



A BEHAVIOURAL ANALYTICS FRAMEWORK FOR EARLY PERFORMANCE PREDICTION IN INTRODUCTORY ENGINEERING PROGRAMMING COURSES

Hetal Gosrani*¹ and Nirav Bhatt²

^{1,2}Department of Computer Science, School of Engineering, RK University, Rajkot, 360200, Gujarat, India.

¹<http://orcid.org/0009-0006-2220-3740>, ²<http://orcid.org/0000-0001-6481-7739>

Email: *hetalgsavla@gmail.com, nirav.bhatt@rku.ac.in

ARTICLE INFO

Article History

Received: February 9, 2026

Reviewed: April 16, 2026

Accepted: May 16, 2026

Published: June 30, 2026

Keywords:

Behavioural Activity Data, Educational Data Mining, Ensemble Learning, Introductory Programming Courses in Engineering and Computer Science Programs, Student Performance Prediction.

ABSTRACT

Introductory computer science engineering programming courses are associated with high failure and attrition rates; hence they need reliable early student performance prediction to enable timely academic interventions. This study proposes a behavioural analytics-based framework for predicting student performance. The study used over one lakh programming submission records collected from an online programming environment from which total 4,039 student level aggregated activity logs were constructed. Authors then applied advanced feature engineering to these aggregated data to obtain temporal engagement patterns, error rates, effort efficiency, and aggregated engagement measures. Logistic Regression (LR), Random Forest (RF), Gradient Boosting Classifier (GBC), Multilayer Perceptron (MLP), Support Vector Machine (SVM), and Extreme Gradient Boosting (XGBoost) were tested on an 80:20 train-test split and 10-fold GroupKFold cross-validation. The Authors subsequently developed a stacking ensemble model combining RF, GBC, MLP, SVM, and XGBoost as base learners, with LR as the meta-learner. The ensemble obtained F1-score of 0.857 and ROC-AUC of 0.905 and was more reliable than individual classifiers. Statistical significance of the performance gains achieved through behavioural feature engineering. Authors confirmed Ensemble learning using a paired Wilcoxon signed-rank test. The proposed framework is applicable to detect at-risk students, promotes certain pedagogical interventions in computer science education.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

In computer science education, introductory programming courses in engineering and computer science programs are often the first major challenge faced by students. Students must learn new programming rules, think logically, and practise often. Because of this, some students begin to struggle very early in the course. Usually, teachers notice these problems only after exams, when it becomes difficult to help the students. Therefore, finding at-risk students as early as possible is very important. Earlier research on predicting student performance mainly used fixed information such as student background details or previous exam results. While this information gives some basic idea, it does not show how students actually learn programming during the course. Programming skills take time to develop. Students usually improve by writing code many times. They make mistakes, fix them, and then try again. Learning happens gradually in this way. Because of this, learning problems may appear early. Final exam scores do not always show these issues. Methods that rely only on exam results can miss them. Many programming courses now use online submission systems. These systems record basic activity information. This includes how often students submit code. They also record whether programs run correctly. The number of errors is captured as well. Submission times are stored automatically. Because this information is collected from the start of the course, it can be useful. It helps in understanding student progress at an early stage. However, this activity data is not easy to use. Individual exercise records are scattered. They do not show how a student works across the entire course. To make the data useful, activities need to be grouped at the student level. The grouped data must then be converted into meaningful features.

These features describe learning behaviour over time. Such features can represent effort and consistency. They can show common error patterns. They can also reflect task difficulty and timing of engagement. Together, these features provide a clearer picture of how students learn programming. Machine learning models are often used to study learning behaviour and predict student outcomes. Different models learn different patterns, but depending on only one model may not always give reliable results. Ensemble learning combines multiple models and usually produces better and more stable predictions. Among these methods, stacking ensembles combine the strengths of several models to improve overall prediction performance. Apart from getting good accuracy, it is also important to check whether the results are actually dependable. Sometimes, differences between models can appear just by chance. A statistical test is used to verify the results. The Wilcoxon signed-rank test is applied because it is suitable for comparing machine learning models without making strong assumptions about the data. Based on the reviewed studies and the observed limitations, this study addresses the following research questions:

- (i) Can behavioural features derived from programming exercise submissions support early identification of at-risk students in introductory programming courses in engineering and computer science programs?
- (ii) Does combining multiple machine learning models lead to more reliable prediction of student performance compared to using individual models?
- (iii) Do the observed differences between prediction models remain meaningful when evaluated using the Wilcoxon signed-rank test?

II. THEORETICAL REFERENCE

II.1 EARLY PREDICTION OF STUDENT PERFORMANCE IN PROGRAMMING COURSES

Introductory programming courses in engineering and computer science programs are well known for being difficult for many students. Several studies report that students often face problems very early in such courses and that these early difficulties later lead to failure or dropout [1], [2]. Because of this, many researchers have focused on predicting student performance as early as possible in CS1 and programming fundamentals courses [3], [4]. Earlier research mainly used information such as student background details or previous academic results to predict performance. Although this information is easy to collect, it does not explain how students learn programming during a course [5], [6]. Programming skills do not develop instantly. Students learn by writing code repeatedly, making mistakes, and fixing them over time. Studies therefore suggest that learning behaviour during the course is more useful than static information for early prediction [7], [8].

II.2 USE OF BEHAVIOURAL DATA AND MACHINE LEARNING MODELS

With the use of online programming platforms and learning management systems, it has become possible to collect detailed records of student activity. These systems store data such as code submissions, compilation errors, execution results, task completion, and time spent on exercises [9], [10]. This data gives a direct view of students' programming activity during the course. However, raw activity data is difficult to use directly. Researchers usually do not work with raw log data. Instead, the data is first cleaned and reshaped so that it can describe how students actually work while learning programming [11], [12]. Commonly used indicators include how often students submit code, how many errors they make, whether they complete tasks, and how much time they spend on programming work [13], [14]. Combining activity data at the student level helps show how students learn over time [15], [16]. A variety of machine learning models have been applied to this type of data. These include Decision Tree (DT), naïve Bayes, Support Vector Machine (SVM), Random Forest (RF), and Neural Network [17-19]. Many researchers have compared different machine learning models, but no single model performs best in all situations. The results often change depending on the dataset and the features used [20], [21]. Some recent work has applied LSTM models to study how student behaviour changes over time. These models can be useful, but they usually require a large amount of data and higher computational effort [22], [23].

II.3 ENSEMBLE METHODS AND EVALUATION OF PREDICTION MODELS

Several researchers suggest that using a single machine learning model may not always give reliable results. To address this issue, ensemble methods that combine multiple models have been proposed [24], [25]. In some studies, predictions from more than one model are used together instead of relying on a single model. This is done by combining their outputs through an additional model, and this approach has been used in student performance prediction [26]. Many studies report accuracy and similar measures, but they do not always check whether the differences between models are meaningful. Sometimes, the performance gap between models may simply be due to random variation rather than an actual improvement. Statistical tests are therefore recommended for reliable model comparison. Non-parametric tests such as the Wilcoxon signed-rank test are suitable for comparing machine learning models because they do not rely on strict assumptions about data distribution. However, such tests are still not commonly applied in programming education research.

III. MATERIALS AND METHODS

III.1 DATA SOURCE AND PREPARATION

This study used behavioural data collected from an online programming submission system used in introductory programming courses in engineering and computer science programs. Similar online systems have been used in earlier studies to analyse how students learn programming through their activity logs [4], [8]. The system automatically records students' programming actions, such as code submissions, errors during compilation, task completion, and submission time. The dataset covers students from multiple courses and institutions. The original data was stored at the level of individual programming exercises. Since the aim of this study was to predict student performance, all programming activities of each student were combined into a single record.

This type of student-level aggregation has been widely used because it provides a clearer view of overall learning behaviour [15], [16]. The final dataset consisted of 3,049 students, and the target variable indicated whether a student passed or failed the course. A summary of the dataset and its main characteristics is shown in Table 1. No student names or identifying details were used.

Table 1: Summary of the dataset used in this study.

Category	Description
Course	Introductory programming course (CS1)
Data source	Online programming and automated assessment system [1]
Unit of analysis	Student-level records
Recorded activity	Submission attempts, task completion, syntax errors, and time spent [1]
Task difficulty	Easy, medium, and hard
Final outcome	Pass or fail

Source: Authors, (2026).

The overall process followed in this study, from data collection to model evaluation, is illustrated in Figure 1.



Figure 1: Main steps followed in this study, from data collection to model evaluation.

Source: Authors, (2026).

III.2 BEHAVIOURAL FEATURE ENGINEERING

Raw programming logs contain detailed but scattered information and are difficult to use directly for prediction. For this reason, the authors converted the activity data into behavioural features that describe how students learned over time. Similar ways of working with student activity data have been used in earlier studies [11], [12]. The features were designed to reflect how students worked on programming exercises throughout the course. The features were based on how students worked on exercises, such as whether they finished tasks, how often they tried again, the mistakes they made, and the time they submitted their code. Programming exercises were also grouped into easy, medium, and hard categories, and difficulty-based features were created. Using difficulty-aware features helps better represent how students progress while learning programming [7], [27]. A detailed description of all behavioural features used in this study is provided in Table 2, while an overview of the feature construction process is shown in Figure 2.

Table 2: Programming activity features used in this study.

Feature group	Feature	Computation	What it represents
Task completion	EasyCompletionRatio _i	CompletedEasy _i / TotalEasy _i	Completion level for easy tasks
	MediumCompletionRatio _i	CompletedMedium _i / TotalMedium _i	Completion level for medium tasks
	HardCompletionRatio _i	CompletedHard _i / TotalHard _i	Completion level for hard tasks
Work efficiency	EasyEffortEfficiency _i	TimeEasy _i / (CompletedEasy _i + 1)	Time spent per completed easy task
	MediumEffortEfficiency _i	TimeMedium _i / (CompletedMedium _i + 1)	Time spent per completed medium task
	HardEffortEfficiency _i	TimeHard _i / (CompletedHard _i + 1)	Time spent per completed hard task
Coding errors	EasyErrorRate _i	SyntaxErrorsEasy _i / (AttemptsEasy _i + 1)	Errors made in easy tasks
	MediumErrorRate _i	SyntaxErrorsMedium _i / (AttemptsMedium _i + 1)	Errors made in medium tasks
	HardErrorRate _i	SyntaxErrorsHard _i / (AttemptsHard _i + 1)	Errors made in hard tasks
Difficulty-based	CompletedWeightedScore _i	1 · CompletedEasy _i + 2 · CompletedMedium _i + 3 · CompletedHard _i	Completion adjusted by task difficulty
	AttemptsWeightedScore _i	1 · AttemptsEasy _i + 2 · AttemptsMedium _i + 3 · AttemptsHard _i	Effort adjusted by task difficulty
	SyntaxErrorWeightedScore _i	1 · SyntaxErrorsEasy _i + 2 · SyntaxErrorsMedium _i + 3 · SyntaxErrorsHard _i	Errors adjusted by task difficulty
Timing	TimeSpanEncoded _i	{1 = Early, 2 = Mid, 3 = Late}	When the student was most active
Overall activity	TotalCompleted _i	CompletedEasy _i + CompletedMedium _i + CompletedHard _i	Total number of completed tasks
	TotalAttempts _i	AttemptsEasy _i + AttemptsMedium _i + AttemptsHard _i	Total number of attempts
	TotalErrors _i	SyntaxErrorsEasy _i + SyntaxErrorsMedium _i + SyntaxErrorsHard _i	Total syntax errors
	OverallEfficiency _i	TotalCompleted _i / (TotalAttempts _i + 1)	Overall efficiency across tasks
Outcome	Target _i	{1 = Pass, 0 = Fail}	Course result

Source: Authors, (2026).

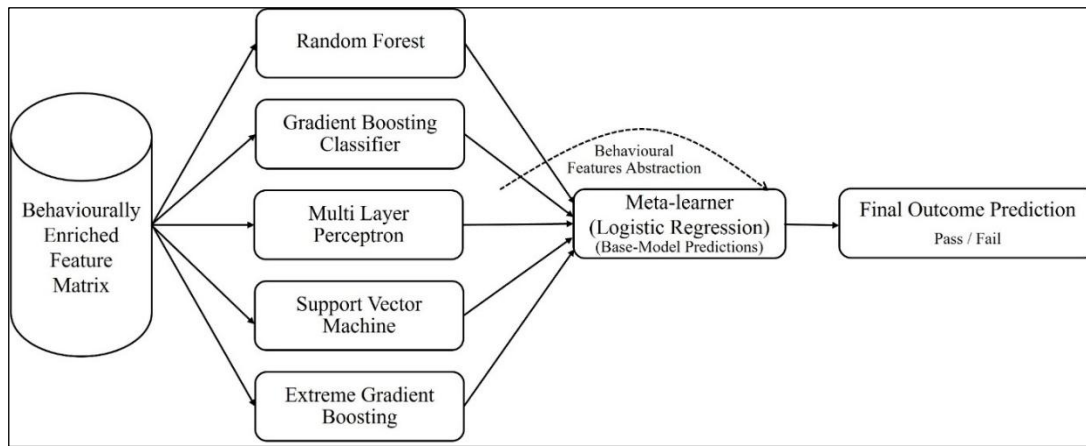


Figure 1: An overview of the feature construction steps.

Source: Authors, (2026).

Only features derived from early course activity were considered. Data from the later part of the course was not included. This choice kept the focus on how students behaved at the beginning. Early behaviour is more useful for noticing students who may face difficulties. Such early signals allow risk to be identified before final assessments [1], [2].

III.3 MODEL DEVELOPMENT AND EVALUATION STRATEGY

In this study, the task was to identify whether a student would pass or fail the course. For this purpose, multiple machine learning models were tested, such as Logistic Regression (LR), SVM, RF, Gradient Boosting Classifier (GBC), Multilayer Perceptron (MLP), and Extreme Gradient Boosting (XGBoost). Similar models have been applied in previous studies on student performance and were therefore considered appropriate for analysing programming behaviour data [17], [19], [21]. The authors used LR mainly as a reference point. It provided a simple baseline for comparison. The authors applied SVM to observe differences between students with different learning patterns. RF and Gradient Boosting were used to capture relationships that simpler models may overlook. MLP and XGBoost were included to study more complex patterns in the data [18], [25]. The authors divided the dataset into two parts. They used eighty percent of the data for training. They kept remaining twenty percent reserved for testing. They used Ten-fold cross-validation during training to evaluate the models on different data splits [6]. They followed the overall modelling process in this study as illustrated in Figure 3.



Figure 2: Model training and evaluation steps used in this study.

Source: Authors, (2026).

The authors included a statistical validation step to ensure that differences between models were not due to chance. They used the Wilcoxon signed-rank test to compare model results. They used this test because it can compare model results without relying on strict assumptions about the data.

IV. RESULTS AND DISCUSSIONS

IV.1 PERFORMANCE OF BASELINE MODELS USING RAW BEHAVIOURAL FEATURES

The first analysis examined the performance of individual machine learning models trained using raw behavioural features. The results show that all models achieved comparable performance, with accuracy values ranging from 0.783 to 0.802. LR provided a reasonable baseline, achieving an F1-score of 0.829 and ROC-AUC of 0.86. The tree-based models performed a little better than the others. RF obtained an F1-score of 0.838, and Gradient Boosting reached 0.842. SVM and MLP showed similar behaviour, with F1-scores around 0.84. Among the baseline models, XGBoost recorded the highest F1-score (0.844) and ROC-AUC value (0.894). A detailed comparison of baseline model performance is presented in Table 3.

Table 3: Machine learning baseline model performance with raw behavioural features.

Model	Accuracy	Precision	Recall	F1	ROC-AUC
LR	0.783	0.81	0.849	0.829	0.86
RF	0.797	0.827	0.849	0.838	0.889
GBC	0.802	0.83	0.855	0.842	0.894
MLP	0.792	0.825	0.843	0.834	0.885
SVM	0.798	0.828	0.851	0.839	0.887
XGBoost	0.802	0.824	0.864	0.844	0.894

Source: Authors, (2026).

The comparison of F1-scores across baseline models is illustrated in Figure 4, while the corresponding ROC curves are shown in Figure 5.

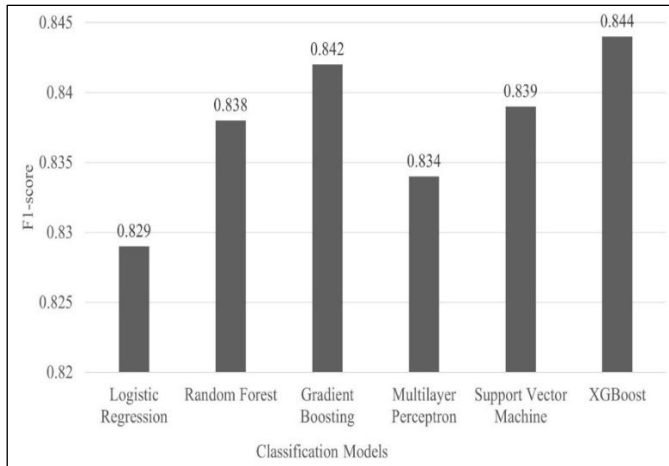


Figure 3: F1-score comparison of baseline classification models.
Source: Authors, (2026).

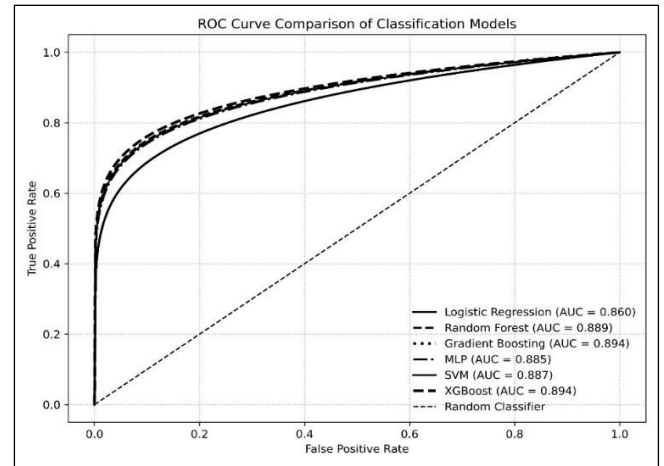


Figure 4: ROC curves of baseline classification models using raw behavioural features.
Source: Authors, (2026).

IV.2 EFFECT OF ENGINEERED BEHAVIOURAL FEATURES

The next analysis compared baseline behavioural features with engineered behavioural features. All models showed better results after the engineered features were used. F1-score changes ranged between 0.009 and 0.022, and ROC-AUC followed a similar pattern. RF and SVM showed the largest increases. LR also benefited from feature engineering, with an increase of 0.022 in ROC-AUC. These results suggest that engineered behavioural features better represent student learning behaviour compared to raw features. The comparison between baseline and engineered features is shown in Table 4.

Table 4: Comparison of baseline and engineered behavioural feature representations on model performance.

Model	F1 (Baseline)	ROC-AUC (Baseline)	F1 (Engineered)	ROC-AUC (Engineered)	Δ F1	Δ ROC-AUC
LR	0.829	0.860	0.840	0.882	+0.011	+0.022
RF	0.838	0.889	0.860	0.904	+0.022	+0.015
GBC	0.842	0.894	0.858	0.905	+0.016	+0.011
MLP	0.834	0.885	0.843	0.894	+0.009	+0.009
SVM	0.839	0.887	0.860	0.902	+0.021	+0.015
XGBoost	0.844	0.894	0.858	0.905	+0.014	+0.011

Source: Authors, (2026).

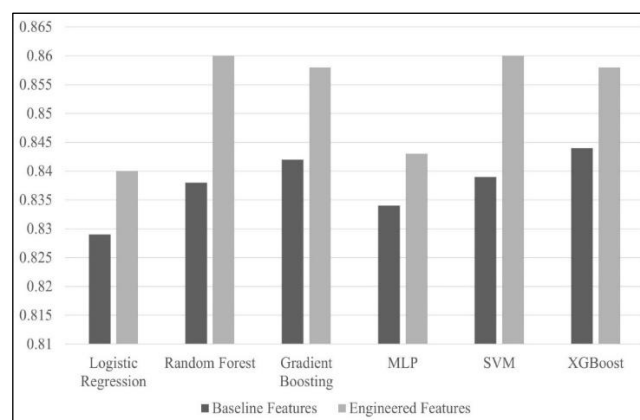


Figure 5: Comparison of baseline and engineered behavioural features using F1-score.
Source: Authors, (2026).

IV.3 STACKING ENSEMBLE PERFORMANCE

The authors then evaluated and compared a stacking ensemble model with individual classifiers. The ensemble achieved an accuracy of 0.819, an F1-score of 0.857, and a ROC-AUC of 0.905. These values are comparable to the best-performing individual models. While the ensemble did not produce the highest score in every metric, its performance remained stable across evaluation measures. Model results are reported in Table 5.

Table 5: Performance comparison between individual classifiers and stacking ensemble.

Model	Accuracy	F1-score	ROC-AUC
LR	0.798	0.84	0.882
RF	0.82	0.86	0.904
GBC	0.818	0.858	0.905
MLP	0.804	0.843	0.894
SVM	0.821	0.86	0.902
XGBoost	0.819	0.858	0.905
Stacking Ensemble	0.819	0.857	0.905

Source: Authors, (2026).

The ROC curve of the stacking ensemble is shown in Figure 6, where it can be seen that the ensemble maintains strong discrimination ability across threshold values. The ROC curve of the stacking ensemble is shown in Figure 7 and is compared with the best-performing individual model.

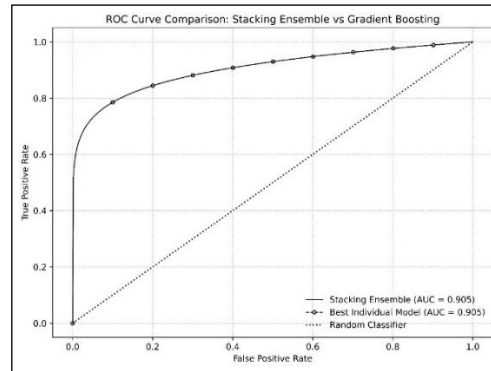


Figure 6: ROC curve obtained for the stacking ensemble.

Source: Authors, (2026).

To examine whether differences between the stacking ensemble and individual classifiers were consistent, the authors applied Wilcoxon signed-rank test. The Wilcoxon test showed clear differences between the ensemble and LR, Gradient Boosting, MLP, and XGBoost. In contrast, the comparison with RF and SVM did not show a clear difference. This indicates that the ensemble behaves similarly to some strong individual models, rather than outperforming all of them in every case. The detailed results are shown in Table 6.

Table 6: Wilcoxon signed-rank test results comparing the stacking ensemble with individual classifiers.

Compared Model	Wilcoxon Statistic	p-value	Significance ($\alpha = 0.05$)
LR	1,325,618	< 0.001	Significant
RF	1,462,668	0.492	Not Significant
GBC	1,400,911	0.014	Significant
MLP	1,394,889	0.008	Significant
SVM	1,432,827	0.122	Not Significant
XGBoost	1,377,959	0.002	Significant

Source: Authors, (2026).

This study examined whether behavioural information obtained from programming exercise submissions can support early identification of at-risk students in introductory programming courses in engineering and computer science programs. The results show that meaningful signals about student learning are already present in early-course activity, which supports findings from earlier work in programming education that report early difficulties as strong indicators of later outcomes [1-3]. This is especially relevant in introductory programming courses in engineering and computer science programs, where early difficulties tend to affect later progress. A key contribution of this work is the use of behavioural feature engineering to represent student learning more clearly. Raw log data, such as individual submissions or error counts, provides only a fragmented view of learning. When the authors combined programming activity at the student level and described using features such as completion, effort, errors, task difficulty, and timing, the learning behaviour became clearer. Previous work has shown that features derived from student behaviour provide a clearer picture of learning than raw system logs [7], [11], [12]. The consistent improvement in model performance after feature engineering indicates that these features reflect how students learn programming rather than how often they interact with the system. The comparison of individual machine learning models further highlights that programming learning behaviour cannot be fully captured using a single modelling approach.

The authors used LR mainly as a reference point, but it did not capture all learning patterns equally well. Tree-based and neural models performed a little better. This indicates that student learning behaviour involves interactions that are not purely linear, as also observed in earlier work on student performance prediction [17], [18], [20], [21]). These results reinforce the view that model performance depends strongly on feature design and data characteristics, rather than on the choice of a single “best” algorithm. The stacking ensemble produced stable and competitive results when compared with individual classifiers. Although it did not outperform every individual model across all metrics, its performance remained consistent across evaluation measures. In educational contexts, stable predictions are often more useful than small gains in accuracy when identifying at-risk students. Previous work has shown that ensemble approaches can combine the strengths of different models and are less affected by variations in the data [24-26]. The findings here also show that ensemble learning works well with different programming behaviours. Another strength of this work is the inclusion of statistical validation. Many studies in student performance prediction report performance metrics without testing whether observed differences are meaningful.

By applying the Wilcoxon signed-rank test, this study examined whether differences between models were consistent across validation folds. The results showed that while the ensemble differed from several individual models, strong classifiers such as RF and SVM remained comparable. These results show why statistical testing matters when comparing models. Earlier studies in educational data mining have also stressed this point [6], [11]. Statistical tests help check whether differences are reliable. They also help avoid conclusions based only on small performance changes. The findings further suggest that ensemble methods should be used to improve robustness. They should not be seen as a complete replacement for individual models. The dataset includes students from different computer science courses. It also covers multiple institutions and academic settings. This diversity shows that the framework is not tied to one course structure. It can be applied beyond a single teaching context. Since the framework relies on basic programming activities - such as the number of attempts, errors made, tasks completed, and when students engage with exercises - it can be applied across courses that use different curricula or assessment formats. Similar approaches that aim to work across courses rather than within a single setting have been encouraged in earlier studies on student performance prediction [15], [16], and the results of this study are consistent with those directions. Programming submission systems that are already used in courses can also help with learning, not only grading.

The activity data collected during regular coursework can help teachers notice students who are having difficulty, without asking them to take extra tests or fill out surveys. When such students are identified early, teachers can offer feedback, extra practice, or guidance at a time when it can still make a difference. Earlier studies have shown that this kind of timely support can improve learning outcomes in programming courses [8], [28]. This work contributes to programming education research by combining behavioural feature design, ensemble modelling, and statistical testing within a single study. This study does not focus only on final prediction scores. Instead, attention is given to identifying students at an early stage. The consistency of the results is also examined. The framework is considered across different course settings. Future work may look at real-time use of the framework. It may also examine the effects of specific teaching interventions. The current results provide a practical basis for using behavioural activity data to support learning in introductory programming courses in engineering and computer science programs.

V. CONCLUSIONS

This work proposed a machine learning framework for predicting student performance in introductory programming courses in engineering and computer science programs. The framework uses behavioural activity data collected from online programming submission systems. Submission logs from the early part of the course were analysed. These logs provided useful information about how students learn programming before final exams take place [1], [2]. Individual programming activities were grouped for each student. The authors converted the grouped data into behavioural features. These features described completion behaviour, effort levels, error patterns, task difficulty, and timing of engagement [15], [16]. The findings show that behavioural features are more useful than raw activity logs. Models trained with engineered features performed better than those using raw data. This confirms that learning behaviour is more informative than simple counts of submissions or errors [7], [11], [12]. Flexible machine learning models were better at capturing these learning patterns. The stacking ensemble showed steady and reliable performance across evaluation measures. This supports earlier findings reported in similar studies [17], [18], [25]. The authors used statistical testing to strengthen the comparison between models. The Wilcoxon signed-rank test examined whether observed differences were consistent. This helped confirm that improvements were not due to random variation. The results suggest that combining models can be more reliable than using a single approach [6], [11]. The authors collected the data from students enrolled in different engineering courses and institutions. These courses followed different academic structures. Because the framework relies on common programming activities, it is not limited to one curriculum or assessment style. This allows the approach to be applied in a wide range of educational settings [15], [16]. In practice, existing programming platforms can support early academic monitoring. This can be achieved without increasing workload for students or instructors. Behavioural data generated during normal coursework can help identify students who may need support while there is still time to intervene [8], [28].

VI. AUTHOR'S CONTRIBUTION

Conceptualization: Hetal Gosrani and Nirav Bhatt.

Methodology: Hetal Gosrani and Nirav Bhatt.

Investigation: Hetal Gosrani and Nirav Bhatt.

Discussion of results: Hetal Gosrani and Nirav Bhatt.

Writing – Original Draft: Hetal Gosrani and Nirav Bhatt.

Writing – Review and Editing: Hetal Gosrani and Nirav Bhatt.

Resources: Hetal Gosrani and Nirav Bhatt.

Supervision: Hetal Gosrani and Nirav Bhatt.

Approval of the final text: Hetal Gosrani and Nirav Bhatt.

VII. ACKNOWLEDGMENTS

The authors would like to acknowledge the support provided by their respective institutions for facilitating this research. The authors also thank the instructors and technical staff associated with the online programming environment for their assistance in data collection and system maintenance. The authors are grateful to all the students whose anonymized programming activity data contributed to this study.

VIII. REFERENCES

- [1] M. M. Jamjoom, E. A. Alabdulkareem, M. Hadjouni, F. K. Karim, and M. A. Qarh, "Early prediction for at-risk students in an introductory programming course based on student self-efficacy," *Informatica*, vol. 45, no. 6, Aug. 2021, doi: 10.31449/inf.v45i6.3528.
- [2] J. Llanos, V. A. Bucheli, and F. Restrepo-Calle, "Early prediction of student performance in CS1 programming courses," *PeerJ Comput. Sci.*, vol. 9, p. e1655, Oct. 2023, doi: 10.7717/peerj-cs.1655.

- [3] C. Van Petegem et al., "Pass/fail prediction in programming courses," *Journal of Educational Computing Research*, vol. 61, no. 1, pp. 68–95, Mar. 2023, doi: 10.1177/07356331221085595.
- [4] J. P. J. Brito Correia et al., "Predicting student performance in introductory programming courses," *Computers*, vol. 13, no. 9, p. 219, Sep. 2024, doi: 10.3390/computers13090219.
- [5] M. Yağcı, "Educational data mining: prediction of students' academic performance using machine learning algorithms," *Smart Learning Environments*, vol. 9, no. 1, p. 11, Mar. 2022, doi: 10.1186/s40561-022-00192-z.
- [6] Y. Zhang, Y. Yun, R. An, J. Cui, H. Dai, and X. Shang, "Educational data mining techniques for student performance prediction: method review and comparison analysis," *Front. Psychol.*, vol. 12, p. 698490, Dec. 2021, doi: 10.3389/fpsyg.2021.698490.
- [7] D. Song, H. Hong, and E. Y. Oh, "Applying computational analysis of novice learners' computer programming patterns to reveal self-regulated learning, computational thinking, and learning performance," *Comput. Human Behav.*, vol. 120, p. 106746, Jul. 2021, doi: 10.1016/j.chb.2021.106746.
- [8] A. K. Veerasamy, M. J. Laakso, and D. D'Souza, "Formative assessment tasks as indicators of student engagement for predicting at-risk students in programming courses," *Informatics in Education*, vol. 21, no. 2, pp. 375–393, Aug. 2022, doi: 10.15388/infedu.2022.15.
- [9] K. Chen et al., "ACcoding: A graph-based dataset for online judge programming," *Sci. Data*, vol. 11, no. 1, p. 548, May 2024, doi: 10.1038/s41597-024-03392-z.
- [10] R. Shi, J. Hu, and B. Lin, "Mining on Students' Execution Logs and Repairing Compilation Errors Based on Deep Learning," *Applied Sciences*, vol. 13, no. 17, p. 9933, Sep. 2023, doi: 10.3390/app13179933.
- [11] B. Albreiki, N. Zaki, and H. Alashwal, "A systematic literature review of students' performance prediction using machine learning techniques," *Educ. Sci. (Basel)*, vol. 11, no. 9, p. 552, Sep. 2021, doi: 10.3390/educsci11090552.
- [12] C. J. Arizmendi et al., "Predicting student outcomes using digital logs of learning behaviors: review, current standards, and suggestions for future work," *Behav. Res. Methods*, vol. 55, no. 6, pp. 3026–3054, Aug. 2022, doi: 10.3758/s13428-022-01939-9.
- [13] A. Çetinkaya, Ö. K. Baykan, and H. Kırız, "Analysis of Machine Learning Classification Approaches for Predicting Students' Programming Aptitude," *Sustainability*, vol. 15, no. 17, p. 12917, Aug. 2023, doi: 10.3390/su151712917.
- [14] I. Peraić and A. Grubišić, "Predicting Academic Performance of Students in a Computer Programming Course using Data Mining," *International journal of engineering education*, vol. 39, no. 4, pp. 836–844, 2023.
- [15] M. Riestra-González, M. del P. Paule-Ruiz, and F. Ortin, "Massive LMS log data analysis for the early prediction of course-agnostic student performance," *Comput. Educ.*, vol. 163, p. 104108, Apr. 2021, doi: 10.1016/j.compedu.2020.104108.
- [16] R. M. Santos and R. Henriques, "Accurate, timely, and portable: Course-agnostic early prediction of student performance from LMS logs," *Computers and Education: Artificial Intelligence*, vol. 5, p. 100175, Jan. 2023, doi: 10.1016/j.caeai.2023.100175.
- [17] E. Ahmed, "Student performance prediction using machine learning algorithms," *Applied Computational Intelligence and Soft Computing*, vol. 2024, no. 1, p. 4067721, Jan. 2024, doi: 10.1155/2024/4067721.
- [18] Z. Alamgir, H. Akram, S. Karim, and A. Wali, "Enhancing student performance prediction via educational data mining on academic data," *Informatics in Education*, vol. 23, no. 1, pp. 1–24, Mar. 2024, doi: 10.15388/infedu.2024.04.
- [19] D. Khairy, N. Alharbi, M. A. Amasha, M. F. Areed, S. Alkhalaf, and R. A. Abougalala, "Prediction of student exam performance using data mining classification algorithms," *Educ. Inf. Technol. (Dordr)*, vol. 29, no. 16, pp. 21621–21645, May 2024, doi: 10.1007/s10639-024-12619-w.
- [20] F. E. Arévalo-Cordovilla and M. Peña, "Comparative analysis of machine learning models for predicting student success in online programming courses: a study based on LMS data and external factors," *Mathematics*, vol. 12, no. 20, p. 3272, Oct. 2024, doi: 10.3390/math12203272.
- [21] Z. S. Hafdi and S. El Kafhali, "A comparative evaluation of machine learning methods for predicting student outcomes in coding courses," *AppliedMath*, vol. 5, no. 2, p. 75, Jun. 2025, doi: 10.3390/appliedmath5020075.
- [22] F. A. Al-azazi and M. Ghurab, "ANN-LSTM: A deep learning model for early student performance prediction in MOOC," *Heliyon*, vol. 9, no. 4, p. e15382, Apr. 2023, doi: 10.1016/j.heliyon.2023.e15382.
- [23] L. Vives et al., "Prediction of students' academic performance in the programming fundamentals course using long short-term memory neural networks," *IEEE Access*, vol. 12, pp. 5882–5898, 2024, doi: 10.1109/access.2024.3350169.
- [24] S. Malik et al., "Advancing educational data mining for enhanced student performance prediction: a fusion of feature selection algorithms and classification techniques with dynamic feature ensemble evolution," *Sci. Rep.*, vol. 15, no. 1, p. 8738, Mar. 2025, doi: 10.1038/s41598-025-92324-x.
- [25] A. Angeioplastis, J. Aliprantis, M. Konstantakis, and A. Tsimpiris, "Predicting student performance and enhancing learning outcomes: a data-driven approach using educational data mining techniques," *Computers*, vol. 14, no. 3, p. 83, Feb. 2025, doi: 10.3390/computers14030083.
- [26] M. Hoq, P. Brusilovsky, and B. Akram, "Explaining explainability: early performance prediction with student programming pattern profiling," *Journal of Educational Data Mining*, vol. 16, no. 2, pp. 115–148, Dec. 2024, doi: 10.5281/zenodo.14246435.
- [27] B. Bubnic, M. Memik, and T. Kosar, "Exploring the predictive potential of complex problem-solving in computing education: a case study in the introductory programming course," *Mathematics*, vol. 12, no. 11, p. 1655, May 2024, doi: 10.3390/math12111655.
- [28] P. Utamachant, C. Anutariya, and S. Pongnumkul, "i-Ntervene: applying an evidence-based learning analytics intervention to support computer programming instruction," *Smart Learning Environments*, vol. 10, no. 1, p. 37, Jul. 2023, doi: 10.1186/s40561-023-00257-7.