max_depth vs. Score: A slight positive effect, indicating that increasing max_depth can improve the model's score up to a certain point. The optimal max_depth appears to be around 9.
- learning_rate vs. Score: A negative effect, suggesting that higher learning rates might decrease the model's score. Lower learning rates are associated with better performance.
- n_estimators vs. Score: Positive effect, showing that increasing the number of estimators tends to improve the model's score. The optimal number of estimators is around 350.
- Gamma vs. Score: Relatively flat, indicating that gamma has a minimal impact on the score within the tested range. Scores are relatively consistent regardless of gamma.
- Subsample vs. Score: Negative effect, indicating higher subsampling rates might reduce the score. Lower subsampling rates tend to result in better performance.
- Colsample_bytree vs. Score: Slight negative effect, suggesting that increasing colsample_by tree might slightly reduce the score. Optimal values seem to be around the middle range of the tested values.
- reg_lambda vs. Score: Negative effect, indicating that higher regularization (lambda) reduces the score. Lower values of reg_lambda are associated with better performance.
- reg_alpha vs. Score: The Negative effect indicates that higher regularization (alpha) reduces the score. The lower values of reg_alpha are associated with better performance.

4.11. Comparative Analysis of Model Performance across Projects

This section discusses the comprehensive comparative analysis of the model performance across various projects based on aspects like stability, hyperparameter effects, and model performance. These statistical approaches help identify the model's performance under different conditions and configurations.

This analysis has been structured to evaluate quantitative and qualitative measures, where quantitative measures include standard deviation, range, and stability scores, and qualitative measures include score trends and gain percentages. It also helps in understanding the behavior of the model.

4.12. Hyperparameter Impact

With the help of visualization techniques, the relationship between hyperparameter values and model scores has been depicted. It helps identify the optimal settings for each project. Figure 12 shows a heatmap of the comparative analysis of Hyperparameter effects where 1 depicts "positive effect", 0 for "Minimal effect", and -1 means "Negative effect". Figure 13 represents the count of positive (+), negative (-), and minimal effects for each hyperparameter.

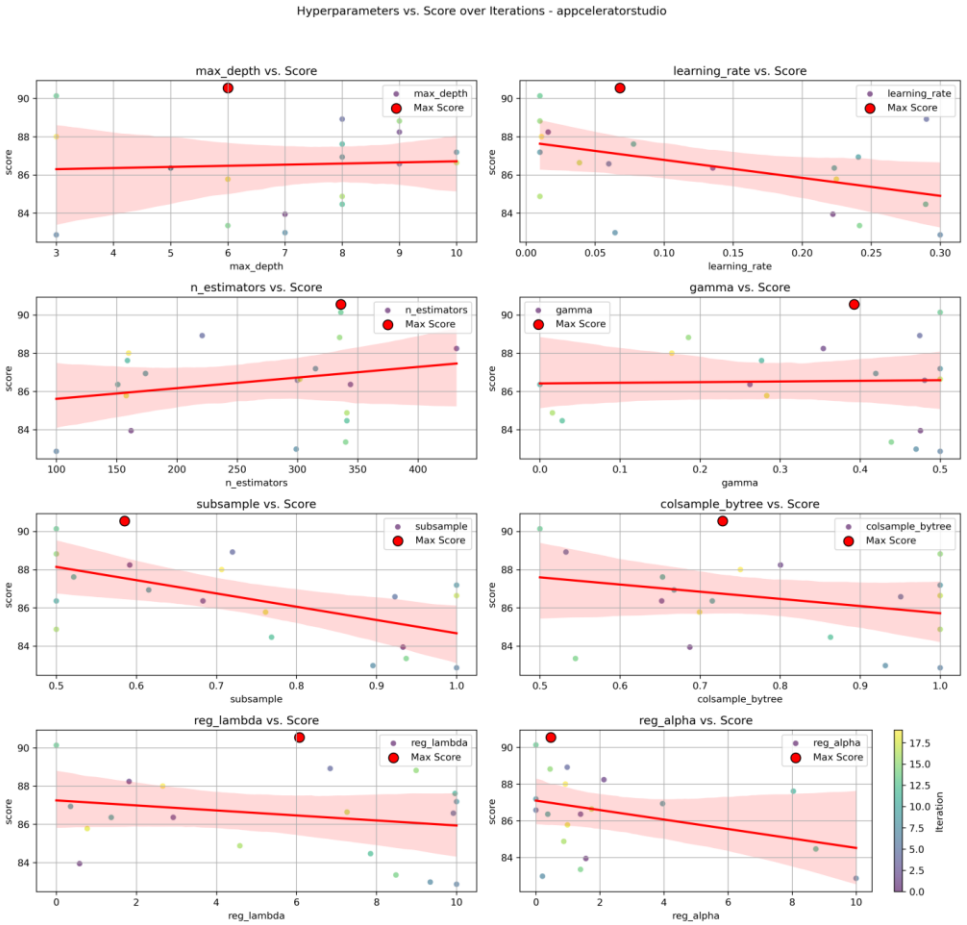

Fig. 11 Hyperparameters vs score over iterations – appcelerator studio

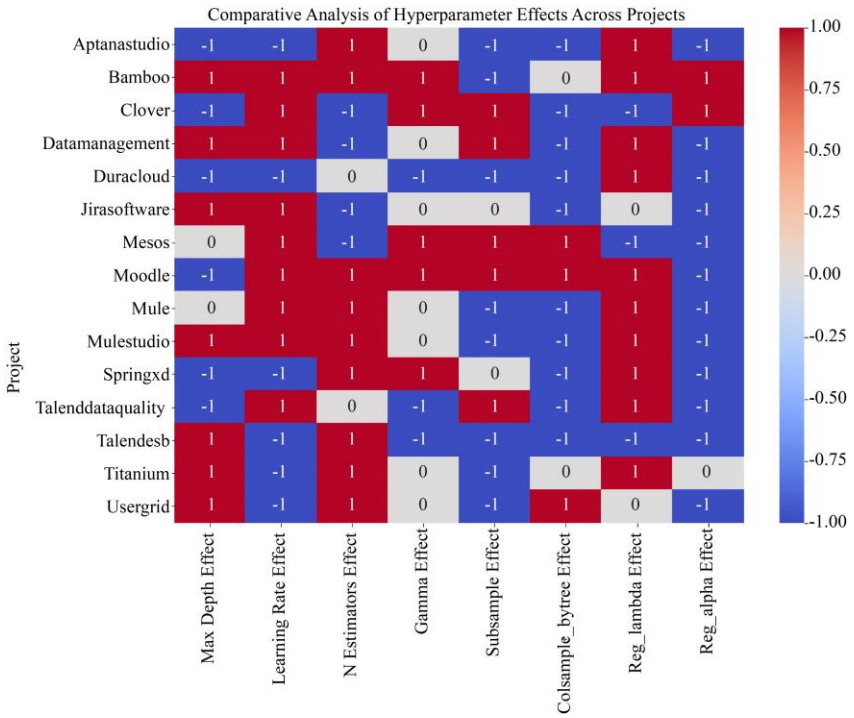

Fig. 12 Comparative analysis of Hyperparameters effects across projects

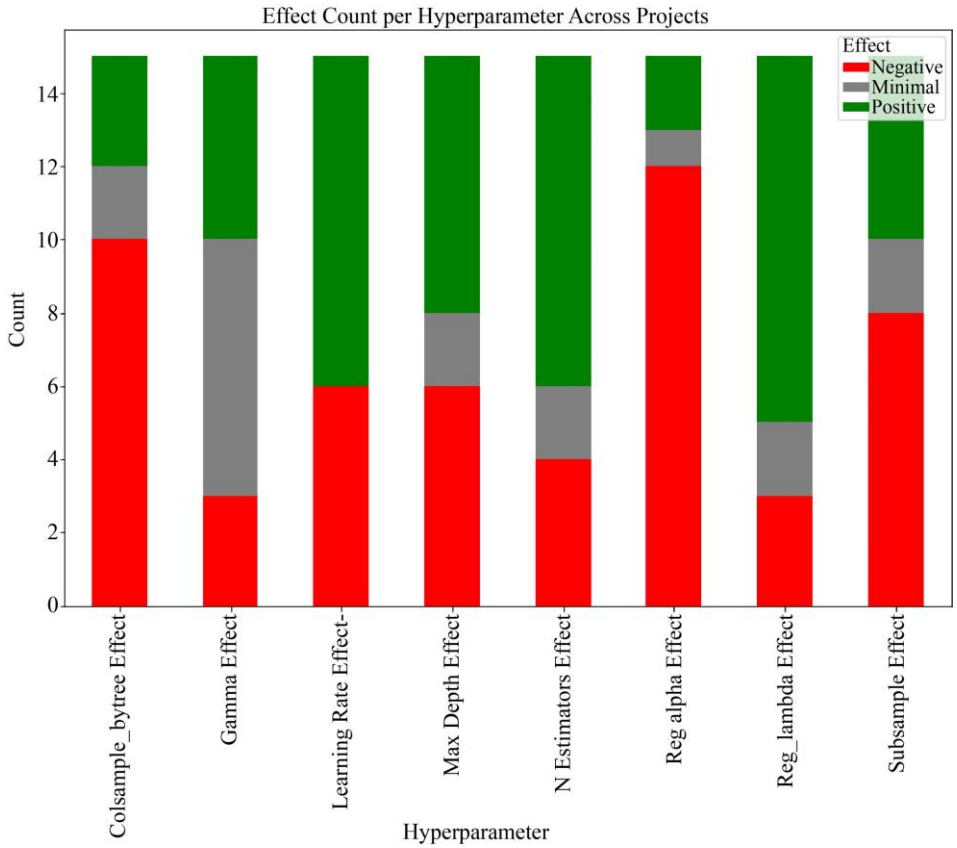

Fig. 13 Effect count per hyperparameter

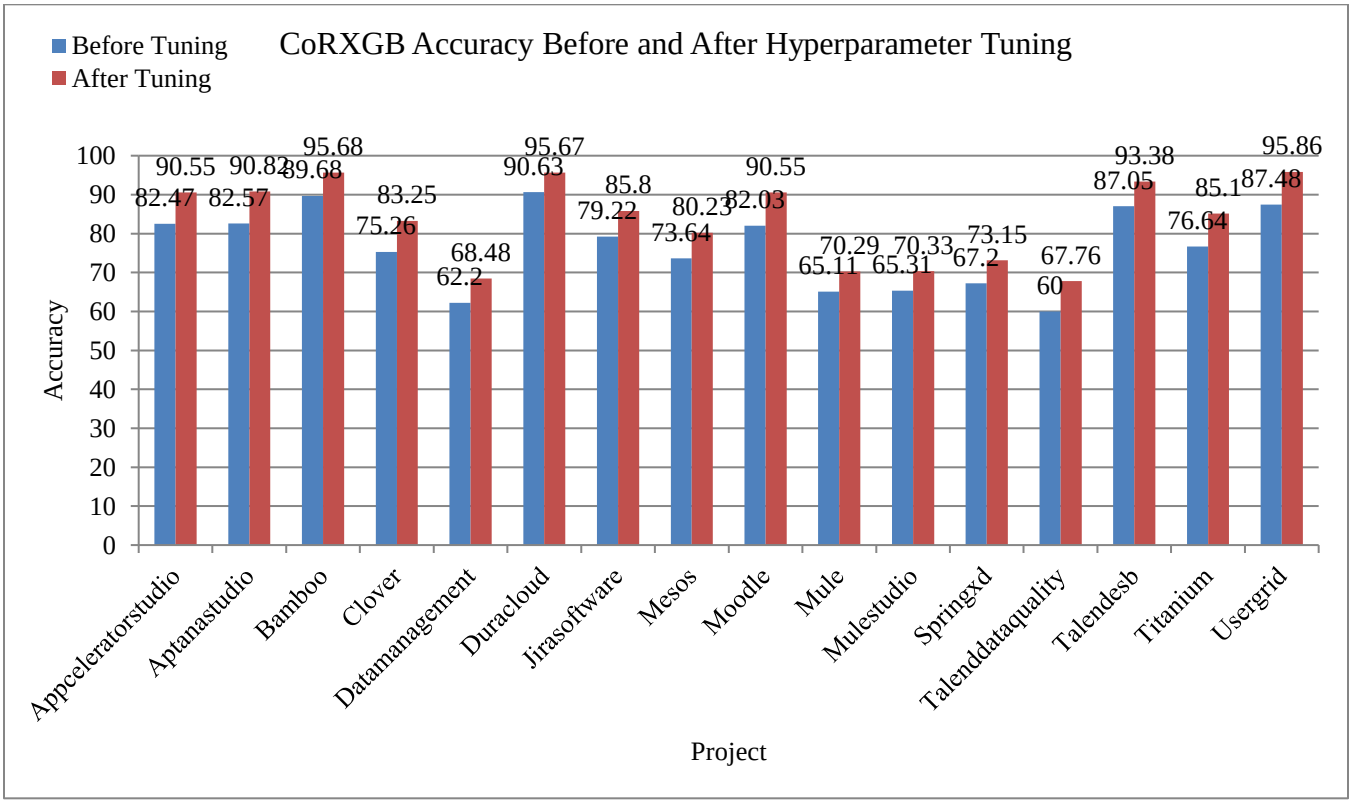

Fig. 14 CoRXGB accuracy comparison (Before and After Hyperparameter Tuning)

4.13. CoRXGB Accuracy Comparison (Before and After Hyperparameter Tuning)

An accuracy comparison of the CoRXGB model has been depicted in Table 15 and Figure 14 by performing hyperparameter tuning before and after. The importance of tuning has been clearly seen in accuracy improvements and increased gain values from moderate to substantial across all projects.

Table 15. CoRXGB Accuracy Comparison after Hyperparameter tuning

Project	Initial Accuracy (Before tuning)	Accuracy After Tuning
Appceleratorstudio	82.47	90.55
Aptanastudio	82.57	90.82
Bamboo	89.68	95.68
Clover	75.26	83.25
Datamanagement	62.2	68.48
Duracloud	90.63	95.67
Jirasoftware	79.22	85.8
Mesos	73.64	80.23
Moodle	82.03	90.55
Mule	65.11	70.29
Mulestudio	65.31	70.33
Springxd	67.2	73.15
Talenddataquality	60	67.76
Talendesb	87.05	93.38
Titanium	76.64	85.1
Usergrid	87.48	95.86

The moodle project shows improvements in accuracy from 82.04% to 90.55% after performing hyper tuning. The average improvement of the CoRXGB model is 6.91 percentage points by performing the hypertuning. It also shows the impact of hyperparameter tuning in increasing model performance.

4.14. Statistical Hypothesis Testing Results for CoRXGB vs DEEP-SE and RNN-CNN

A series of statistical hypothesis tests have been performed. T-tests and Wilcoxon Signed-Rank tests have been performed to check the accuracy between CoRXGB and the other two models. A paired t-test is used to find out the difference between two related groups by calculating means. The equation for the paired t-test is:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}} \tag{20}$$

The Wilcoxon Signed-Rank Test W is calculated as follows:

$$W = \sum Ranks\ of\ positive\ differences - \sum Ranks\ of\ negative\ differences \tag{21}$$

Hypothesis

- Null Hypothesis (H0): No major difference in the accuracy of CoRXGB and other models (DEEP-SE or RNN-CNN).
- Alternative Hypothesis (H1): Major difference in the accuracy of CoRXGB and other models (DEEP-SE or RNN-CNN).

4.15. Results Table

The results for both tests are given in Table 16.

The paired t-test results show extremely low p-values, indicating a significant difference in accuracy between CoRXGB and both DEEP-SE and RNN-CNN. The high t-statistic values further reinforce this conclusion. The Wilcoxon Signed-Rank Test results also exhibit extremely low p-values, indicating that the accuracy differences between CoRXGB and the other two models are statistically significant. The Wilcoxon statistic being 0.0 suggests a consistent pattern where CoRXGB consistently outperforms DEEP-SE and RNN-CNN across all datasets. The results from the paired t-tests and the Wilcoxon Signed-Rank tests strongly suggest that the CoRXGB model significantly outperforms DEEP-SE and RNN-CNN regarding accuracy across various projects. The p-values, i.e., <0.05, show that the null hypothesis can be rejected. The CoRXGB model provides improved performance than DEEP-SE and RNN-CNN. This makes the CoRXGB model a better choice for predicting the complexity of user stories. Paired t-tests and Wilcoxon Signed-Rank tests accuracy results show that the CoRXGB model outperforms DEEP-SE and RNN-CNN approaches.

Table 16. Statistical Comparison of CoRXGB with DEEP-SE and RNN-CNN Model

Comparison	Paired t-test t-statistic	Paired t-test p-value	Wilcoxon Signed-Rank Test Statistic	Wilcoxon Signed-Rank Test p-value
CoRXGB vs DEEP-SE	9.548	9.15E-08	0	3.05E-05
CoRXGB vs RNN-CNN	9.235	1.41E-07	0	3.05E-05

5. Conclusion and Future Work

The paper represents the results in the field of Agile project management using the CoRXGB model. The CoRXGB model improves story point estimation accuracy by combining the features of CNN, RNN, and XGBoost algorithms. The CoRXGB model predicts the user story points. Our study is supported by the extensive literature work in the field of Agile project Development. The CoRXGB model uses CNN for feature extraction, LSTM for extracting sequential dependencies, and XGBoost for efficient classification. Advanced preprocessing techniques like TF-IDF vectorization have also been implemented. The model was trained using Bayesian Optimization ensuring optimal performance supported with Hyperparameter tuning. Overall, the CoRXGB model has proven to be the tool for enhancing Agile project management efficiency.

The various advanced machine learning techniques, like transformers, could be integrated with the CoRXGB model. Team dynamics and human factors such as team size and experience can also be included to improve the model's adaptability. The CoRXGB model can be deployed in the real world by agile practitioners as it provides customized estimations of industry-specific requirements. The dataset biases can also be explored to improve the model's generalizability. Explainable AI techniques like SHAP, LIME, etc., can enhance the model's interpretability.

References

[1] Nisma Gaffar et al., "A Proposed Framework for Enhancing Story Points in Agile Software Projects," *Indian Journal of Science and Technology*, vol. 11, no. 31, pp. 1-11, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[2] Janeth Lopez-Martinez et al., "Estimating User Stories' Complexity and Importance in Scrum with Bayesian Networks," *Recent Advances in Information Systems and Technologies*, pp. 205-214, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[3] M. Shepperd, and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Transactions on Software Engineering*, vol. 23, no. 11, pp. 736-743, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[4] Ravi Kiran Mallidi, and Manmohan Sharma, "Study on Agile Story Point Estimation Techniques and Challenges," *International Journal of Computer Applications*, vol. 174, no. 13, pp. 9-14, 2021. [Google Scholar] [Publisher Link]

[5] Suyash Shukla, and Sandeep Kumar, "Study of Learning Techniques for Effort Estimation in Object-Oriented Software Development," *IEEE Transactions on Engineering Management*, vol. 71, pp. 4602-4618, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Mohit Arora et al., "An Efficient ANFIS-EEBAT Approach to Estimate Effort of Scrum Projects," *Scientifc Reports*, vol. 12, pp. 1-14, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Ekrem Kocaguneli, Tim Menzies, and Jacky W. Keung, "On the Value of Ensemble Effort Estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403-1416, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[8] Onkar Malgonde, and Kaushal Chari, "An Ensemble-Based Model for Predicting Agile Software Development Effort," *Empirical Software Engineering*, vol. 24, pp. 1017-1055, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Claudio Ratke et al., "Effort Estimation Using Bayesian Networks for Agile Development," *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, Riyadh, Saudi Arabia, pp. 1-4, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[10] Ali Bou Nassif et al., "Neural Network Models for Software Development Effort Estimation: A Comparative Study," *Neural Computing and Applications*, vol. 27, no. 8, pp. 2369-2381, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[11] Asad Ali, and Carmine Gravino, "A Systematic Literature Review of Software Effort Prediction using Machine Learning Methods," *Journal of Software: Evolution and Process*, vol. 31, no. 10, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[12] Vlad-Sebastian, Horia, and Istvan-Gergely, "Natural Language Processing and Machine Learning Methods for Software Development Effort Estimation," *Studies in Informatics and Control*, vol. 26, no. 2, pp. 219-228, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[13] Akshay Jadhav et al., "Effective Software Effort Estimation Leveraging Machine Learning for Digital Transformation," *IEEE Access*, vol. 11, pp. 83523-83536, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[14] Harish Kumar Mittal, Mohd Arsalan, and Puneet Garg, "A Novel Deep Learning Model for Effective Story Point Estimation in Agile Software Development," *2024 International Conference on Emerging Innovations and Advanced Computing (INNOCOMP)*, Sonipat, India, pp. 404-410, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[15] Bhaskar Marapelli, Anil Carie, and Sardar M.N. Islam, "RNN-CNN MODEL:A Bi-directional Long Short-Term Memory Deep Learning Network For Story Point Estimation," *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*, Sydney, Australia, pp. 1-7, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[16] Haithem Kassem, Khaled Mahar, and Amani A. Saad, "Story Point Estimation Using Issue Reports with Deep Attention Neural Network," *E-Informatica Software Engineering Journal*, vol. 17, no. 1, pp. 1-15, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[17] Ali Bou Nassif et al., "Software Development Effort Estimation Using Regression Fuzzy Models," *Computational Intelligence and Neuroscience*, vol. 2019, pp. 1-17, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jasem M. Alostad, Laila R.A. Abdulla, and Lamya Sulaiman Aali, "A Fuzzy Based Model for Effort Estimation in Scrum Projects," *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 9, pp. 270-277, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[19] Marta Fernandez-Diego et al., "An Update on Effort Estimation in Agile Software Development: A Systematic Literature Review," *IEEE Access*, vol. 8, pp. 166768-166800, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] Pekka Abrahamsson et al., "Predicting Development Effort from User Stories," *2011 International Symposium on Empirical Software Engineering and Measurement*, Banff, AB, Canada, pp. 400-403, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[21] Simone Porru et al., "Estimating Story Points from Issue Reports," *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, Ciudad Real, Spain, pp. 1-10, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[22] Ezequiel Scott, and Dietmar Pfahl, "Using Developers' Features to Estimate Story Points," *Proceedings of the 2018 International Conference on Software and System Process*, Gothenburg, Sweden, pp. 106-110, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[23] Morakot Choetkiertikul et al., "A Deep Learning Model for Estimating Story Points," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 637-656, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[24] Michael Fu, and Chakkrit Tantithamthavorn, "GPT2SP: A Transformer-Based Agile Story Point Estimation Approach," *IEEE Transactions on Software Engineering*, vol. 49, no. 2, pp. 611-625, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[25] Tianqi Chen, and Carlos Guestrin, "XGBoost : A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA, pp. 785-794, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[26] Xilu Wang et al., "Recent Advances in Bayesian Optimization," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1-36, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[27] James Bergstra, Daniel Yamins, and David Cox, "Making a Science of Model Search : Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, no. 1, pp. 115-123, Atlanta, Georgia, USA, 2013. [Google Scholar] [Publisher Link]

[28] Macarious Abadeer, and Mehrdad Sabetzadeh, "Machine Learning-based Estimation of Story Points in Agile Development : Industrial Experience and Lessons Learned," *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pp. 106-115, Notre Dame, IN, USA, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[29] Muaz Gultekin, and Oya Kalipsiz, "Story Point-Based Effort Estimation Model with Machine Learning Techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 1, pp. 43-66, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[30] Przemyslaw Pospieszny, "Software Estimation: Towards Prescriptive Analytics," *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*, Gothenburg Sweden, pp. 221-226, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[31] Janeth Lopez-Martinez et al., "User Stories Complexity Estimation using Bayesian Networks for Inexperienced Developers," *Cluster Computing*, vol. 21, pp. 715-728, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[32] Panut Chongpakdee, and Wiwat Vatanawood, "Estimating User Story Points Using Document Fingerprints," *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China, pp. 149-152, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[33] Ahmad Azzazi, "A Framework using NLP to Automatically Convert User-Stories into Use Cases in Software Projects," *International Journal of Computer Science and Network Security*, vol. 17, no. 5, pp. 71-76, 2017. [Google Scholar] [Publisher Link]

[34] M. Thangaraj, and M Sivakami, "Text Classification Techniques: A Literature Review," *Interdisciplinary Journal of Information, Knowledge, and Management, Knowledge*, vol. 13, pp. 117-135, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[35] Vali Tawosi, and Rebecca Moussa, "Agile Effort Estimation: Have We Solved the Problem Yet? Insights from a Replication Study," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2677-2697, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[36] Hung Phan, and Ali Jannesari, "Story Point Effort Estimation by Text Level Graph Neural Network," *arxiv*, pp. 1-4, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[37] Jiale Wu et al., "Toward Efficient and Effective Bullying Detection in Online Social Network," *Peer-to-Peer Networking and Applications*, vol. 13, no. 5, pp. 1567-1576, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[38] Chaudhary Hamza Rashid et al., "Software Cost and Effort Estimation : Current Approaches and Future Trends," *IEEE Access*, vol. 11, pp. 99268-99288, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[39] Lan Cao, "Estimating Efforts for Various Activities in Agile Software Development : An Empirical Study," *IEEE Access*, vol. 10, pp. 83311-83321, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[40] Indra Kharisma Raharjana, Daniel Siahaan, and Chastine Fatichah, "User Stories and Natural Language Processing: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 53811-53826, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[41] Julliano Trindade Pintas, Leandro A.F. Fernandes, and Ana Cristina Bicharra Garcia, "Feature Selection Methods for Text Classification: a Systematic Literature Review, vol. 54, no. 8, pp. 6149-6200, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[42] Sepp Hochreiter, and Jurgen Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997. [CrossRef] [Google Scholar] [Publisher Link]

[43] Burcu Yalcıner et al., "Enhancing Agile Story Point Estimation: Integrating Deep Learning, Machine Learning, and Natural Language Processing with SBERT and Gradient Boosted Trees," *Applied Sciences*, vol. 14, no. 16, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[44] Eduardo Rodriguez Sanchez, Eduardo Filemon Vazquez Santacruz, and Humberto Cervantes Maceda, "Effort and Cost Estimation Using Decision Tree Techniques and Story Points in Agile Software Development," *Mathematics*, vol. 11, no. 6, pp. 1-31, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[45] Remah Younisse, and Mohammad Azzeh, "Application of Natural Language Processing Techniques in Agile Software Project Management: A Survey," *2023 14th International Conference on Information and Communication Systems (ICICS)*, Irbid, Jordan, pp. 1-6, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[46] Haithem Kassem, Khaled Mahar, and Amani Saad, "Software Effort Estimation using Hierarchical Attention Neural Network," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 18, pp. 5308-5322, 2022. [Google Scholar] [Publisher Link]

[47] N.V. Chawla et al., "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002. [CrossRef] [Google Scholar] [Publisher Link]