AnonTool: An Interactive Rule-Based Tool for Detecting Struggling Programming Students

Friday Emmanuel James, Kansas State University

Friday James is a Teaching Assistant Professor at Kansas State University, from where he bagged a PhD in Computer Science in 2025. 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 Associate 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.

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.

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.

Michelle Friend

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.

Mrs. Safia Malallah, Kansas State University

Dr. Safia Malallah is a teaching assistant professor at Kansas State University, where she completed her Ph.D. in Computer Science and early childhood. Her research is dedicated to advancing computer science and data science education across the PreK-12 and undergraduate levels. Dr. Malallah is particularly passionate about designing innovative and accessible learning experiences that cultivate essential computational skills in students

TrackIt: An Interactive Rule-Based Tool for Detecting Struggling Programming Students

Abstract

Identifying students who struggle during programming tasks remains a persistent challenge in computer science education, often resulting in delayed interventions and untimely support. Keystroke data analysis has emerged as an approach for capturing students' coding behaviors and identifying those who might have difficulties. This is made possible due to the granular nature of the data. By examining the keystroke dynamics such as typing speed, frequency of deletions, pauses and code reconstruction, it becomes possible to make an inference of the level of confidence, frustration and hesitations students experience during programming tasks. Nevertheless, while keystroke data can reveal insightful patterns, the interpretation of the results is complex and requires complementary data sources to validate inferences about students' struggles or confidence. This study introduces *TrackIt*, a rule-based system designed to detect struggling programming students by analyzing keystroke data in conjunction with self-reported survey responses. By applying a series of predefined thresholds and scoring rules, *TrackIt* computes a struggling score that categorizes students into five struggle levels. The system was validated in an introductory Python course involving 40 students with data collected from 547 labs and 130 homework keystroke logs, alongside structured post-assignment surveys. Results revealed that *TrackIt* effectively identified key struggle behaviors such as long pauses, frequent deletions, low insert-delete ratios and copy-paste events. These indicators closely aligned with students' self-reports, particularly for challenging assignments.

1.0 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 [24]. The novice programmers, when first exposed to the world of coding, often face a series of challenges – understanding the logic of programming, problems with debugging

errors, problems with compiling errors and runtime errors, and others [16] – all of which can hinder their learning progress. These challenges are quite common, and without adequate support, students become discouraged thereby leading to low engagement, and, in extreme cases, withdrawal from the course [19]. Computer science educators often resort to assignments and examination scores to assess students' level of understanding [21], 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 [18]. This is particularly important for large classes, where individualized attention is practically impossible [25]. We hereby propose a *TrackIt*, a rule-based system for detecting student struggle in programming assignments. Our approach is based on the development of a comprehensive feature extraction framework that captures behavioral patterns from keystroke logs, thereby allowing instructors to analyze the patterns of coding activities with fine granularity. We build on these features to develop a rule-based scoring system and classification algorithm for quantifying struggle in an interpretable manner, and correlate the system-generated struggle scores with self-reported measures obtained through structured survey instruments - enabling assessment of the alignment between the observed behavior and students' subjective experiences. This study seeks to provide answers to the following research questions:

RQ1: How does self-reported student feedback compare to keystroke-based indicators in identifying struggling students?

RQ2: What are the major areas/concepts in fundamental or introductory level programming courses that are most challenging to students?

2.0 Existing Approaches to Struggle Detection in Programming

Identifying struggling students in programming courses has been approached through various methodologies, including surveys, behavioral analysis, machine learning, and rule-based systems. Survey-based approaches, such as those by Bennedsen and Caspersen [4], provide self-reported insights into student difficulties, including confidence levels, time management, and concept comprehension [3, 5, 9, 10, 12]. While surveys capture emotional and cognitive dimensions of struggle, they are limited by self-reporting biases and lack immediacy. Behavioral and error-based analyses leverage coding logs and debugging patterns to detect struggle. Studies

by Altadmri and Brown [1] and Lundberg [14] highlight the prevalence of syntax and semantic errors, while Geng et al. [8] categorize learners by error resolution habits. Keystroke-based studies, such as James et al. [11], use test-driven code reconstruction to assess error correction timing. Repetitive test failures and trial-and-error debugging have also been flagged as struggle indicators [28, 29], although compiler error limitations persist [15]. Machine learning models offer predictive power but often act as black boxes. Castro-Wunsch et al. [7] employed neural networks for early detection, while Lokhande [13] and Vives et al. [32] explored deep learning and LSTM models, reporting high accuracy yet lacking interpretability. Language modeling approaches have predicted help-seeking behavior using code structure [17], while Liao et al. [22] used clicker data and regression analysis for early classification. These data-driven methods, despite their predictive strength, face issues of interpretability, generalizability, and reliance on large historical datasets. Rule-based systems, in contrast, offer transparency. Ulla et al. [30] assessed cognitive competencies using Bloom's Taxonomy, while Shirafuje et al. classified programming errors by expertise level. Rahman et al. [23] developed a recommender system based on rule-mined submission profiles, and rule-based classifiers have shown competitive accuracy with traditional models [6]. These systems offer adaptable, interpretable frameworks that align well with educational goals, especially where data scarcity or contextual constraints hinder machine learning applications.

3.0 Method

3.1 Research Design

This study utilizes quantitative approaches that integrate analysis of keystroke data and survey responses. The goal of combining these two data sources is to provide a more holistic understanding of student struggles and to assess how well the system aligns with self-reported experiences. The study is therefore designed to analyze survey responses of students in order to measure self-reported struggle as well as develop a set of predefined rules to classify students into struggle levels by using a weighted scoring system.

3.2 Participants and Course/Assignment Context

The programming tasks analyzed in this study consisted of ten lab assignments and five homework assignments designed to progressively develop students' proficiency in python programming. Each lab assignments consisted of one or more exercises and focused on foundational concepts such as variables, input/output, conditionals, loops, functions, lists, and dictionaries, with increasing complexity across topics. In contrast, the homework assignments were application-driven, requiring students to integrate multiple programming constructs. These included implementing a compound interest calculator, simulating a cash register and a vending machine using finite state machines, designing a decision tree for classification, and building a basic encryption system using a shift cipher. Together, the labs emphasized conceptual learning and syntax practice, while the homeworks emphasized problem-solving and the application of integrated skills in realistic scenarios.

3.3 Baseline Survey Data of Self-reported Struggles

To establish a ground truth for struggle classification, a structured survey was administered immediately after students completed their programming assignments. The goal was to comprehensively capture students' subjective and objective experiences related to difficulty struggle, confidence, and time taken to complete the assignments. The students provided their anonymized identifiers and responded to questions about perceived difficulty (on a five-point Likert scale ranging from "Very Easy" to "Very Difficult"), struggle level (using a five-point Likert scale from "Not all all" to "Struggled a lot") confidence (on a five-point Likert scale from "Not confident at all" to "Very confident"), completion time, and specific areas they found challenging. The resulting data served as a benchmark against which the automated predictions of the TrackIt system could be evaluated.

3.3.1 Validation of the Survey Instrument

To ensure methodological rigor, the survey instrument used as a baseline measure of self-reported struggle went through a validation process. First, content validity was established through alignment with established constructs in computer science education research. The survey measured perceived difficulty, self-reported struggle, confidence level, completion time, and open-ended identification of challenging concepts. These constructs are widely used indicators in studies examining novice programming experiences, affective responses, and

perceived cognitive load. By grounding the instrument in established theoretical dimensions of programming difficulty and student struggle, the survey reflects domain-relevant constructs. Second, the survey items were reviewed by multiple instructors with extensive experience teaching CS0 and CS1 courses across several semesters. This expert review ensured clarity, appropriateness of Likert-scale anchors, and alignment with the instructional context of introductory Python programming. Minor refinements were made to improve wording precision and eliminate ambiguity prior to deployment.

Finally, the survey was administered immediately after assignment completion to minimize recall bias and capture students' experiences while still cognitively salient. Together, theoretical grounding, expert review, and careful administration support the validity of the survey instrument as an appropriate benchmark for evaluating *TrackIt*'s behavioral classifications.

3.4 Keystroke Data

Keystroke data was collected from Codio, the learning platform where students write their code. The data comprises a total of 547 student keystroke logs from Lab assignments and 130 from homework assignments. Each Lab assignment includes at least one programming exercise, and for every exercise attempted, the keystroke logs were collected. Each entry records the precise time an event occurred (Date), the cursor location of the action (Position), and the nature of the change — whether characters were added (Insert) or removed (Delete). The Version column tracks incremental updates to the code, while the User field identifies the student responsible for the action. Notably, the dataset includes system-generated events under the user label "internal#Reload," which represent preloaded or starter code provided by the instructor. This structured log enables fine-grained analysis of student coding behavior over time.

3.5 Data Preprocessing

Before applying the rule-based classification, the raw keystroke data was preprocessed as follows:

- Timestamp conversion, to ensure that all times are in a uniform format of Day, Hour, Minutes, Seconds.

- Anonymization of students' usernames by creating pseudonyms to replace the student user ids (except for the "internal#reload" events – which represents instructor-supplied starter codes).
- The time differences between the keystrokes were also computed, with the aim of helping to identify pauses while programming.

3.6 Features of TrackIt

TrackIt was developed with a range of practical features that make it a powerful tool for analyzing students' programming behavior and identifying occurrences of struggle. One of the key functionalities is the ability to upload and analyze programming assignments one lab or homework at a time, allowing instructors to isolate specific tasks and provide contextual evaluation of students' performance. Once a ZIP file – containing keystroke logs from each student for an assignment – is uploaded, it processes the raw keystroke logs and computes the time differences between each keystroke timestamp per student, in order to identity pauses while programming. Instructors can select individual students and inspect their keystrokes and behavioral features and the associated struggle scores and predicted struggle ratings. In addition, the system provides a summary view of each student's struggle score that offers instructors a searchable table that helps quickly identify students who may need support.

Another core feature of TrackIt is its ability to detect copy-paste behavior, thereby offering powerful insights into students' dependency on external resources. For example, in our analysis, the tool flagged a student who had directly pasted texts from Gemini – a popular AI assistant (Figure 1) and another student who pasted texts from Reddit – an online interactive platform (Figure 1). The keystroke tracked and revealed a sudden influx of a well-structured code block with no prior incremental typing or editing activity, which is a clear indicator of external sourcing. These behaviors were captured by TrackIt, and the pasted segment automatically displayed. This feature not only helps in identifying potential academic dishonesty but also serves as a marker of struggle as students who resort to such methods are often perceived to do so out of frustration, or lack of confidence. As such, TrackIt appropriately penalizes such situations by flagging them as struggling, providing instructors with early signs for timely intervention.

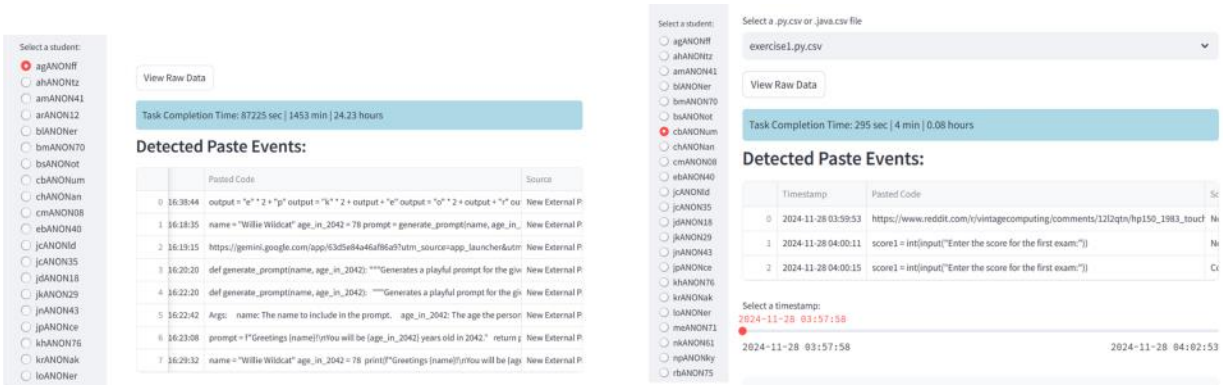

Figure 1: Copy-Paste Detection

3.7 Feature Extraction from Keystroke Data

To assess student struggles effectively, *TrackIt* extracts a range of keystroke features that provide insight into students' programming behaviors. These features are meant to capture cognitive load, problem-solving efficiency, hesitation patterns and overall engagement with the programming task.

3.7.1 Time-based Features.

- ***Total Duration:*** This represents the entire period a student spends writing their code, from the first recorded keystroke to the last. A short duration may indicate that a student completed the task quickly with confidence but could also mean that the student turned in the assignment out of frustration from struggling with it.
- ***Total Pauses:*** This represents the cumulative time when no keystrokes were recorded. That is, the student was either thinking, planning, debugging or disengaged from the assignment.
- ***Idle Time Ratio:*** This quantifies the proportion of the student's total coding session spent pausing rather than actively typing or making code modifications. It serves as an indicator of struggle as excessive idle time may suggest that the student is encountering difficulties in progressing through the coding task.

3.7.2 Pause Classification Features: To better understand student hesitation and engagement levels, *TrackIt* categorizes the pauses into different ranges based on their duration. Each pause type reflects a different level of problem-solving effort and struggle. This work draws from pause categorizations of [27], which were based on studies by [2, 20, 31] to create five key pause categories.

- ***Micro Pauses:*** Micro pauses represent brief interruptions in typing that might just be small hesitations. This could be the student thinking about syntax, recalling function names or planning their next step. The micro pauses are natural and are often expected in programming.
- ***Mild Pauses:*** Mild pauses occur when a student stops typing for longer than 15 seconds but less than 2 minutes (120 seconds). These often happen when students are consulting notes, reviewing instructions or debugging minor issues.
- ***Short Pauses:*** Short pauses refer to gaps between 2 and 3 minutes, often resulting from students strategizing or rethinking their approach to the problem.
- ***Mid Pauses:*** Mid pauses indicate longer hesitations lasting between 3 and 10 minutes. This often results from the student being deeply stuck, debugging a complex issue or searching for external resources.
- ***Long Pauses:*** Long pauses occur when students stop interacting with the code editor for over 10 minutes. This is an indication of total disengagement, suggesting that the student may have abandoned the assignment, become frustrated or lost confidence and motivation to continue with the programming task.

3.7.3 Typing and Editing Behavior Features

- ***Total Insertions:*** This indicates the number of characters added to the code during the session and includes both the new code and rewritten codes after deletions, not considering characters that come from instructor-provided starter codes.
- ***Total Deletions:*** Total deletions count the number of characters removed from the code.
- ***Insert-Delete Ratio:*** The Insert-Delete Ratio measures the proportion of the insertions to deletions in a programming assignment. It provides insight into how efficiently a student is typing versus correcting mistakes.
- **Correction Speed/Frequency:** This measures how frequently a student corrects errors, which could indicate debugging behavior or difficulty in making progress in writing the code.

3.8 Rule-Based Classification of Struggle

This study employs a rule-based classification system to quantify and categorize student struggle levels based on keystroke data. The classification system computes a struggle score by evaluating the features extracted from the keystroke data. The features used for the classification contribute a specific number of points to the struggle score based on predefined threshold values — with higher scores indicating more struggle. These threshold and conditions, as well as the scores, were determined from previous experiences of teaching CS0 and CS1 level programming students over multiple semesters. This struggle score is computed as a weighted sum of contributions from multiple keystroke features. The final score is then used to categorize students into five struggle levels. Table 1 shows the rules set to govern the assignment of struggle scores

Table 1: Struggle Classifications

Feature	Thresholds / Conditions	Struggle Score	Struggle Level
Idle Time Ratio	< 0.3	0	No Struggle
	0.3 – 0.5	+2	Slight Struggle
	0.5 – 0.7	+4	Some Struggle
	> 0.7	+6	Moderate/High Struggle
Pause Frequency	Short (2–3 mins): >5 / >10 times	+3/+6	Slight / Moderate
	Mid (3–10 mins): >5 times	+3	Moderate
	Long (>10 mins): >2 times	+6	High Struggle
Insert-Delete Ratio	> 3.0	0	Confident Typing
	1.5 – 3.0	+2	Slight Struggle
	1.0 – 1.5	+4	Some Struggle
	< 1.0	+6	Moderate/High Struggle
Deletion Ratio	< 0.3	0	Few Corrections
	0.3 – 0.7	+3	Slight Struggle
	> 0.7	+6	High Struggle
Correction Frequency	< 10 deletions/min	0	Confident Typing
	10 – 30 deletions/min	+2	Some Struggle
	> 30 deletions/min	+5	Moderate/High Struggle
Copy-Paste Behavior	One large paste (>50 chars)	+2	Slight Struggle
	One very large paste (>100 chars)	+4	Moderate Struggle
	Frequent large pastes (>3 times)	+6	High Struggle
Final Struggle Score Range	0–4	Not at all	
	5–8	Slight Struggle	
	9–12	Some Struggle	
	13–16	Moderate Struggle	
	>16	Struggled a lot	

4.0 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. However, this anonymization is for research purposes only. The instructors utilizing this tool will have access to student information to be able to provide early intervention where necessary.

5.0 Results

5.1 Survey Reports Summary Statistics

Table 2 shows the summary statistics of the perceived difficulty, struggle rating, and confidence levels from students, as well as their associated completion times across the lab and homework assignments. In terms of difficulty level, the mean difficulty for lab assignments is 3.34, whereas homework assignments have a slightly higher mean of 3.62. This suggests that students generally perceived homework as more difficult than lab work. The completion time presents a more pronounced difference – while the average time taken to complete the labs is 129.48 minutes, that of the homework assignment is 152.09 minutes. As expected, the median struggle rating for labs is 3 while it is 4 for the homework assignments – a majority of the students found the homework more difficult than lab assignments. Confidence levels remain consistent across both assignment types, with an average of 2.74, suggesting that the level of self-assurance students felt while completing the assignments did not change significantly. However, the standard deviation is slightly higher for homework tasks, indicating a greater spread in confidence levels among the students. Overall, the results show that students struggle more with homework assignments and generally find them more difficult and time-consuming than lab assignments.

Table 2. Summary Statistics for Lab and Homework Assignments

Statistic	Difficulty Level		Completion Time (Minutes)		Struggle Rating		Confidence Level	
	Lab	Homework	Lab	Homework	Lab	Homework	Lab	Homework
Count	292	129	279	129	292	129	292	129
Mean	3.34	3.62	129.48	152.09	3.35	3.60	2.74	2.74
Std Dev	0.98	0.97	110.95	121.69	1.14	1.17	1.10	1.20
Min	1	1	10	30	1	1	1	1
25%	3	3	60	70	3	3	2	2
50%	3	4	100	120	3	4	3	3
75%	4	4	180	180	4	5	3	4
Max	5	5	900	840	5	5	5	5

5.2 Correlations from Survey Reports

The correlation heatmaps (Figure 5) for both lab and homework assignments reveal both similarities and subtle differences in how difficulty, struggle, confidence and completion times interact across these two types of homework. One of the most consistent patterns between the two is the strong positive correlation between the difficulty level and struggle rating, which is 0.87 for both lab and homework. This indicates that regardless of the type of assignment, students tend to struggle more as the difficulty increases. However, differences emerge in the relationship between difficulty level and confidence. In the lab assignments, the negative correlation between difficulty level and confidence level is -0.75, whereas in the homework, it is slightly weaker at -0.65. This suggests that while confidence declines as difficulty increases, the impact is more pronounced in lab assignments. In a similar way, struggle rating and confidence level show a strong negative correlation in both cases, with -0.71 for labs and -0.7 for homework. This consistency implies that increased struggle leads to lower confidence regardless of whether the task is a lab or homework assignment.

Completion time exhibits notable differences between labs and homework. In lab assignments, difficulty level and completion times have a moderate positive correlation of 0.51, while in homework, this relationship is weaker at 0.42. A similar pattern occurs between struggle rating and completion time, which is 0.51 for labs but drops to 0.39 for homework. This indicates that

while harder and more challenging tasks take longer to complete in both cases, the effect is more pronounced in lab assignments. This could be because students tend to start homework assignments late and have to work within a short time frame to get them done. Confidence level and completion time also show differences between the two assignment categories. In lab assignments, the correlation is -0.36 suggesting that students with higher confidence tend to complete the tasks more quickly. However, in homework, this correlation weakens to -0.21, implying that confidence plays a smaller role in determining how long it takes to finish their homework compared to lab assignments. This may be as a result of the more flexibility in the pacing of the homework assignments, making the students feel more confident and comfortable completing them.

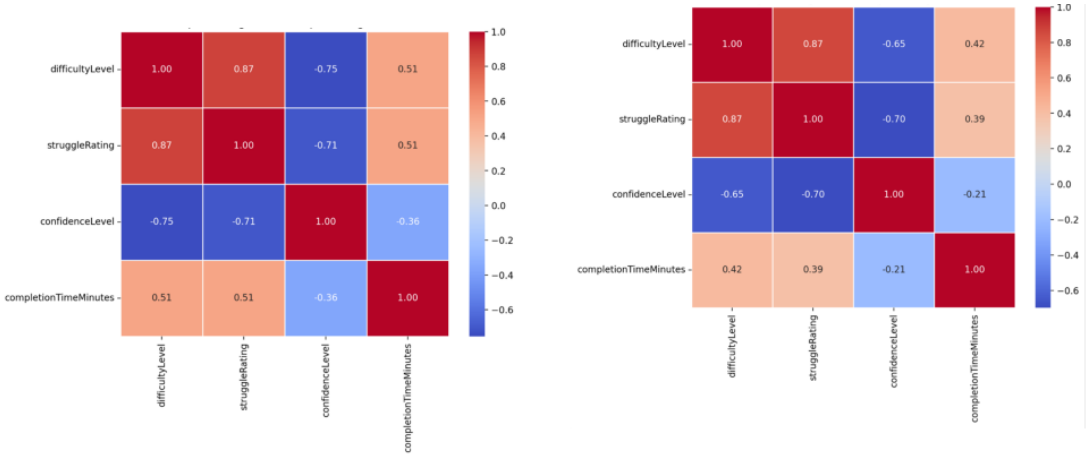

Figure 2: Correlation Coefficients

5.3 Identifying Most Challenging Assignments

A striking observation from the comparison of struggle ratings between *TrackIt's* automated analysis and student self-reported surveys is the strong alignment in identifying the most difficult assignments (Figure 3). For example, in the lab comparison, lab 5 stands out as the most difficult task in both the survey and *TrackIt* analysis. Similarly, the homework comparison shows that both the survey and *TrackIt* consistently rate homework 3 as the most challenging, thereby validating the accuracy of TrackIt's detection methods. Across all the assignments, *TrackIt* tends to slightly overestimate the level of students' struggle compared to self-reports,

which could be due to its sensitivity to behavioral indicators like frequent deletions, long pauses and copy-paste events.

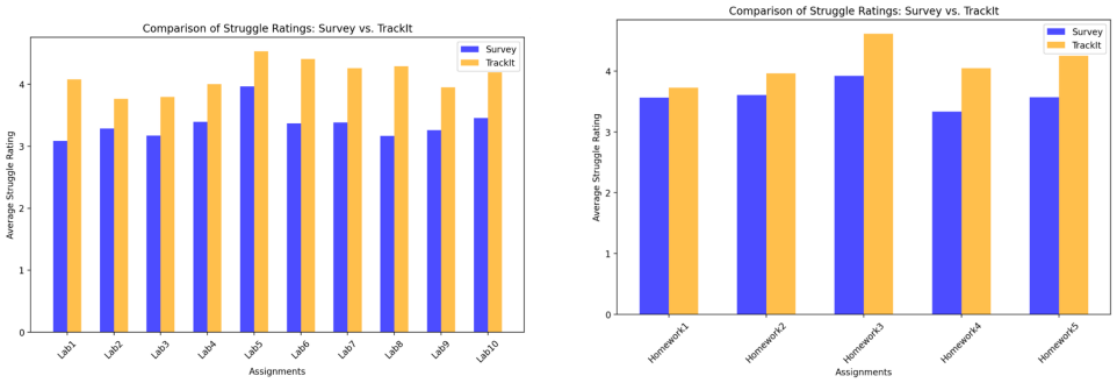

Figure 3: Identifying Most Challenging Labs and Homework Assignments

5.4 Comparison Between Self-Reported Struggle and Keystroke-Based Indicators (RQ1)

Our first research question investigates how self-reported student feedback compares to keystroke-based indicators in identifying struggling students. To address this, alignment was examined at both the assignment and behavioral-feature levels.

5.4.1 Assignment level: Tasks that received higher average self-reported struggle ratings were also associated with elevated automated struggle scores generated by *TrackIt*. Assignments identified by students as particularly challenging corresponded to those exhibiting increased idle time ratios, longer cumulative pause durations, higher deletion ratios, and lower insert-delete ratios. This convergence suggests that behavioral signals captured in keystroke logs reflect patterns consistent with students' subjective experiences.

5.4.2 Feature level: Specific keystroke indicators demonstrated meaningful alignment with perceived difficulty:

- Elevated idle time ratios were observed in assignments with higher reported struggle.
- Increased frequencies of mid and long pauses coincided with assignments rated as difficult.
- Lower insert-delete ratios reflected iterative debugging behavior associated with conceptual uncertainty and ultimately, confidence level.
- Higher deletion ratios and correction frequency aligned with reported frustration.

- Copy-paste events were more common in assignments where students reported greater difficulty, suggesting potential disengagement or reliance on external resources.

Together, these findings indicate that behavioral analytics derived from keystroke data capture observable manifestations of difficulty that align with students' self-reported experiences.

5.5 Identification of Struggling Areas/Concepts (RQ2)

Analysis of both the keystroke behaviors and the self-reported surveys revealed patterns about which programming concepts were most difficult for the students. Assignments were mapped to their primary conceptual focus, including variables, conditionals, loops, functions, data structures, state machines, classification logic, and encryption algorithms. From both the survey data and *TrackIt's* analysis, students' assignments involving nested structures, multi-conditionals statements and loops had higher levels of flagged struggle. From Figures 4, it can be observed that the students' level of confidence was relatively low in the assignments that covered these concepts, hence their high average struggle rating. Figure 3 shows that both Survey and TrackIt identified Lab 5 (covering nested conditionals) and Homework 3 (Building a Finite State Machine where loops are utlized) to be the most challenging concept areas.

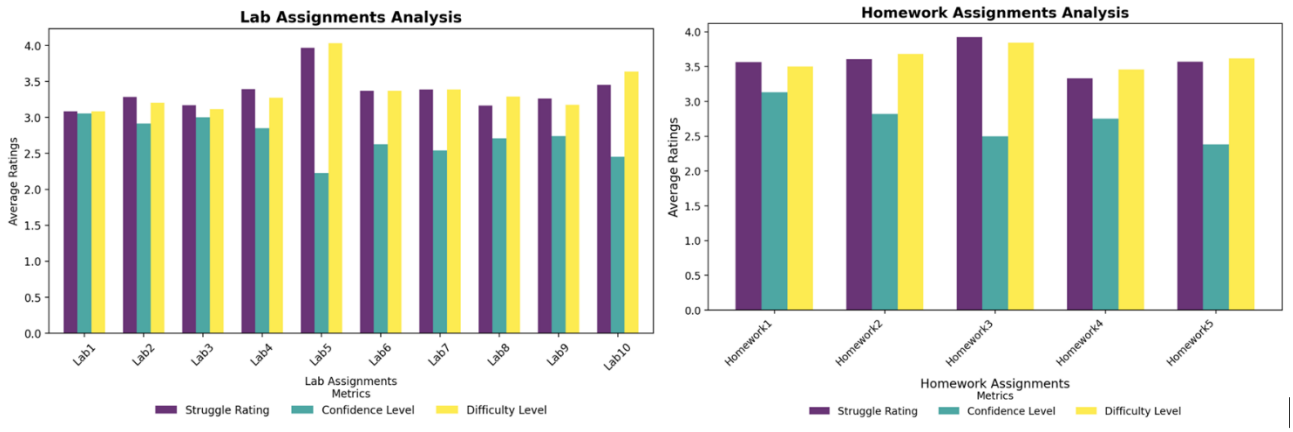

Figure 4: Assignment Ratings

T6.0 Discussion

This study highlights the promising capabilities of *TrackIt* as a behavioral analytics tool for detecting struggling students in programming assignments. By comparing *TrackIt's* outputs with students' self-reported struggle surveys, the system was able to identify the same assignments that students rated as the most challenging. This agreement suggests that TrackIt can meaningfully reflect the perceived difficulty of tasks. While student surveys provided insights into personal experiences such as confidence, task difficulty and completion times, offers perspective based on user interaction patterns. These two sources offer complementary scenarios in student learning challenges. A particularly insightful component of TrackIt was its ability to flag copy-paste behavior, which often occurs among high struggling students. Students who face prolonged difficulty tend to stop attempting their own solutions and rather copy code into the environment. Although it is not always a problem to paste codes while coding – *TrackIt* is sensitive in detecting code that was pasted from external sources and copied from already created block of codes by the student – repeated or untimely copy-paste behavior often signaled disengagement.

6.1 Pedagogical Implications

The findings from this study highlight the promising pedagogical value of TrackIt as an analytical tool for identifying struggling students in programming courses. Although the current implementation of TrackIt operates on a post-hoc basis – analyzes keystroke data after an assignment has been completed – it still provides instructors with powerful insights into the unseen dimensions of the learning process. Its ability to align with student-reported struggle through behavioral indicators validates its potential to enhance timely intervention. By this, instructors are not limited to interpreting students' learning through final grades, rather, they can have a process-oriented view of how students arrived at their solutions, where they paused and probably hesitated, how the patterns of their interactions in the coding environment changed as they progressed with the assignments.

In addition, TrackIt potentially enables proactive teaching, where instructors can analyze struggle between class sessions or before the next assignment in order to channel supports accordingly. For example, if several students displayed patterns of struggle in a

lab assignment involving loops or conditions, the next class could begin with a targeted review or an interactive activity that focuses on those concepts. This data-driven pedagogy offers a greater response to students' learning needs. Looking ahead, the pedagogical implications of TrackIt would be amplified through real-time integration. While its current form offers post-hoc assignment analysis, its foundational capabilities lay the groundwork for a future in which students could receive right-on-the-spot feedback, and instructors could be alerted as struggle unfolds.

REFERENCES

1. Amjad Altadmri and Neil CC Brown. 2015. 37 million compilations: Investigating novice programming mistakes in large-scale student data. In Proceedings of the 46th ACM technical symposium on computer science education. 522–527. https://doi.org/doi:10.1145/2676723.2677258
2. Rui A Alves, São Luís Castro, Liliana De Sousa, and Sven Strömqvist. 2007. Influence of typing skill on pause-execution cycles in written composition. Writing and cognition: Research and applications (2007). https://doi.org/10.1163/9781849508223_005
3. Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D Hundhausen, and Abigayle Peterson. 2021. In situ identification of student self-regulated learning struggles in programming assignments. In Proceedings of the 52nd ACM technical symposium on computer science education. 467–473. https://doi.org/10.1145/3408877.3432357
4. Jens Bennedsen and Michael E Caspersen. 2007. Failure rates in introductory programming. AcM SIGcSE Bulletin 39, 2 (2007), 32–36. https: //doi.org/doi:10.1145/1272848.1272879
5. Susan Bergin and Ronan Reilly. 2005. Programming: factors that influence success. In Proceedings of the 36th SIGCSE technical symposium on Computer science education. 411–415. https://doi.org/10.1145/1047344.1047480
6. Luca Cagliero, Lorenzo Canale, Laura Farinetti, Elena Baralis, and Enrico Venuto. 2021. Predicting student academic performance by means of associative classification. Applied Sciences 11, 4 (2021), 1420. https://doi.org/10.3390/app11041420
7. Karo Castro-Wunsch, Alireza Ahadi, and Andrew Petersen. 2017. Evaluating neural networks as a method for identifying students in need of assistance. In Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education. 111–116. https://doi.org/10.1145/3017680.3017792
8. Chuqin Geng, Wenwen Xu, Yingjie Xu, Brigitte Pientka, and Xujie Si. 2023. Identifying different student clusters in functional programming assignments: From quick learners to struggling students. In Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. 750–756.
9. Jamie Gorson, Nicholas LaGrassa, Cindy Hsinyu Hu, Elise Lee, Ava Marie Robinson, and Eleanor O'Rourke. 2021. An approach for detecting student perceptions of the

programming experience from interaction log data. In International Conference on Artificial Intelligence in Education. Springer, 150–164. https://doi.org/10.1007/978-3-030-78292-4_13

10. Noman Islam, G Shafi Sheikh, Ridah Fatima, and Farrukh Alvi. 2019. A study of difficulties of students in learning programming. Journal of Education & Social Sciences 7, 2 (2019), 38–46. https://doi.org/10.20547/jess0721907203
11. Friday E James, Russell Feldhausen, Nathan H Bean, Josh Weese, David S Allen, and Michelle Friend. 2025. From Typing to Insights: An Interactive Code Visualization for Enhanced Student Support Using Keystroke Data. In Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2. 1493–1494.
12. Paivi Kinnunen and Beth Simon. 2010. Experiencing programming assignments in CS1: the emotional toll. In Proceedings of the Sixth international workshop on Computing education research. 77–86. https://doi.org/10.1145/1839594.1839609
13. DB Lokhande, TA Chavan, and DP Patil. 2022. Deep neural network in prediction of student performance. education 18 (2022), 22.
14. Leo Lundberg. 2024. Classifying Struggling Students Through Error-Message Analysis in Programming Education: A Swedish Case Study.
15. Davin McCall and Michael Kölling. 2014. Meaningful categorisation of novice programmer errors. In 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE, 1–8.
16. Davin McCall and Michael Kölling. 2019. A new look at novice programmer errors. ACM Transactions on Computing Education (TOCE) 19, 4 (2019), 1–30. https://doi.org/10.1145/3335814
17. Marco Moresi, Marcos J Gomez, and Luciana Benotti. 2021. Predicting students' difficulties from a piece of code. IEEE Transactions on Learning Technologies 14, 3 (2021), 386–399. https://doi.org/10.1109/TLT.2021.3092998.
18. Jonathan P Munson and Joshua P Zitovsky. 2018. Models for early identification of struggling novice programmers. In Proceedings of the 49th ACM technical symposium on computer science education. 699–704. https://doi.org/10.1145/3159450.3159476
19. Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky–a qualitative analysis

of novices' strategies. ACM SIGCSE Bulletin 40, 1 (2008), 163–167. https://doi.org/10.1145/1352322.1352191

20. Thierry Olive, Rui Alexandre Alves, and São Luís Castro. 2009. Cognitive processes in writing during pause and execution periods. European Journal of Cognitive Psychology 21, 5 (2009), 758–785. https://doi.org/10.1080/09541440802079850
21. José Carlos Paiva, José Paulo Leal, and Álvaro Figueira. 2022. Automated assessment in computer science education: A state-of-the-art review. ACM Transactions on Computing Education (TOCE) 22, 3 (2022), 1–40. https://doi.org/10.1145/3513140
22. Anupama Prasanth and Haitham Alqahtani. 2023. Predictive Modeling of Student Behavior for Early Dropout Detection in Universities using Machine Learning Techniques. In 2023 IEEE 8th International Conference on Engineering Technologies and Applied Sciences (ICETAS). IEEE, 1–5. https://doi.org/10.1109/ICETAS59148.2023.10346531
23. Md Mostafizer Rahman, Yutaka Watanobe, Uday Kiran Rage, and Keita Nakamura. 2021. A novel rule-based online judge recommender system to promote computer programming education. In Advances and Trends in Artificial Intelligence. From Theory to Practice: 34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2021, Kuala Lumpur, Malaysia, July 26–29, 2021, Proceedings, Part II 34. Springer, 15–27.
24. Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and teaching programming: A review and discussion. Computer science education 13, 2 (2003), 137–172.
25. Mark R Shinn. 2007. Identifying students at risk, monitoring performance, and determining eligibility within response to intervention: Research on educational need and benefit from academic intervention. School Psychology Review 36, 4 (2007), 601–617. https://doi.org/10.1080/02796015.2007. 12087920
26. Atsushi Shirafuji, Taku Matsumoto, Md Faizul Ibne Amin, and Yutaka Watanobe. 2023. Rule-Based Error Classification for Analyzing Differences in Frequent Errors. In 2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE). IEEE, 1–7. https://doi.org/10.1109/TALE56641.2023.10398341

27. Raj Shrestha. 2022. Programming Process, Patterns and Behaviors: Insights from Keystroke Analysis of CS1 Students. Master's thesis. Utah State University.
28. Gabriel Silva de Oliveira, Zhikai Gao, Sarah Heckman, and Collin Lynch. 2024. Exploring Novice Programmers' Testing Behavior: A First Step to Define Coding Struggle. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1. 1251–1257. https: //doi.org/10.1145/3626252.3630851
29. B Tabarsi, Ally Limke, Heidi Reichert, Rachel Qualls, Thomas Price, Chris Martens, and Tiffany Barnes. 2022. How to catch novice programmers' struggles: Detecting moments of struggle in open-ended block-based programming projects using trace log data. In Proceedings of the 6th Educational Data Mining in Computer Science Education (CSEDM) Workshop, Virtual. https://doi.org/10.5281/zenodo.6983260
30. Zahid Ullah, Adidah Lajis, Mona Jamjoom, Abdulrahman H Altalhi, Jalal Shah, and Farrukh Saleem. 2019. A rule-based method for cognitive competency assessment in computer programming using Bloom's taxonomy. IEEE Access 7 (2019), 64663–64675. https://doi.org/10.1109/ACCESS. 2019.2916979
31. Luuk Van Waes and Peter Jan Schellens. 2003. Writing profiles: The effect of the writing mode on pausing and revision patterns of experienced writers. Journal of pragmatics 35, 6 (2003), 829–853. https://doi.org/10.1016/S0378-2166(02)00121-2
32. Luis Vives, Ivan Cabezas, Juan Carlos Vives, Nilton German Reyes, Janet Aquino, Jose Bautista Cóndor, and S Francisco Segura Altamirano. 2024. Prediction of students' academic performance in the programming fundamentals course using long short-term memory neural networks. IEEE Access 12 (2024), 5882–5898. https://doi.org/10.1109/ACCESS.2024.3350169