Chapter 2

Literature Review

In this section different research publications related to study are arranged according to its logical flow of study.

2.1 Online Coding Platform for Learner

Online coding environments are now an essential component in the programming education process, which gives students flexibility to code, debug, and receive feedback remotely. Abbeel and Ng (2004) conceptually established adaptive systems when they introduced apprenticeship learning via inverse reinforcement learning, thereby enabling platforms to mimic expert behavior [1]. Cedazo et al. (2015) presents an online C compiler with integrated self-assessment features intended to improve outcomes in programming education. It allows students to black-box test their software functionalities, thus permitting them to test their own code independently and receive feedback in the form of error messages and hints. This increases participation because the process of debugging is more engaging. Scalability is cited as an advantage, and this is relevant for large classrooms. Use of adaptive assessments identifies common patterns of error among students. Their methodology included empirical testing on students at the level of university and demonstrated substantial improvements in debugging and comprehension. This paper limits their handling of extremely complex code but has provisions for the next version and needs visualization tools that support the process of tracking error. This research provides a foundational approach to developing scalable and interactive programming environments, thus directly applicable to online education systems [2].

Das et al. (2016) introduce Prutor, a cloud-based programming tutor that addresses the problems of large-scale programming education. Prutor collects real-time data on student submissions and provides targeted feedback based on individual performance. The platform is ascendable which makes it possible for instructors to handle large numbers of submissions effectively. The study involves an assessment of Prutor on the students' learning outcome, and it has greatly improved their comprehension and debugging skills. The interactive interface provided by the prutor helps learners to find

and fix programming mistakes without help from the instructor. The authors note that the incorporation of machine learning to predict common error patterns is critical. Ambiguity in code submission or overly complex submissions are some of the limitations the authors acknowledged with future solutions in mind. Prutor stands as an excellent example of adaptive and scalable programming education tools [3].

Gupta et al. (2017) introduces, an AI-powered system DeepFix that automatically repairs common syntax and logical errors in the C programming code. Through deep learning models, these authors analyzed and corrected coding errors, which reduced debugging times for novice programmers. DeepFix is equipped with explainable feedback mechanisms that guide learners into their mistakes, creating even deeper understanding. This was demonstrated by the testing of this system on a dataset comprising students' submissions of programmed solutions, indicating high precision in error detection and correction. Some other challenges arising with ambiguous code were successfully overcome using hierarchical models. This research therefore points towards an integration with AI where it would greatly reduce frustrations of students about programming subjects, hence bringing retention. Repetitive debugging has been automated for DeepFix so that lecturers do more conceptual teaching. This is one of the pioneer works applying deep learning techniques in the programming education context [4].

Rivers and Koedinger (2015) addresses the area of self-improving Python tutor in student interaction data generating contextual hints. This system recommends with dynamics based on a change in student performance that has proven to offer customized learning for students. Through demonstrations, the authors exhibited the potential of improving solving tasks as complex as simple step breakdowns. The students using the tutor achieved much higher retention and understanding in comparison with traditional approaches. Scalable adaptive systems offer an approach to fulfilling diversified learner needs. There remain some integration difficulties when introducing the tutor into other languages. The study further proposed future work in terms of extending the adaptability of the tutor and the feature of peer learning. This system is a great example of how data-driven feedback can be used in the practical implementation of programming education [5]. Nguyen et al. (2014) introduces Codewebs: a scalable

system that makes collaborative learning in programming easy by enabling students to find similar coding solutions. The method employed by Codewebs to improve error correction and task understanding in students is search-based. The authors present a semantic matching framework that will allow the detection of coding patterns and, hence, facilitate the overcoming of misconceptions by the learner. A user-friendly interface for the system motivates students to find alternative solutions and helps inculcate a collaborative problem-solving culture. Additionally, the authors conducted a massive evaluation, showing that considerable benefits accrue to students in terms of performance and engagement. Efficient indexing algorithms ensured scalability and faster processing. By bridging the gap between individualized learning and collaborative approaches, Codewebs offers a novel model for online programming education. This research forms an important foundation for future systems that will balance autonomy with peer collaboration [6]. Liao et al. (2016) developed a selective syntactic compiler for delivering personalized error feedback tailored to promote engagement and learnability effects. Novice programmers experience different kinds of difficulties, including debugging and conceptual confusion difficulties [7].

Robins (2019) presents these issues with an orientation toward direction that should be addressed through an articulated sequence of learning [8]. Piech et al. (2012) proposes a machine learning-based approach for modeling student learning trajectories over programming assignments. Applying Hidden Markov Models, the authors graphically instantiated the student development pathways and uncovered patterns that best predict subsequent performance. This study proved that temporal developmental data better predicts learning success than the final grades obtained. This strategy supports early intervention by categorizing students into performance-based groups, allowing teachers to have actionable insights about how students are progressing [9]. Sukanya and Albert (2023) proposed new approaches to isolate novice-specific errors in support of early learners [10].

2.2 Security Concerns in Remote Code Execution

The main threats are security issues relating to remote code execution. Alhothali et al. (2022) reviews vulnerabilities in RCE systems, especially on educational platforms. The authors identified main risk factors: weak mechanisms of isolation and improper

handling of sessions. To prevent such risks, the paper proposed robust sandboxing techniques that isolate user sessions from unauthorized access. The evaluation demonstrated the effectiveness of these techniques in reducing attack vectors. Challenges such as maintaining performance under heavy usage were also addressed. The research focuses on integrating advanced security protocols for the protection of online learning platforms. This work is specifically relevant in designing secure, scalable environments for programming education [11]. Devi et al. (2011) suggested the technique of sandboxing that ensures isolation in the containment of the untrusted code to ensure secure environments [12]. Liao et al. (2019) presents the lightweight machine learning methodology used to forecast low-performing students early in the term using clicker data from Peer Instruction courses. The researchers were able to achieve a 62% success rate using Support Vector Machines (SVM). The group demonstrates its robustness with validation across several terms and institutions. They emphasize the early intervention point and lightweight data collection methodologies that can scale to massive classes. This research would thus align with programming education, which calls for the need for data-driven approaches to identify those learners at risk [13].

Kim et al. (2020) addresses security concerns in online coding platforms using compiler-assisted encryption techniques. The authors proposed a framework that prevents remote code execution to include a check for security in the path of execution. Their core work is real-time user submission analysis since it uses a combination of static and dynamic monitoring, thus including the two methodologies to provide assurance at all levels. Major improvements in detection and mitigation of various security risks were noticed on such a large dataset. Scalability for very large datasets and addressing sophisticated attack vectors are challenges. This research sets a precedent for incorporating security measures into online programming environments, making safe learning spaces for students [14].

Murpy et al. (2009) suggested Retina, which is a detection tool that analyzes suspicious student submissions to ensure safety and integrity of academics and systems [15]. Li et al. (2020) has also proposed an illustration of applying deep learning models to identify and mitigate RCE attacks in real-time by providing AI-based security solutions for

online platforms [16]. Restrepo-Calle et al. (2018) also came up with an interactive learning and automated assessment of programming skills in the form of UNCode. It offers both summative and formative feedback through automated grading tools. Summative feedback grades the programming solutions on syntax, semantics, and efficiency. The formative feedback provides explanations of errors and hints. This system solves the problem of manual evaluation that takes a lot of time and is not consistent. It uses static and dynamic analysis to evaluate code functionality and behavior. UNCode was built mainly for engineering students but supports iterative learning where students may perfect solutions based on feedback provided. The paper gives out its effectiveness in developing engagement and skills in the learners [17].

2.3 Collecting Learners' Programming Data

The backbone of the adaptive programming education system is the collection of learner data and its subsequent analyses. According to Ala-Mutka (2005), there is a need to collect and analyze granular information such as the kind of errors, times to perform tasks, and submission history to feed into the predictive model [18]. Ihantola et al. (2015) reviewed preprocessing techniques for educational data mining by putting emphasis on how structured and clean data was more probable to extract meaningful ideas [19]. Watson et al. (2013) focus on logging students' interactions in programming tasks to determine performance trends and learning difficulties. The authors proposed a framework for log data collection and analysis revealing insights into common errors and task completion strategies. Their findings indicate that logged data can inform predictive models to allow educators to intervene at the right time. They covered issues like data privacy and processing of massive logs. The research identifies the need for data gathering with detail in programming education [20].

Denny et al. (2012) categorized syntax errors to determine novice programmers' issues and create automated feedback systems [21]. Bilegjargal and Hsueh (2021) analyzed the adoption and efficacy of online judge systems in programming education. The authors used structural equation modeling to analyze the attitudes students have toward features such as real-time feedback, error analysis, and task recommendation. The results showed learners who used online judge systems improved their problem-solving skill and reduced debugging time significantly. The study also outlined the importance

of user interfaces in enhancing student engagement. Challenges such as system scalability and accuracy of data logging were identified, along with recommendations for future improvements. This study displays the importance of having such systems in adaptive learning [22]. Ghosh et al. (2024) used computational thinking in visual programming with task-specific data to customize pathways for novice learners [23].

2.4 Feature Engineering on Dataset

Feature engineering is one of the major steps in transforming raw learner data into actionable inputs for machine learning models. According to Rivers and Koedinger (2015), some of the significant features that were added to the models included error patterns, task complexity, and time spent on a particular task that would signify student success [5]. Liao et al. (2016) presented lightweight feature selection methods in order to optimize the performance of the model while not compromising the interpretability [7]. Lin et al. (2019) utilized dimensionality reduction techniques like PCA to manage high-dimensional data, losing less information but increasing computational efficiency [24]. Yadav and Pal (2012) explore data mining techniques in predicting the academic performance of engineering students. The authors used classification algorithms such as Decision Trees and Naive Bayes to identify at-risk students. Through historical performance metrics and grades, the study discovered some important predictors of academic success; attendance and prior coursework stand out as predictors. The findings have improved intervention timing, so that educators can start to assist students earlier if they are having trouble in the class. This paper provides a basis for using data-driven techniques to enhance the teaching process, especially for technical courses like programming [25].

Gupta et al. (2017) demonstrated how deep learning models can automatically extract meaningful features from complex programming datasets for enhancing the accuracy of prediction [4]. Shen et al. (2022) focuses on the use of profiling techniques for predicting programming performance, relying on granular data such as error patterns and task completion times. The authors developed a machine learning model that achieved high accuracy in identifying at-risk learners. Their system also gave personalized suggestions to assist students in overcoming obstacles. The paper highlights the following limitations: The datasets are small, which calls for larger

datasets to increase model generalizability. This research underlines the need for early interventions in programming education to improve learning outcomes. This work is a good contribution to the field of predictive analytics in education [26]. Marjan et al. (2021) designed feature engineering techniques specific to programming education, which capture domain-specific nuances well [27].

2.5 Applying Different Machine Learning Algorithms

Sharma and Harkishan (2022) proposed an intelligent tutoring system for programming education incorporating real-time feedback and adaptive learning pathways. The authors utilized machine learning algorithms in the analysis of student interactions and the prediction of trends of performance, thus recommending personal tasks. The results showed considerable improvements in engagement and retention rates among students. The scalability and adaptability of the system were identified as some of its strengths and facilitated its use in diverse educational settings. Some of the discussed challenges include integrating cross-platform support, with ideas on future work. The work thus makes a compelling argument for the inclusion of intelligent systems in programming education [28]. Modi et al. (2024) demonstrate diversity in machine learning models was extensive, including K-Nearest Neighbors (KNN), Decision Trees, Logistic Regression, XGBoost, Random Forest, and Deep Neural Networks (DNN). These models suggest that Random Forest and XGBoost outperform, as they handle noise in the data quite well and could capture the more intricate patterns existing in the data. That is why the ensemble learning methodology is so robust. With this regard, ensemble models can therefore address the complexity and variability held in novice programming datasets, while maintaining high accuracy in modeling learning behavior [29].

Breiman (2001) is one of the earlier innovators of introducing robust ensemble methods commonly used in educational data mining, including Random Forest [30]. Buenaño-Fernández et al. (2019) uses ensemble learning techniques, such as Random Forest and Gradient Boosting, to predict performance in programming. It further showed how the application of multiple algorithms enhances accuracy in prediction and robustness in prediction. The method is based on the large amount of student submissions and attempts to identify trends of performances. The study therefore demonstrated how

ensemble methods can handle noisy high dimensional data. Challenges would lie in the computational overhead incurred in training ensemble models in large datasets. Such studies demonstrate the effectiveness of using ensemble methods in establishing sound educational analytics systems [31].

Chen and Guestrin (2016) presents XGBoost, which is an optimized gradient boosting algorithm that has become the benchmark in predictive modeling. The authors have pointed out its scalability to handle sparse and high-dimensional data and its suitability for educational data mining. The evaluation proved significant performance improvement over traditional boosting methods with applications that range from predicting student performance to adaptive task recommendations. The interpretability features of feature importance scores make XGBoost very useful for teaching. The paper ends by mentioning possible extensions, namely the support for deep learning integrations [32]. Altabrawee et al. (2019) checked the classification techniques of SVM and KNN in modelling student performance [33].

Wang et al. (2017) applied deep knowledge tracing for programming exercises with a demonstration of the necessity of sequential data analysis as it relates to learning performance [34]. Cabo et al. (2021) studies the application of machine learning in predicting the performance of engineering students in programming courses. The authors modeled student outcomes as decision trees and neural networks over the variables of time-on-task, submission frequency, and error patterns. The authors found that the neural networks successfully identified the at-risk students and that decision trees offered an interpretability benefit for instructors. Their work, too put great emphasis on the idea that domain-specific feature engineering played a major role for boosting accuracy in prediction. The authors end by calling for embedding predictive models within the programming education systems in order to create early intervention and tailored support [35].

An extensive review of different algorithms implemented in educational data mining over the effectiveness of those techniques in predicting student performance was well delivered by Alsariera et al. (2022). A comparative analysis over multiple datasets between supervised, unsupervised, and ensemble methods showed that ensemble-based

approaches such as Random Forest and Gradient Boosting are significantly more accurate and scalable as compared to others. According to the authors, the selection of a number of features was essential mainly to improve the interpretability of the model. Most of the issues were related to working with noisy and imbalanced datasets as well as possible preprocessing strategies, so the results indicate an importance of robust algorithms in adaptive learning environments, making the study be used as a point of reference for the use of ML in programming education [36]. Rokach and Maimon (2005) gives an exhaustive review of Decision Trees as a classifier that covers widely used algorithms like C4.5 and CART. The authors explain splitting criteria: Information Gain and Gini Index, as well as advance pruning techniques to enhance accuracy of trees. They note that DTs are quite simple and interpretable so suitable for educational applications, like predicting student performance. Challenges, such as those from imbalanced datasets, are discussed, and the hybrid technique is proposed in order to improve robustness. This is a foundational piece for feature selection and classification within educational data mining [37].

Gupta et al. (2019) extends reinforcement learning to programming error correction in terms of syntactic and logical errors. The authors designed an RL-based system that can automatically detect and correct errors while giving the learner explanations. The findings were that reinforcement learning models performed better than rule-based systems in adapting to the different error scenarios. Challenges involved include the computationally expensive cost of training RL models and their dependency on large, labeled datasets. The authors proposed hybrid approaches to address these issues. This paper highlights the autonomous systems potential for improving education in programming [38]. Pires et al. (2024) explores the applicability of long short-term memory networks to the analysis of time-related patterns in programming data. The authors showed in their work how LSTM can be used to predict accurately student performance, especially concerning sequential learning tasks. The evaluation portrayed how temporal models outclass their traditional foils in the capture of the learning trajectory and that challenges are at the computational complexity and being adjusted to the domain concerned. This research opens pathways in educational data mining that might be explored by advanced neural networks for adaptive learning systems [39].

2.6 Proposed Solutions and Recommendations

Gupta et al. (2017) stated that task recommendations must be aligned with the learner's profile, which is the core concept of an ensemble algorithm [4]. Rivers and Koedinger (2015) established the success of hint generation systems in dynamic adaptation according to the needs of the students, which is related to complexity-aware weighting [5]. Durak and Bulut (2023) review classification and prediction-based machine learning algorithms particularly for educational applications with the focus on programming education. The authors analyzed the commonly used algorithms, such as logistic regression, XGBoost, and Random Forest, and have shown that ensemble techniques are working effectively at growing accuracy and robustness for diverse datasets. Hybrid models combining different techniques were also indicated as promising for better generalizability. The authors noted that computational complexity and the need for large datasets were still significant barriers. Yet, the findings of the study highlighted the importance of algorithm optimization in adaptive learning systems [40]. Jokhan et al. (2022) discussed how student performance can be predicted using AI in higher education studies. The subject of discussion was the study concerning the sustainable development goals. Here, the authors used the Random Forest classification model for predicting performance, achieving a 97.03% accuracy level in just six weeks. The virtual education system improved because of the application of this early intervention model against the background of the COVID-19 pandemic. The paper highlighted the potential to analyze digital interactions and use that information to optimize the teaching strategy, thus achieving equitable education. The interplay of analytics and pedagogy in the paper underlines the transformative impact AI has on education [41].

Gupta et al. (2019) introduce reinforcement learning in error correction systems by showing that adaptive algorithms could optimize a real-time experience of learning [38]. Shen et al. (2022) established that complexity-aware models do function to adjust the dynamic pathway of learning based on not only difficulty in a given task but also based on the learners' performance [26]. Parihar et al. (2017) created an automatic grading system with introductory programming courses that employs the use of program repair techniques. Leverage AI-based repair models and assess student submissions while offering detailed feedback. Methodologies enhance grading

efficiency, personalized guidance, and results yield substantial reductions in grading time without sacrificing feedback quality. The research emphasizes the great potential of integrating AI in automated educational assessments that aim to improve the scalability of programming education, such as in your project [42]. Jayasree and Asim (2021) Suggested predictive models for prediction of student performance in the context of MOOC. The authors used the regression analysis and supervised ML models, such as Random Forest and Gradient Boosting models, on the OULAD dataset. The model identified behavioral, temporal, and demographic features as some of the most significant predictors. The study had great accuracy, with the gradient boosting model performing the best in final performance predictions. By identifying high-risk students early, the models facilitate timely interventions. This research highlights the role of ML in improving student outcomes in online education [43].

Kotsiantis et al. (2004) use machine learning techniques to anticipate student's performances in learning environments at distances. The writers used data drawn from the Hellenic Open University to perform experiments and run algorithms: Naive Bayes, Decision Trees, and Support Vector Machines. Naive Bayes had been demonstrated to be particularly the most acceptable due to high accuracy scores and even a simple interface. As proved by these results, demographics and grades of assignment accurately classify student performance. This approach enables early detection of vulnerable learners to which tutors provide support in an appropriate way. It also calls upon the strength of robust techniques of machine learning for boosting distance education effectiveness [44].

In programming education, evaluating models of machine learning demands inclusive methodologies. According to Breiman (2001), the technique of using the metrics precision, recall, and F1-score along with the qualitative information helps to perform a thorough evaluation of model performance [30]. Rubio (2020) utilized trajectory analysis to predict novice success and provide actionable insights in the refinement of adaptive learning systems [45]. Liao et al. (2016) combines clicker response data and early-term programming assessments to predict student performance in introductory courses. The methodology involves the creation of predictive models capable of identifying struggling students by the third week of the course. This study showed

scalable, real-time intervention in programming education with an accuracy of up to 70%. Moreover, it shows that interaction data with dynamic students could complement predictive frameworks, hence rendering it as a precious asset of adaptive learning systems [7].

For instance, Pires et al. (2024) presented evidence of deep models such as long short term memory networks that analyze sequential program data for better accuracy of predicting [39]. Knowledge tracing has been applied to analyze the programming exercise for continuous learner assessment, such as in Wang et al. (2017). These results present strong arguments for dynamic and adaptive systems with ensemble algorithm as an approach to catering for different learner needs [34]. Moonsamy et al. (2021) did a meta-analysis of Educational Data Mining (EDM) techniques for predicting student performance in programming. PRISMA methodology was used, where 11 studies are analyzed and it is discovered that ensemble methods such as Random Forest performed with the highest prediction task accuracy. The research revealed the heterogeneity of performance from the algorithms due to data source variability and variations in preprocessing techniques. Potential for EDM in early student-at-risk detection and timely interventions is suggested. Data inconsistencies and publication bias were other discussed issues with recommendations such as standardization of the dataset and methodologies. This meta-analysis confirms the need for EDM in dealing with multiple educational problems [46].

Sehaba (2020) demonstrated usage of machine learning algorithms in educational technology for predictive analytics toward the improvement of student learning outcomes. It discusses the assessment framework of student performance with the use of classification models, thereby laying much emphasis on data preprocessing, feature selection, and optimization of the algorithm. Here, different machine learning algorithms have been compared for predicting the student's performance based on effectiveness-Decision Trees, Random Forest, and Gradient Boosting. This also underscores feature engineering, derived attributes, ensemble learning techniques, and XAI techniques in model improvement. This research indicates predictive analytics is able to predict the risky learners as early as possible so that interventions could be given just in time and individual support can be provided to at-risk learners. It suggests that

such predictive and adaptive technologies improve the novice programming learning experiences [47]. Mustapha (2023) discussed advanced applications of artificial intelligence and machine learning in education, it seeks to focus on adaptive learning systems that cater to diverse needs within learners. It underlines the importance of data preprocessing and feature engineering for correct predictions in educational datasets where raw data is transformed into actionable insights. Ensemble learning methods, like Random Forest and Gradient Boosting, are used with the ability to deal with imbalanced and noisy datasets, predict student success, and identify at-risk learners at an early stage. The paper advocates for the use of semi-supervised learning techniques to deal with incomplete or sparse labels in educational data, demonstrating how they may provide robust predictions in resource-constrained settings. Explainable AI (XAI) techniques are essential to incorporate into educational systems to enable interpretable results, such that educators can identify what is not working and come up with appropriate teaching strategies. This paper contributes to the understanding of how the advanced AI techniques can benefit the learning experience of novice programmers and lead to a better educational outcome [48].

Brooks et al. (2023) explores how artificial intelligence and machine learning can be applied in education systems, focusing on individualized learning paths and sophisticated analytics. It underlines the need for AI-enabled systems to analyze learner behaviors, predict performance, and adaptively recommend content. Another important point raised by this study is that robust preprocessing of data and feature engineering are required to ensure educational datasets' quality and reliability. Ensemble learning techniques such as Random Forest and Gradient Boosting are discussed, along with explainable AI (XAI) for transparency and informed decision-making. These results are congruent with studies about finding the learning paths for novice programmers [49]. Ouahi et al. (2024) presents a literature review on the application of machine learning techniques to predict learner outcomes in online training courses. The objective is to provide a summary of the latest models developed for the purposes of forecasting student performance, categorical coding methodologies, and the datasets used. The study runs experiments to test the proposed models against each other as well as against some prediction methods with other machine learning algorithms side by side. The experimental results indicate that using encoding method for preprocessing categorical

variables improves the performance of deep learning architectures. Indeed, with the help of long short-term memory networks, it performs very well on the problem under investigation [50].

Ahadi (2016) examines the application of machine learning to predict difficulties that a novice programmer would encounter early in their learning. The research shows how predictive models can analyze coding patterns and error behaviors to intervene early, thereby improving learning outcomes and engagement [51]. Ahmed et al. (2019) propose an innovative method to generate error-specific examples that assist novice programmers in debugging compilation errors. The system uses machine learning and program analysis techniques that generate educationally valuable, error-specific examples that help in understanding and correcting coding faults. The experiment shows targeted feedback enhances learning efficiency as well as decreases the tendency of error repetition, helping in more efficient programming training [52]. Alalawi et al. (2023) provides a systematic review of machine learning applications in predicting student performance highlighting trends, challenges, and opportunities in the field. The importance is on robust feature engineering, good algorithm selection, and attending to data quality issues towards improving the accuracy of such predictions. It provides useful information on how ML can be used to enhance the experiences of personalized learning and early detection of at-risk students' which lines up well with the work in programming education research [53]. Anand et al. (2018) presents a recursive clustering method to assess students' performance in programming courses. The technique uses clustering for students along with key performance indicators like time taken for completing the tasks and error rates for patterns and profiling learners into levels of proficiency. This method will help teachers understand student learning behavior more clearly and guide interventions for such students. The research thus proves the utility of clustering in managing different learning needs in programming education [54].

Baker and Yacef in 2009 have outlined the state of EDM until 2009, elaborating on its applications for understanding and improvement of learning processes. The discussion focuses on some key techniques of clustering, classification, and sequential pattern mining for the analysis of educational datasets. The author stresses that EDM can

facilitate personalized learning, predict student performance, and identify at-risk learners. The authors also give direction for the future, which entails the integration of real-time analytics and further EDM adoption in more diverse educational contexts [55]. Bergin and Reilly (2006) reported a multi-institutional study that used a multivariate approach to predict the student performance in introductory programming courses. The research determined various factors that influence success; these factors include prior experience with programming, mathematical aptitude, and study habits. By using statistical and machine learning techniques, it could be shown that both academic and behavioral variables may serve as predictors of programming outcomes. Thus, the research calls for an understanding of various characteristics of learners to effectively formulate programming education strategies [56].

Chen and Ding (2023) propose a framework for predicting academic performance in the schools of Pennsylvania based on machine learning. Algorithms like Random Forest, XGBoost, and Support Vector Machines are employed to analyze factors such as socio-economic status, attendance, and previous academic records. The findings revealed that ensemble methods are quite useful in achieving high accuracy. The study stresses the need for feature selection and data preprocessing to improve model performance and offers useful insights for educational policy-making and personalized interventions [57]. Guo et al. (2003) provide an in-depth study of the K-Nearest Neighbors algorithm as a model-based approach in classification tasks. The paper discusses improvements over traditional KNN, including feature weighting and distance metrics, to enhance accuracy and robustness. The authors demonstrate the algorithm's applicability to high-dimensional and noisy datasets, which shows its flexibility in diverse scenarios. This paper, while acknowledging the simplicity and efficiency of KNN, also addresses the problem of the sensitivity of this algorithm towards irrelevant features with optimized parameter tuning. It lays a foundational insight in using KNN for educational and other classification problems [58].

Hosseini et al. (2017) explores stereotype modeling to predict the performance of problem solving in both MOOCs and traditional classes. The authors show that learners can be clustered into groups based on demographic and behavioral characteristics, which can allow for effective forecasting of student outcomes with stereotype-based

models. The authors point to the utility of such models in identifying at-risk learners and personalizing educational content. The research underlines the significance of using user profiles to improve prediction accuracy, providing useful insights into adaptive learning environments in programming education [59]. Llanos et al. (2023) focus on early prediction of student performance in introductory programming (CS1) courses using machine learning techniques. The paper compares models, such as Random Forest, Support Vector Machines, and Logistic Regression, to predict student outcomes from the initial performance and behavioral data. Its findings underscore the value of early predictors: assignment completion rates and error patterns for at-risk students' identification. The research underlines the need for proactive intervention strategies to enhance learning outcomes, which closely aligns with efforts to personalize programming education [60].

Samonte et al. (2024) presents an adaptation of the DAS3H model to create a personalized distributed practice schedule aimed at enhancing long-term memorization in an intelligent programming language tutor. By integrating distributed practice techniques into programming education, the study demonstrates improvements in retention and learning outcomes. The model dynamically adjusts practice schedules based on individual performance and learning behaviors, providing a tailored approach to skill development. This research highlights the potential of combining cognitive science principles with intelligent tutoring systems to optimize programming education for novice learners [61]. Effenberger and Pelánek (2019) explore how to measure students' performance on programming tasks through the analysis of completion rates, error patterns, and time spent. This study underlines the necessity of designing programming tasks in a way that would really differentiate between students' skill levels while providing meaningful feedback. By using data-driven metrics, the authors show how the assessment of performance can be enhanced to identify learning gaps and guide personalized interventions. This study offers some very interesting insights into monitoring the development of novice programmers and making the programming curriculum individualistic [62].