From Typing to Insights: An Interactive Code Visualization and Error Detection for Enhanced Student Support Using Keystroke Data

Friday Emmanuel James, Kansas State University

Friday James is a PhD Candidate at Kansas State University. He has a double-majored Bachelor's degree in Statistics/Computer Science from University of Agriculture, Makurdi - Nigeria. He got a Master's degree in Statistics and a Master's degree in Computer Science from University of Ilorin - Nigeria and Kansas State University - Kansas USA in 2015 and 2021 respectively. His research interest cuts across the use of machine learning and data science in Computing Science Education to improve teaching and learning.

Joshua Levi Weese, Kansas State University

Dr. Josh Weese is a Teaching Assistant Professor at Kansas State University in the department of Computer Science. Dr. Weese joined K-State as faculty in the Fall of 2017. He has expertise in data science, software engineering, web technologies, computer science education research, and primary and secondary outreach programs. Dr. Weese has been a highly active member in advocating for computer science education in Kansas including PK-12 model standards in 2019 with an implementation guide the following year. Work on CS teacher endorsement standards are also being developed. Dr. Weese has developed, organized and led activities for several outreach programs for K-12 impacting well more than 4,000 students.

Russell Feldhausen, Kansas State University

Russell Feldhausen received a bachelor's degree in computer science in 2008, and a master's degree in computer science in 2018, both from Kansas State University. He is currently pursuing a doctorate in computer science with a focus on computer science education, also at K-State. Feldhausen's research interest is computer science education, targeting rural populations and exploring ways to integrate mastery learning into CS curricula. He is also actively involved in many K-12 outreach programs providing curricula and teacher training throughout Kansas.

Nathan H Bean, Kansas State University

Dr. Nathan Bean is a Teaching Associate Professor at Kansas State University Department of Computer Science and Co-Director of the Advancing Learning and Teaching in Computer Science (ALT+CS) Lab. His research is focused on the need to grow the body of students skilled in computing – both within the field of Computer Science, and within other disciplines that increasingly rely on the tools computer science makes available to advance their own work. Thus, his research involves investigations into how to effectively reach a broader and more diverse audience of students, and developing pedagogical techniques and technologies that allow it to be done at scale.

Dr. Michelle Friend, University of Nebraska - Omaha

Dr. Michelle Friend is an Associate Professor in the Teacher Education Department at the University of Nebraska at Omaha. She teaches CS teaching methods and research methods. Her research focuses on equity in computer science and interdisciplinary connections between computer science and other subjects. She received her Ph.D. from Stanford University in Learning Science and Technology Design, and previously taught middle school computer science.

Dr. David S. Allen, Kansas State University

David is an Associate Professor in the Department of Curriculum and Instruction at Kansas State University and the Director of the Center for STEAM Education. His work involves professional development for K-12 schools in STEAM related areas, and he is currently focused on on-line programing development in mathematics and computer science education.

From Typing to Insights: An Interactive Code Visualization and Error Detection for Enhanced Student Support Using Keystroke Data

Abstract

The continuous rise of digital learning platforms has unarguably brought about a surge in the amount of data obtained from different learning environments. This, in turn, presents a great opportunity for computer science teachers to gain an understanding of students' coding processes, which is vital for enhancing student support. Traditional assessment methods, such as feedback after homework submissions or completed lab assignments, often result in late and untimely intervention that would prevent early dropouts and high failure rates as the students' learning journey is neglected in the process. This paper presents an analytical tool that leverages students' keystroke data to improve teaching and learning in introductory programming courses. Using fine-grained data from Python and Java-based assignments delivered through the Codio learning platform – covering 3 semesters and having 13 programming assignments each, the tool reconstructs students' code progression. The tool also integrates unit tests and execution logs to deliver immediate feedback on code correctness, track error correction times, and highlights students' debugging strategies, thereby offering instructors a comprehensive view of the students' programming process. It employs a mixed-method approach, combining quantitative analysis of keystroke data with qualitative insights into students' problem solving approaches. Pedagogically, the tool helps shift the focus from traditional grade-based assessments to a data-driven approach of identifying struggling students. Insights from the tool inform curriculum design and allow educators to address common challenges and refine teaching methods. This work contributes to computer science education by demonstrating the potential of data-driven methods to enhance students' learning outcomes. The findings highlight the importance of fine-grained analytics in understanding behaviors of novice programmers, thereby paving the way for adoption of such tools in existing educational management systems. This research underscores the impact of integrating analytics into programming education by bridging the gap between raw coding data and actionable insights.

1 INTRODUCTION

In the field of Computer Science Education (CS Ed), programming assignments and projects play a crucial role in fostering students' problem-solving skills, computational thinking, and competence. However, for many students, particularly inexperienced ones, programming can be a difficult journey that is characterized by trial and error, moments of confusion, and periods of frustration. Computer science educators often resort to assignments and examination scores to assess students' level of understanding, which often results in late

and untimely intervention that could prevent early dropouts and high failure rates. Early identification of when and why students are struggling during the process of coding is essential for both timely intervention and effective teaching. This is particularly important for large classes, where individualized attention is practically impossible.

To address this problem, we develop and evaluate an analytical tool that leverages keystroke data to identify struggling students by reconstructing and visualization their coding process. The basis for the functionality of the tool is the interaction of students with their programming assignments through keystroke dynamics, which will offer significant insight into their thought process and general coding habits.

Keystroke data, which captures granular details about students' coding process such as insertions and deletions, offer a great wealth of information for understanding how students approach problem solving. The application also tests students' code snippets against unit tests to evaluate code correctness and functionality. This work therefore seeks to provide answers to the following research questions:

- **RQ1:** How can keystroke data be effectively utilized to understand analyze students' programming behaviors and identify struggling students for early intervention?
- **RQ2:** How can code execution logs and error reports enhance the teaching and learning experience in programming courses?

By addressing these questions, this research aims to contribute to the broad issue of data-driven interventions in CS education by providing actionable insights for both educators and learners.

2 BACKGROUND AND RELATED WORK

2.1 Overview Of Learning Analytics

Learning Analytics is the measurement, collection, analysis, and reporting of data about learners and their contexts for the purpose of understanding and optimizing learning and the environments in which it occurs [1]. Its emergence is a response to the need for a more data-driven approach to education by integrating different disciplines such as machine learning, cognitive psychology, data science and statistics [2]. The overall goal of learning analytics is to gather educational data and utilize the insightful information obtained from the data to enhance learning, provide actionable feedback to students, and guide educators toward providing intervention to struggling students [3]. It has developed into an essential tool used by educational instit utions to enhance both student learning and how educators provide intervention through the use of data mining, machine learning, and visualizations [2]. Thus, by leveraging data mining techniques, institutions of higher education can uncover patterns hidden in student data to provide assistance to students at risk of under-performing [4].

Beyond data mining approaches, educational institutions can leverage the power of deep learning techniques in learning analytics. Supervised machine learning algorithms, such as Support Vector Machines (SVM), Naive Bayes, Decision Trees, as well as unsupervised machine learning algorithms such as clustering, are useful frameworks for predicting student

performance and improving students' learning outcomes [4]. The data used for learning analytics goes beyond just student grades and attendance. Different types of data can be used to improve educational learning outcomes. According to [5], eye-tracking data is useful for evaluating inter-subjectivity in face-to-face collaboration, log data can be used to predict student performance and contrition in team work for project-based learning, and automated dialog data can be used to predict the coming of newcomers in online learning. Also, other forms of data such as audio, student logs, and video capturing can be utilized in building Multimodal Learning Analytics (MMLA), which is useful for automating complex assessments and providing real-time valuable feedback, especially in project-based learning. This is applicable in text mining, students' gesture tracking, and programming analysis [6].

Learning analytics has had a significant impact in the ability to analyze students' learning outcomes by providing valuable insights into students' behavior and learning processes, but it has not yet been fully utilized beyond small-scale studies to cover larger and more complex education-related issues [7].

Within that context, the development of this tool, which aims to analyze students' programming keystroke dynamics, aligns well with the overall goal of learning analytics by aiming to identify students who might be struggling with their programming assignments and enabling instructors to provide early and proactive interventions to avoid dropouts and failures.

2.2 Keystroke Dynamics In Education

Keystroke dynamics, also referred to as keystroke biometrics, typing dynamics, or typing biometrics, refers to the collection of biometric information generated by key-press-related events that occur when a user types on a keyboard [8]. Thus, keystroke dynamics provides granular data that gives significant insights into students' keystroke actions – insertions, deletions, pauses, and so on. Keystroke patterns can also be obtained through typing games, and can be used to demonstrate the potential of identifying students' programming aptitude, especially in the early stages of taking programming courses [9].

Another application of keystroke dynamics is the identification of students who might be struggling in programming courses. Typing speed, typing rhythms, pauses, insertions, and deletions are features that can be used as powerful indicators of students' programming proficiency and performance in coding [10]. In a similar fashion, Shrestha R. [11] explored the use of keystroke data from students to detect struggling students by analyzing the students' thought processes, typing, and programming patterns. The analysis of their pausing behavior, pause-frequencies of different lengths and the last keystroke action before pausing, correlated with exam scores and provided insight into identifying struggling students [11].

Keystroke dynamics also provides effective ways of predicting student performance in courses that involve programming. In educational data mining, predicting students' academic outcomes — such as exam scores, mid-term exam, and final grades - is made possible by identifying struggling students for effective intervention early in the course using keystroke data that can be collected without having to place unexpected burden on the instructor. This is done by analyzing keystroke data comprising of time-on-task, typing speed, and pauses [12]. The correlation between students' pausing behavior and learning outcomes obtained through

keystroke dynamics also provides an understanding of students' coding strategies [13]. While keystroke data can be modeled to predict students' success, the models may not generalize well across different contexts and it could be misleading to rely solely on a single semester's data [14].

Keystroke dynamics goes beyond identifying struggling students and performance prediction. It also provides insight into students' emotional states while working on programming tasks. Emotions, such as being frustrated or stressed, can have significant impacts on students' learning outcomes. Cowley et al. [15] used a combination of keystroke data with surveys collected on students' perceptions of difficulty to identify when programmers are in a state of "flow" — defined as the mental state occurring when a person is so concentrated on an activity that they lose track of time and awareness of the self, which can occur when the difficulty of a task matches or slightly exceeds the individual skills [16]. Keystroke data can provide information that identifies typing patterns that are induced by stress, which can be used to detect users' emotional state [17]. Specifically, emotional states such as confidence, hesitance, nervousness, relaxation, sadness, and tiredness have been modeled using keystroke dynamics with the potential of identifying anger and excitement as well [18]. In addition, keystroke data has been found to be useful in detecting engagement and boredom. Research conducted by Bixler and D'Mello [19], which used keystroke data obtained from students typing three different essays on topics ranging across academic, socially charged issues and personal emotional experiences, found that keystroke latency, which is defined as the time interval between key presses, can be used to model students' emotional states.

2.3 Existing Approaches To Identifying Struggling Students

In the educational sectors, research to understand and identify students that are struggling has gained a lot of attention. Ranging from studying students' code compilation behaviors using real-time feedback systems to visualizations and building predictive models, several approaches have been used to provide early intervention to students that are struggling, especially in programming courses. A study analyzing students' code compilation behavior – precisely the edit-compile cycle – using the BlueJ programming environment reveals that struggling students are often in iterative cycles of editing and compiling their code [20]. A similar study has shown that the frequency of syntax errors detected in students' programs can provide the teacher with valuable insights toward identifying the areas where the students need support [21].

Beyond examining students' code compilation behavior, keystroke data have been shown to be useful for identifying struggling students. Gao et al. [22] employed Differential Sequence Mining (CDSM), a data-driven feature engineering algorithm, to identify sequences of events (i.e. patterns) in students' log data that differentiate two groups of students: high-performing (HP) and low-performing (LP) students. Specifically, the method identifies two types of patterns: patterns that occur for more students in one group than another (for example 30% of LP students vs. 60% of HP students), and secondly, patterns that occur for students in both groups, but appear more times for students in one group than another (for example, an average 5 times per assignment for HP students vs. 1 time per assignment for LP students). These patterns were used as features in a predictive model for students' performance early in

the semester using their keystroke data, which allows early identification of students at risk of dropping out or failing.

In courses where a Version Control System (VCS) such Git is used, students' commit histories have been used to track the progress of the students. In [23], Karakaš and Helic revealed that keeping track of students' commit histories proves a great way of early identification of students in struggles. They pointed out a positive correlation between students' frequency of commits and student performance while fewer commits indicate delay in starting the assignment and ultimately struggling to finish on time. The researchers concluded by opining that a combination of local testing and automatic commits of students assignment ensured that a continuous tracking of the students' progress is achieved.

Visualization tools are incredibly useful for monitoring student progress. McBroom et al. [24] proposed a novel visualization tool, Progress Networks, that summarizes the progression of a student population through a learning task. In particular, for each student they record a trace of their solution states while working on the task, then map each of the states to a level of progress, resulting in a trace of progress levels. Finally, they summarize these progress traces in a network, where progress levels are nodes and edges indicate how many students moved from one level to another. In addition, in [25], CodeProcess Charts was introduced using keystroke data generated from students to enable instructors to monitor students' coding processes and provide an assessment of their problem-solving approaches, as well as detect potential plagiarism.

Early detection of students at risk of dropping out has been revealed to identify struggling students. Pereira et al. [26] used data from the online CodeBench judging system, consisting of 9 introductory programming courses, to build a decision tree machine learning model that could predict early dropout (within two weeks of the start of classes), which helped to identify struggling students. Cabo [27] used association analysis and noted that early success in solving simple programming problems could predict students' ability to solve more complex problems that include loops and conditionals.

3 METHODOLOGY

3.1 Data Collection And Description

The data was collected from Codio [28], an online learning platform that is specifically designed for programming and computer science courses. Codio provides an Integrated Development Environment (IDE), where all actions of the students are logged while working on coding problems. The platform provides a rich source of fine-grained data about the students' coding habits, strategies, behaviors, and struggles for the entire duration of writing the programs. The data covers three semesters and having 13 programming assignments in CS1 courses (Introduction to Computing - Python and Java based)

The dataset consists of the following attributes:

- Date: This column contains the exact timestamp in the ISO 8601 format (e.g. 2022-09-05T00:13:53.698Z) and records when an event occurred. It tracks every student action

including character insertions, character deletions, and when code version changes. It also tracks when the student imports starter code or when starter code is provided for the students to complete.

- Position: This column shows th e cursor position where character insertion or deletion occurs.
- Insert: This column defines the character or lines inserted at the specified position, which gives a reflection of the contents that the students are adding to their program.
- Delete: This column defines the character or lines deleted by the student at the specified position. The delete attribute gives us an insight into students' frequency of revision or code removal, which plays a key role in identifying struggles or confusion.
- Version: This column is an integer value that represents the version of the code at that timestamp. The version increases with changes made by the students to their code, which allows for tracking of their code progression over time.
- User: This column is simply an identifier for each student that works on the program. Each file is unique to each student. There is an "internalReload" user that represents an automatically generated action provided by Codio. It is basically starter code for the student, and it does not reflect any action by the student.

3.2 Data Processing

Effective data processing is essential for the usability of the application. Data processing involves two steps: file extraction and directory parsing implemented using Python programming language using Visual Studio Code IDE. Initially, instructors upload a ZIP file containing the students' keystroke logs, which is then extracted and organized in a temporary directory to prevent conflict with existing files. The application validates the data extracted to ensure that the supported file formats are included. The directory parsing involves identification of the students' assignment folders and anonymizing the student names. The application handles errors by flagging empty or unsupported file types, which ensures that the data remains intact and usable for further analysis.

3.3 System Development And Design

The development of the analytical tool combines several components – reconstruction of code progression, error detection, visualization, and generation of code execution logs – into a cohesive system. The components work together to transform the raw keystroke data into interactive insights that help instructors analyze students' coding behaviors and challenges. The front-end is built with HTML and styled with CSS and Bootstrap to ensure a responsive design. The back-end uses Flask as the primary framework for handling HTTP requests and responses, while Flask-SocketIO enables the bi-directional communication.

Reconstruction Of Code Progression
Code reconstruction is the core functionality that transforms the students' raw keystroke data into code snippets at each recorded timestamp. As a student inserts and deletes

characters, each action is recorded with an associated timestamp, position index, and the inserted or deleted text. These events are chronologically applied in the order executed by the student to preserve the natural flow of the code progression and stored in an empty code buffer. The reconstruction algorithm iterates through each event, modifies the code representation according to the inserted or deleted text, and saves the resulting state at each timestamp.

This approach yields a complete timeline of the code evolution, thereby enabling the instructors to examine the sequential and incremental development of a student's solution and to pinpoint the exact moments when the student encountered challenges or appeared to be struggling.

The constructed code snippets are displayed to include syntax highlighting. This provides a visually accessible representation of the code as well as allowing for clarity and conciseness to the codes [29]. Once the code snippet is reconstructed, the application associates each state with its timestamp. The timestamps-to-code snippet matching is stored in a list of timestamp and code-snippet tuples. A slider/progress bar is constructed within the interface to allow the instructor to move seamlessly through the code evolution, thereby offering the flexibility of jumping to any point in time, reveal changes line by line, and observe the students' approaches and how it shifts throughout the entire problem solving process as shown in Figures 1 and 3.

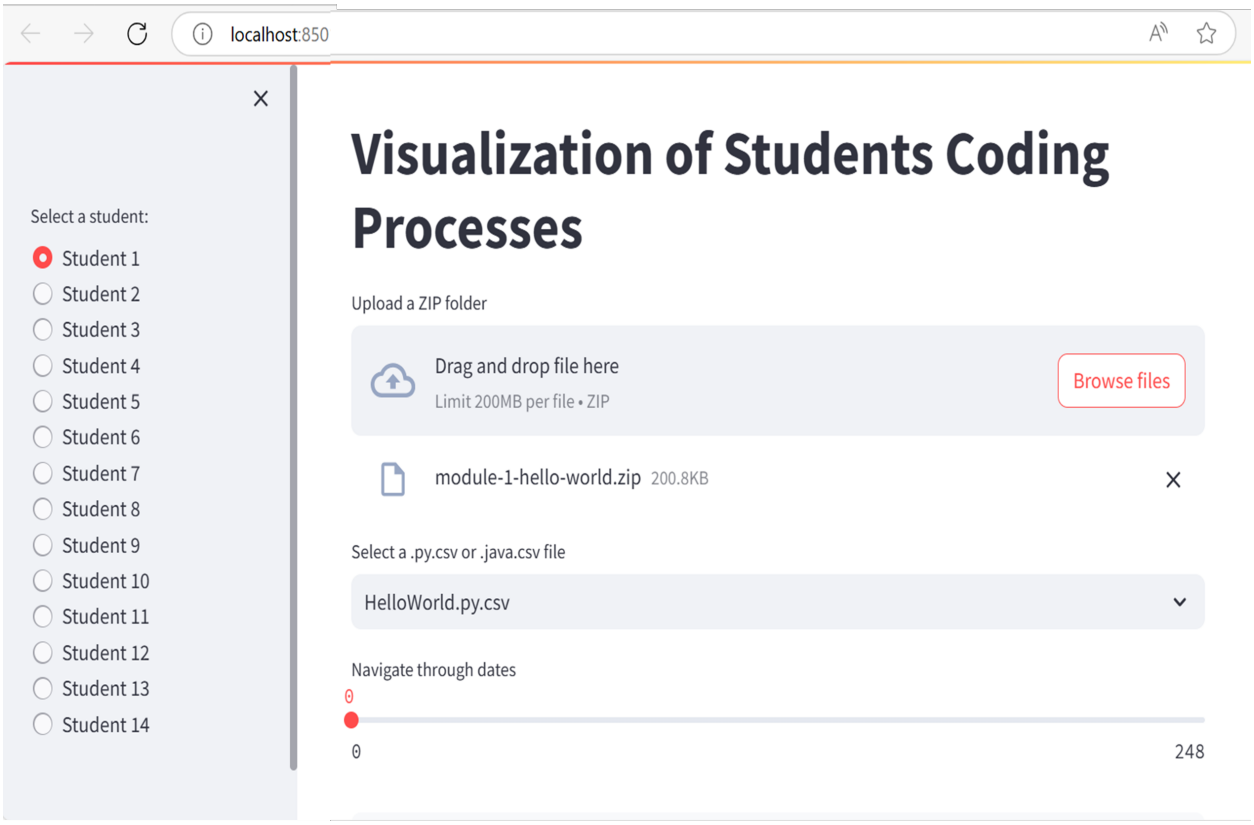

Figure 1: **Interface Showing File Upload and Navigation Timestamp Slider**

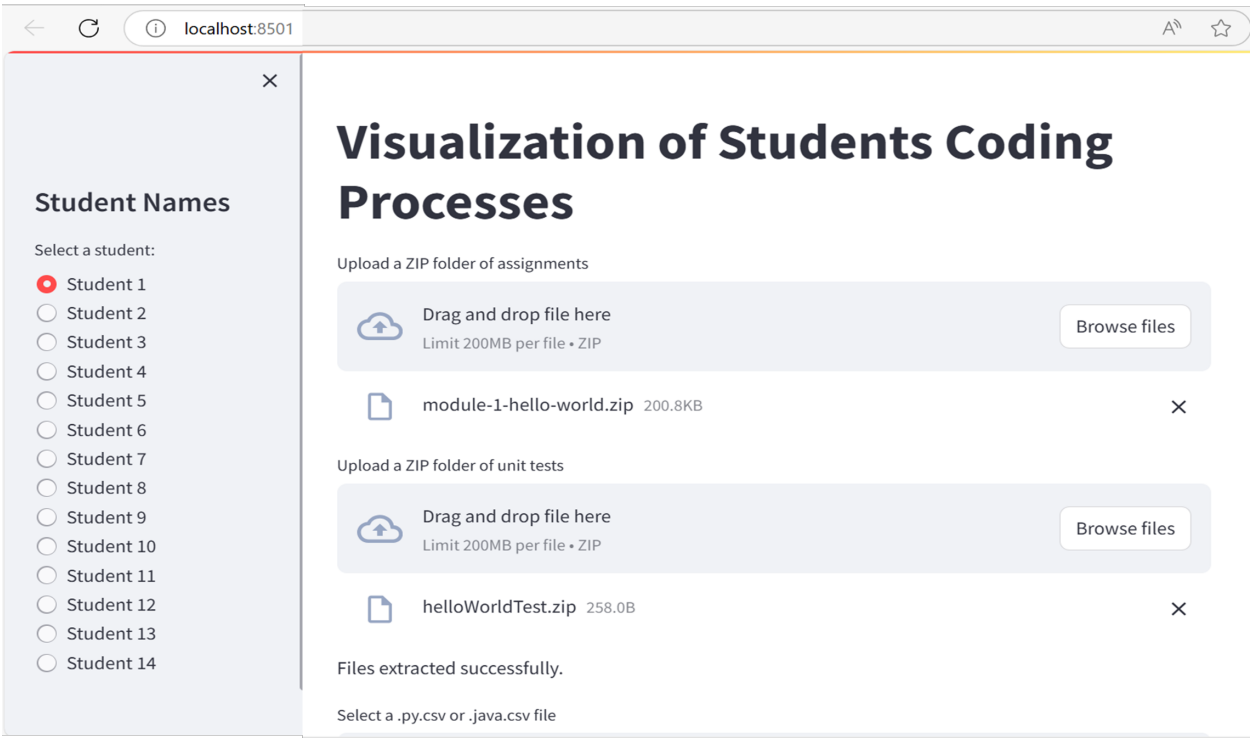

Figure 2: **Interface showing unit test upload functionality**

Interfacing With External Code Editor

The application integrates error detection by interfacing with an external code editor execution environment shown in Figure 4. The development of this interactive code editor involves combining Flask and Flask-SocketIO to create a dynamic, user-friendly platform for writing and executing code in real time. Python's Flask framework is a useful tool for developing dynamic web application [30].

The user interface as shown in Figure 4 includes a dual-pane layout: one for the code editor and another for displaying execution results. Additional fields are provided for entering command-line arguments and simulated user-input. A "Run Code" button triggers the execution of the code.

Unit Test Integration And Code Execution Logs

In addition to merely detecting compilation or syntax errors by interfacing with the external code editor, the application extends its functionality to provide a comprehensive and interactive review environment. This includes the integration of execution logs associated with a series of unit tests, which helps to ensure that the students' code performs as expected across various conditions. These unit tests can be simple outputs or a combination of inputs and their expected outputs, depending on the requirements of the assignment. These unit tests are loaded into the system (Figure 2) and are tailored towards verifying that the logic of the students' code behaves as expected. If input files are present, the application passes them as command-line arguments and the outputs are captured and compared against the expected results, revealing any discrepancies.

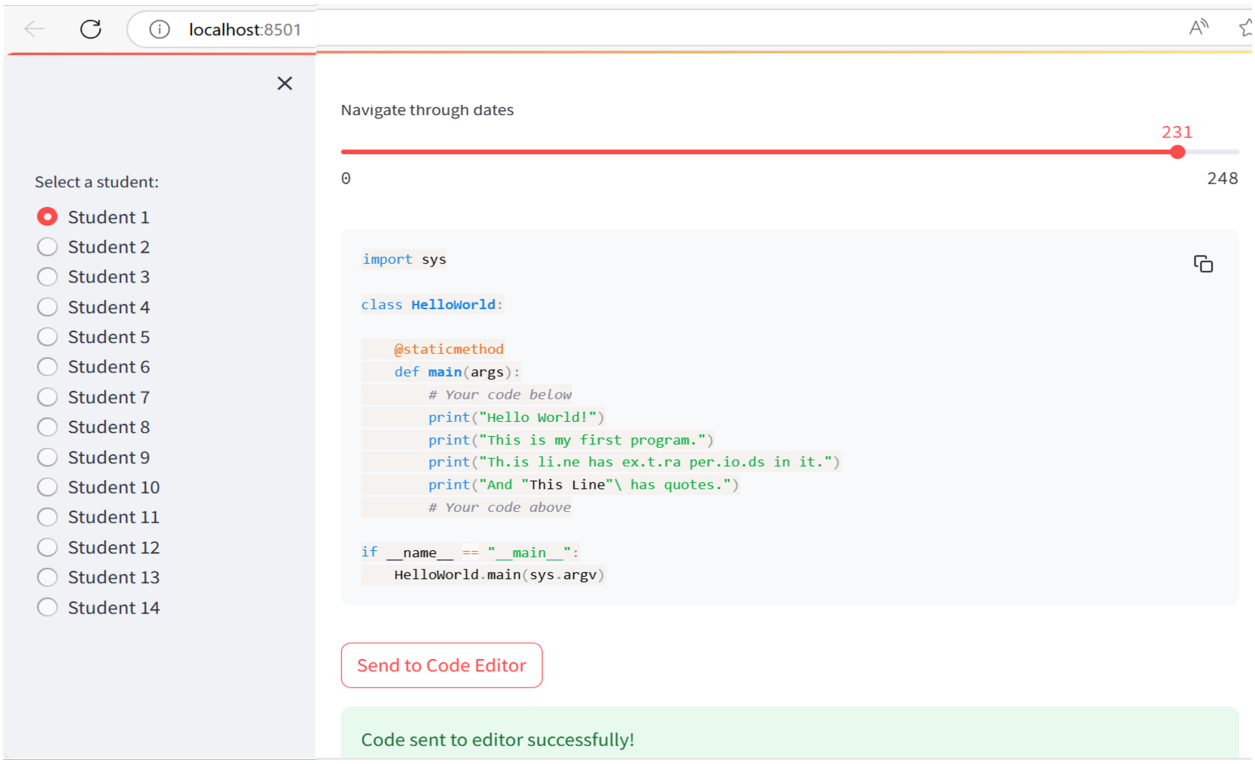

Figure 3: **Interface showing slider movement and corresponding code snippets**

Also, as part of the process, the system continuously tracks the pass/fail status of each test case and generates detailed execution logs that provide useful feedback as shown in Figure 5. The error details indicate different types of errors - runtime errors, syntax errors or logical mismatches that occur during execution.

Error Correction Times

The application includes a feature to track error correction times, providing a detailed timeline of when errors occurred and when the errors were resolved. Errors in this context implies the mistakes the students make while writing their programs. This could by syntax, logical, runtime or compilation errors (in case of Java codes). This feature records the timestamp of the error, the resolution timestamp, the details of the error and the time taken to correct it. For example, as shown in Figure 6, syntax errors such as "unterminated string literal" were logged with their timestamps, and the application automatically computed the time it took for the student to resolve the issue. This feature was implemented by capturing error events during code execution, which were then linked to students' subsequent keystrokes to determine when the errors were corrected.

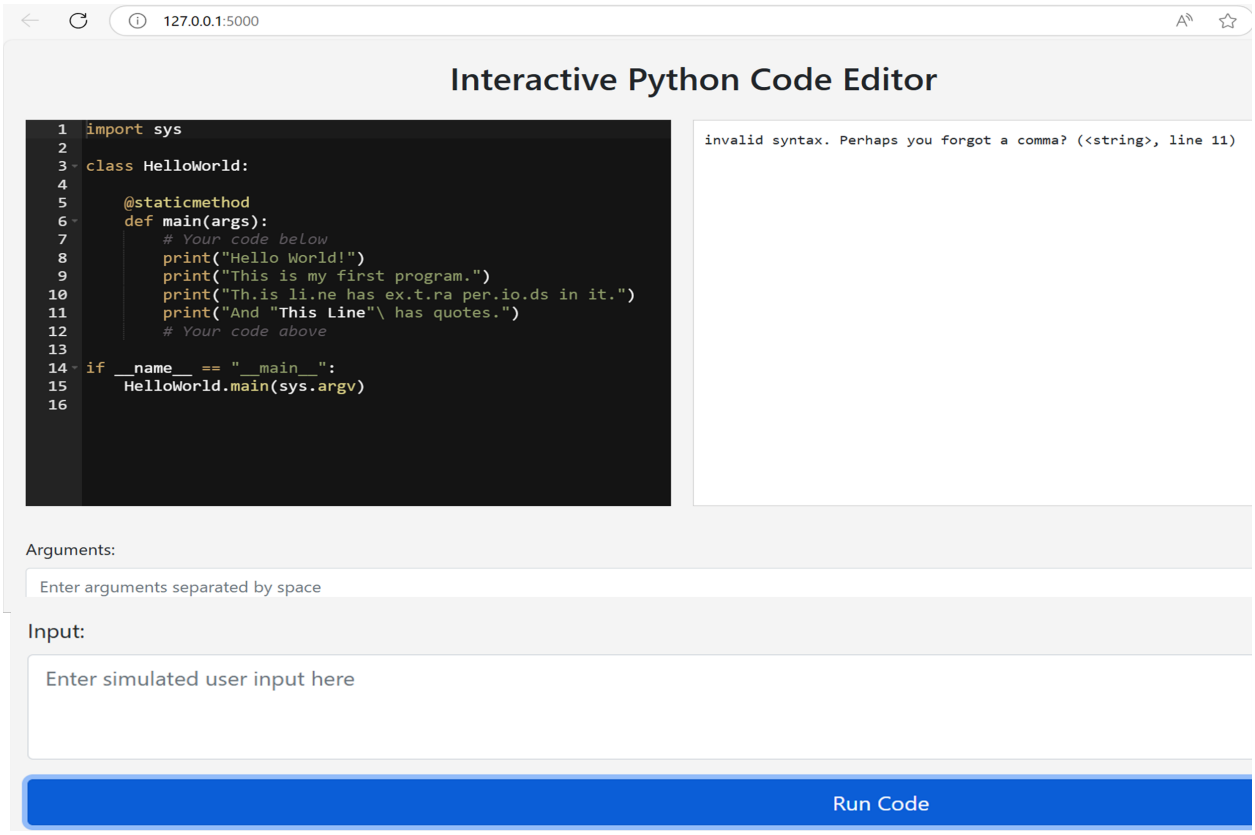

Figure 4: **Interaction with Code Editor on Clicking "Send to Code Editor" Button**

	Interval	Code Snippets	Pass Count	Fail Count	Execution Status	Error Details
0	2022-08-27 05:00:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	0	1	Error	File "C:\Users
1	2022-08-27 05:05:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	0	1	Successful Execution	
2	2022-08-27 05:10:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	1	0	Successful Execution	
3	2022-08-27 05:15:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	0	1	Error	File "C:\Users
4	2022-08-27 05:20:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	0	1	Successful Execution	
5	2022-08-27 05:25:00	import sys class HelloWorld: @staticmethod def main(args): # Your code be	1	0	Successful Execution	

	Error Timestamp	Resolution Timestan	Error Details	Time to Fix
0	2022-08-27 05:00:00	2022-08-27 05:05:00	File "C:\Users\frida\AppData\Local\Temp\tmp2fbz2qbk\student_code.py", line 12	300
1	2022-08-27 05:15:00	2022-08-27 05:20:00	File "C:\Users\frida\AppData\Local\Temp\tmpn2qm9rlq\student_code.py", line 11	300

Figure 5: **Interface Showing Code Execution Logs and Pass/Fail Counts**

4 Discussion Of Results

4.1 RQ1: Understanding Students' Programming Behaviors And Identifying Struggling Students

The analysis of students' keystroke data provided a comprehensive understanding of students' programming behaviors by reconstruction of their coding processes. The tool provided a clear progression of how the students' code evolved over time. Instructors could observe problem solving approaches that led to final implemented solutions. Struggling students demonstrated erratic patterns of coding with frequent rewrites and a seemingly uncoordinated approach to writing their codes.

The analysis revealed that extended pauses over time as well as frequent backtracking were indicative of confusion or difficulty with the assignment. Students who struggled displayed longer hesitations and tended to delete large portions of code before rewriting. These behaviors, if detected early, can enhance timely intervention.

The timeline slider enabled the instructors to identify the students who consistently approached the programming problems in a well-coordinated manner versus those who relied on a trial-and-error approach. This helped the instructor make clear distinctions between high-performing and struggling students without having to wait for final grades, which would make timely intervention practically impossible.

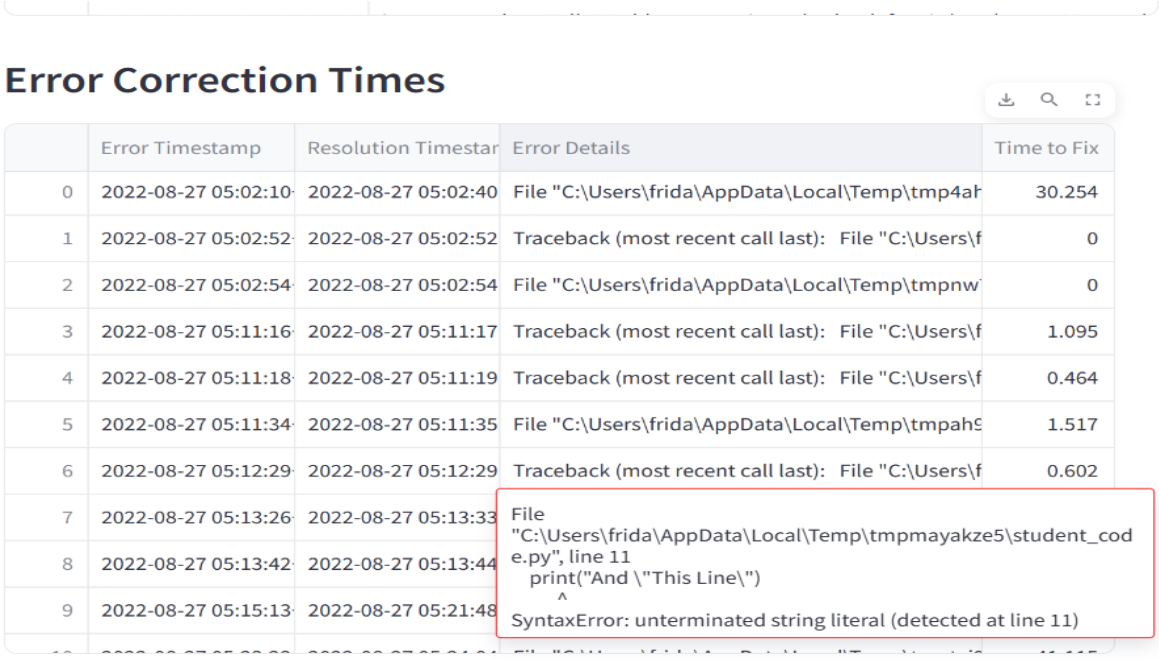

Figure 6: **Error Correction Times**

4.2 RQ2: Enhancing Teaching And Learning Through Code Execution Logs And Error Reports

The integration of automatic code execution logs and unit tests provided critical insights. The tool provided error tracking by capturing syntactical, logical, and runtime exceptions as they occurred at intervals of five minutes. Based on this, instructors could pinpoint common stumbling blocks and provide early intervention. In addition, keystroke execution logs revealed trends that informed instructional and pedagogical strategies. Instructors could identify recurring issues such as common syntax errors, or a more widespread misunderstanding of specific areas of the course materials since the analysis is done on a module-by-module basis. These provide a basis for instructors to refine course content and teaching strategies to better align with the needs of students. For example, if a significant proportion of students failed the same test case, it suggests that the underlying concept required further explanation or a change of the teaching method.

The tool also tracks the time that students spent correcting their errors. This provides insights into how quickly students observe and resolve coding issues as they work on programming tasks. Students with extended error resolution times are perceived to display struggling behaviors. By tracking the duration and frequency of error corrections, instructors can gain insight into students' debugging strategies.

Furthermore, by integrating unit tests with the keystroke analysis, the tool enables the instructors to dynamically assess code correctness. The pass/fail rates of the unit tests are clear measures of students' progress.

5 Ethical Considerations

Given the focus of this research on student data collection and analysis, the study adheres to established ethical guidelines in order to protect the students' privacy and maintain data security. This research has been approved by our University's Institutional Review Board (IRB), ensuring that all data collection and analysis comply with ethical standards for human-subject research. The tool operates locally and no data processing or analysis is transmitted to external servers or third-party platforms. This is to ensure data security and prevent unauthorized access to student keystroke logs. To further protect the students' privacy, all identifying information is fully anonymized before processing and analysis. The tool replaces the students' user IDs with generic identifiers (student 1, student 2, etc.). However, this anonymization is for research purpose only. The instructors utilizing this tool will have access to student information to be able to provide early intervention where necessary.

6 Scalability And Generalizability

This tool is currently designed to analyze keystroke data from the Codio learning platform. The underlying methodology and data processing can be extended to other Integrated Development Environments (IDEs). A key step in enhancing the tool's scalability is the current development of a plugin for Visual Studio Code (VS Code) which will allow keystroke tracking of student data similar to that of Codio. The plugin will capture similar keystroke

logs, including insertions, deletions and timestamps. This integration will allow use of this tool beyond Codio.

The tool has the ability to scale effectively in large classroom settings. Unlike traditional programming courses with large classes, where individual attention is challenging, the efficient data processing framework ensures that even with large class size, it will remain effective.

While the tool has been primarily tested on Python and Java, it is designed to make it adaptable to any programming language where keystroke logs can be obtained. Because the tool operates based on universal keystroke actions which are not language-dependent, the tool can be utilized to visualize code progression and generate automatic error reports across code snippets at specific timestamps.

7 Comparison Of The Proposed Keystroke Analytics Tool With Existing Learning Analytics Tools

Existing research studying programming behaviors has focused primarily on students' compilation behaviors rather than the entire programming process. One notable study is the work by Jadud [20], which introduced a compilation behavior analysis tool to track how novice programmers interact with the compiler in the BlueJ Integrated Development Environment (IDE). This tool collects data only when students compile their code, and provides insights into students' edit-compile cycles, focusing on the frequency of compilation attempts, time spent between compilation errors and most common compilation error occurrences. The proposed keystroke analytics tool differs by capturing every keystroke event rather than relying on compilation snapshots. It continuously collects records on insertions, deletions, monitors pauses and error corrections, thereby showing how students progressively develop their code before attempting to compile it. This makes it possible to detect when a student is struggling, even before attempting to run their code.

One of the key contributions of Jadud's study was the introduction of the Error Quotient(EQ) – a metric for quantifying how much a student struggles based on the frequency of syntax errors across multiple compilations [20]. However, students who hesitate, or delete and rewrite their code multiple times without compiling, may not be flagged as struggling even though they are experiencing difficulties. The proposed keystroke tool improves on the Error Quotient approach by providing more struggle indicators such as frequent deletions, erratic patterns of code progression, and tracking the time students spend resolving specific errors.

The proposed keystroke analytics tool also builds on the capabilities of CodeBench - a learning analytics system designed by Pereira et al. (2020) [31] to analyze students' interaction with an Online Judge system. CodeBench focuses on an automated assessment of students' programming assignments by collecting fine-grained data on keystrokes, number and correctness of submissions and time spent in the IDE. While CodeBench provides valuable insights into students' submission behaviors and the effectiveness of their problem-solving approaches, it's focus on submission-based evaluation contrasts it from the keystroke analytic tool. The proposed tool goes beyond submission-based analysis to reconstruct the entire coding process, thereby providing deeper insights into students' coding behaviors, point(s) of struggle and debugging strategies.

Another fundamental difference between the CodeBench and the proposed tools is in error analysis and debugging insights. CodeBench primarily evaluates final code correctness through automated test cases and employs machine learning and clustering techniques to classify students into performance categories. This does not track the intermediate steps students take to debug their programs, nor link error messages to specific keystroke events. The proposed tool addresses this gap by tracking error correction timelines, which allows an in-depth analysis of how students attempt to fix program errors over time.

The proposed keystroke analytics tool shares similarities with **CodeProcess Charts** [25] in that both tools aim to provide insights into students' programming behaviors by analyzing their code processes rather than just final submissions. However, while CodeProcess focuses primarily on visualizing the evolution over time, the proposed tool extends beyond visualization to provide keystroke-based struggle detection, error analysis and debugging insights.

8 Limitations

Although the tool provides valuable insights into students' coding process, and infers struggles based on patterns like frequent deletions, pauses or repeated errors, it does not determine whether a student is struggling due to some external distractions. Supplementing the keystroke analysis with other data such as self-reporting surveys from the students will ensure a more comprehensive assessment of struggle.

The tool is currently being implemented using completed courses. Additional data is being collected and will be used as a baseline performance metric to enhance the accuracy of the tool in detecting struggling students during the course itself.

9 Recommendations

To maximize the potential of keystroke data in computer science education, institutions should actively embrace the use of data-driven strategies. This is because integration of tools that utilize keystroke data and error reports allows educators to adopt a more proactive approach in identifying struggling students and providing early interventions, thereby preventing academic setbacks and improving student retention rates.

Given that the success of the tool relies on the ability of educators to interpret and review the effectively, it is important to provide instructors with training programs that would equip them with the skills to analyze keystroke patterns, interpret error logs, and identify actionable insights.

Furthermore, institutions should should explore ways to integrate these tools with existing educational technologies, such as embedding keystroke analysis and error reporting features into commonly used learning management systems like Canvas.

10 Future Work

Future enhancements of the keystroke analytics tool will focus on addressing implementation challenges and expanding its functionality through predictive analytics, broader platform

integration and multi-modal learning analytics. A key area of development involves the incorporation of machine learning models to automatically identify patterns indicative of struggling students. Extensive feature engineering will be performed to build predictive models that anticipate when a student is likely to struggle. In line with that, implementation challenges such as avoiding false positives and false negatives in struggle detection as well as optimizing computational efficiency will be carefully addressed.
To enhance the tool's versatility and effectiveness, future research will explore its application in diverse programming environments and collaborative coding platforms. While the current tools has been tested primarily in introductory programming courses, expanding its usage to more complex programming tasks such as software engineering, data structures or artificial intelligence courses will help validate its generalizability.
Another future direction for future work involves integration with widely used Learning Management System(LMS) such as Canvas. Embedding keystroke analytics within these platforms will allow instructors to access real-time insights directly. However, scalability and data integration challenges must be taken into account.

11 Summary

This research introduces a keystroke analytics tool designed to enhance programming education by tracking and analyzing students' coding behaviors at a granular level. The tool leverages keystroke dynamics to reconstruct students' code evolution, providing instructors with insights into problem-solving approaches, debugging strategies and potential points of struggle. Unlike traditional assessment methods that rely on final code submissions or compilation logs, the tool continuously captures keystrokes, insertions, deletions, pauses and error correction times, allowing for detection of struggling students. The research is structured around two primary research questions: The first explores how keystroke data can be effectively used to understand students' programming behaviors and identify struggling students for early intervention. The second research question examines the role of code execution logs and error reports in improving programming instruction. The tool integrates automated unit tests and execution logs, tracking syntax, runtime and logical errors.
The research compares the tool to existing educational technologies such as the BlueJ and CodeBench. The study acknowledges the limitation of not accounting for possible external distractions that might make a student struggle with programming code. Self-reported data is being collected to serve as a baseline performance metric to refine the struggle detection mechanism.

References

[1] G. Siemens, "Learning analytics: The emergence of a discipline," *American Behavioral Scientist*, vol. 57, no. 10, pp. 1380–1400, 2013.

[2] L. Tateo, "The journey of learning," *Mind, Culture, and Activity*, vol. 26, no. 4, pp. 371–382, 2019.

[3] J. T. Avella, M. Kebritchi, S. G. Nunn, and T. Kanai, "Learning analytics methods, benefits, and challenges in higher education: A systematic literature review.," *Online Learning*, vol. 20, no. 2, pp. 13–29, 2016.

[4] R. K. Veluri, I. Patra, M. Naved, V. V. Prasad, M. M. Arcinas, S. M. Beram, and A. Raghuvanshi, "Learning analytics using deep learning techniques for efficiently managing educational institutes," *Materials Today: Proceedings*, vol. 51, pp. 2317–2320, 2022.

[5] N. Nistor and Á. Hernández-García, "What types of data are used in learning analytics? an overview of six cases," *Computers in Human Behavior*, vol. 89, pp. 335–338, 2018.

[6] P. Blikstein, "Multimodal learning analytics," in *Proceedings of the third international conference on learning analytics and knowledge*, pp. 102–106, 2013.

[7] S. Dawson, S. Joksimovic, O. Poquet, and G. Siemens, "Increasing the impact of learning analytics," in *Proceedings of the 9th international conference on learning analytics & knowledge*, pp. 446–455, 2019.

[8] R. Moskovitch, C. Feher, A. Messerman, N. Kirschnick, T. Mustafic, A. Camtepe, B. Lohlein, U. Heister, S. Moller, L. Rokach, *et al.*, "Identity theft, computers and behavioral biometrics," in *2009 IEEE International Conference on Intelligence and Security Informatics*, pp. 155–160, IEEE, 2009.

[9] T. Nakada and M. Miura, "Extracting typing game keystroke patterns as potential indicators of programming aptitude," *Frontiers in Computer Science*, vol. 6, p. 1412458, 2024.

[10] J. Leinonen *et al.*, "Keystroke data in programming courses," *Department of Computer Science, Series of Publications A*, 2019.

[11] R. Shrestha, "Programming process, patterns and behaviors: Insights from keystroke analysis of cs1 students," Master's thesis, Utah State University, 2022.

[12] J. Edwards, K. Hart, R. Shrestha, *et al.*, "Review of csedm data and introduction of two public cs1 keystroke datasets," *Journal of Educational Data Mining*, vol. 15, no. 1, pp. 1–31, 2023.

[13] R. Shrestha, J. Leinonen, A. Zavgorodniaia, A. Hellas, and J. Edwards, "Pausing while programming: insights from keystroke analysis," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 187–198, 2022.

[14] Z. Pullar-Strecker, F. D. Pereira, P. Denny, A. Luxton-Reilly, and J. Leinonen, "G is for generalisation: Predicting student success from keystrokes," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 1028–1034, 2023.

[15] B. U. Cowley, A. Hellas, P. Ihantola, J. Leinonen, and M. Spape, "Seeking flow from fine-grained log data," in *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Software Engineering Education and Training*, pp. 247–253, 2022.

[16] J. Nakamura, M. Csikszentmihalyi, *et al.*, "The concept of flow," *Handbook of positive psychology*, vol. 89, p. 105, 2002.

[17] A. Kołakowska, "A review of emotion recognition methods based on keystroke dynamics and mouse movements," in *2013 6th international conference on human system interactions (HSI)*, pp. 548–555, IEEE, 2013.

[18] C. Epp, M. Lippold, and R. L. Mandryk, "Identifying emotional states using keystroke dynamics," in *Proceedings of the sigchi conference on human factors in computing systems*, pp. 715–724, 2011.

[19] R. Bixler and S. D'Mello, "Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable traits," in *Proceedings of the 2013 international conference on Intelligent user interfaces*, pp. 225–234, 2013.

[20] M. C. Jadud, "Methods and tools for exploring novice compilation behaviour," in *Proceedings of the second international workshop on Computing education research*, pp. 73–84, 2006.

[21] A. Altadmri and N. C. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proceedings of the 46th ACM technical symposium on computer science education*, pp. 522–527, 2015.

[22] G. Gao, S. Marwan, and T. W. Price, "Early performance prediction using interpretable patterns in programming process data," in *Proceedings of the 52nd ACM technical symposium on computer science education*, pp. 342–348, 2021.

[23] A. Karakaš and D. Helic, "Combining local testing with automatic commits: Benefits for progress tracking and cs2 students' learning experience," in *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pp. 108–114, 2024.

[24] J. McBroom, B. Paassen, B. Jeffries, I. Koprinska, and K. Yacef, "Progress networks as a tool for analysing student programming difficulties," in *Proceedings of the 23rd Australasian Computing Education Conference*, pp. 158–167, 2021.

[25] R. Shrestha, J. Leinonen, A. Hellas, P. Ihantola, and J. Edwards, "Codeprocess charts: Visualizing the process of writing code," in *Proceedings of the 24th Australasian Computing Education Conference*, pp. 46–55, 2022.

[26] F. D. Pereira, E. Oliveira, A. Cristea, D. Fernandes, L. Silva, G. Aguiar, A. Alamri, and M. Alshehri, "Early dropout prediction for programming courses supported by online judges," in *Artificial Intelligence in Education: 20th International Conference, AIED 2019, Chicago, IL, USA, June 25-29, 2019, Proceedings, Part II 20*, pp. 67–72, Springer, 2019.

[27] C. Cabo, "Student progress in learning computer programming: Insights from association analysis," in *2019 IEEE Frontiers in Education Conference (FIE)*, pp. 1–8, IEEE, 2019.

[28] C. Inc., "Codio — the hands-on platform for computing & tech skills education," 2024.

[29] M. Patrignani, "Why should anyone use colours? or, syntax highlighting beyond code snippets," *arXiv preprint arXiv:2001.11334*, 2020.

[30] V. R. Vyshnavi and A. Malik, "Efficient way of web development using python and flask," *Int. J. Recent Res. Asp*, vol. 6, no. 2, pp. 16–19, 2019.

[31] F. D. Pereira, E. H. Oliveira, D. B. Oliveira, A. I. Cristea, L. S. Carvalho, S. C. Fonseca, A. Toda, and S. Isotani, "Using learning analytics in the amazonas: understanding students' behaviour in introductory programming," *British journal of educational technology*, vol. 51, no. 4, pp. 955–972, 2020.