Advancing computer science education: strategies for student support and teacher development

by

James, Friday Emmanuel

B.S., University of Agriculture, Makurdi - Nigeria, 2009
M.S., University of Ilorin - Nigeria, 2015
M.S., Kansas State University, 2021

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R.Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2025

Abstract

The rapid evolution of computer science education calls for innovative strategies to support both students and teachers to ensure equitable learning opportunities and effective teaching methodologies. This dissertation contributes to the broader goal of improving computer science education by proposing novel data-driven approaches to identifying struggling students while strengthening teacher preparedness.

The first contribution of this dissertation focuses on leveraging the granularity of keystroke data to analyze students' programming behaviors and identify early indicators of struggle. Programming assignments play a crucial role in developing problem-solving skills and computational thinking; however, students — especially in introductory level programming courses — frequently struggle with grasping the syntax and logical structure of their code leading to high dropout and failure rates. To address this challenge, this work introduces, *From Typing to Insights*, a code visualization tool that reconstructs students' coding processes from keystroke logs and automatically generates execution logs against unit tests at different time intervals, offering deep insights into students' coding habits, debugging patterns and logical progression in problem-solving.

Building upon this, the second contribution integrates keystroke analytics with self-reported student experiences to propose *TrackIt*, a novel rule-based detection system that analyzes students' keystroke data to classify students into different struggle categories. *TrackIt* features a copy-paste detection functionality, which flags students who paste large portions of code, including those potentially from AI-generated tools. The tool enables a more precise understanding of students' learning difficulties. Additionally, *TrackIt*, combined with the

baseline reports, helps identify the most challenging concepts in introductory programming courses, thereby providing instructors with valuable insights to refine instructional approaches.

The third contribution shifts focus to teacher development, addressing disparities in computer science education, particularly in rural and underserved communities. Although CS education has expanded in recent years, access remains uneven due in large part to a shortage of qualified instructors. We investigate how participation in a structured teacher training program influences teachers' professional computing identities, commitment and overall confidence and competence in teaching computer science. Following training, the findings show that rural teachers reported positive changes in their identities and teaching competencies and are more likely to advocate for more students to take computer science courses. Teachers in rural areas also showed a marked improvement in confidence and commitment to teaching computer science.

Finally, to further understand teachers' learning perspectives, the fourth contribution of this dissertation conducts in-depth study of teacher reflective journals, utilizing both human annotations and Large Language Model (LLM)-based analysis. By examining teachers' insights into their learning experiences, instructional challenges and educational growth, this research contributes to the broader discourse on teacher development. These contributions present a comprehensive framework for advancing computer science education.

Advancing computer science education: strategies for student support and teacher development

by

James, Friday Emmanuel

B.S., University of Agriculture, Makurdi - Nigeria, 2009
M.S., University of Ilorin - Nigeria, 2015
M.S., Kansas State University, 2021

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R.Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2025

Approved by:

Major Professor
Dr. Joshua L. Weese

Copyright

Abstract

The rapid evolution of computer science education calls for innovative strategies to support both students and teachers to ensure equitable learning opportunities and effective teaching methodologies. This dissertation contributes to the broader goal of improving computer science education by proposing novel data-driven approaches to identifying struggling students while strengthening teacher preparedness.

The first contribution of this dissertation focuses on leveraging the granularity of keystroke data to analyze students' programming behaviors and identify early indicators of struggle. Programming assignments play a crucial role in developing problem-solving skills and computational thinking; however, students — especially in introductory level programming courses — frequently struggle with grasping the syntax and logical structure of their code leading to high dropout and failure rates. To address this challenge, this work introduces, *From Typing to Insights*, a code visualization tool that reconstructs students' coding processes from keystroke logs and automatically generates execution logs against unit tests at different time intervals, offering deep insights into students' coding habits, debugging patterns and logical progression in problem-solving.

Building upon this, the second contribution integrates keystroke analytics with self-reported student experiences to propose *TrackIt*, a novel rule-based detection system that analyzes students' keystroke data to classify students into different struggle categories. *TrackIt* features a copy-paste detection functionality, which flags students who paste large portions of code, including those potentially from AI-generated tools. The tool enables a more precise understanding of students' learning difficulties. Additionally, *TrackIt*, combined with the

baseline reports, helps identify the most challenging concepts in introductory programming courses, thereby providing instructors with valuable insights to refine instructional approaches.

The third contribution shifts focus to teacher development, addressing disparities in computer science education, particularly in rural and underserved communities. Although CS education has expanded in recent years, access remains uneven in large part due to a shortage of qualified instructors. We investigate how participation in a structured teacher training program influences teachers' professional computing identities, commitment and overall confidence and competence in teaching computer science. Following training, the findings show that rural teachers reported positive changes in their identities and teaching competencies and are more likely to advocate for more students to take computer science courses. Teachers in rural areas also showed a marked improvement in confidence and commitment to teaching computer science.

Finally, to further understand teachers' learning perspectives, the fourth contribution of this dissertation conducts in-depth study of teacher reflective journals, utilizing both human annotations and Large Language Model (LLM)-based analysis. By examining teachers' insights into their learning experiences, instructional challenges and educational growth, this research contributes to the broader discourse on teacher development. These contributions present a comprehensive framework for advancing computer science education.

Table of Contents

List of Figures

List of Equations

List of Tables

Acknowledgements

This research was made possible through the guidance, mentorship and valuable feedback received during the doctoral program. I would like to acknowledge my PhD advisor, Dr. Joshua L. Weese, for providing direction and scholarly insights that shaped the depth and clarity of the work. I also thank Professors Dr. Doina Caragea, Dr. William Hsu, and Dr. David Allen for serving on my thesis committee and providing suggestions on improving and extending my work.

To every member of the Advanced Learning and Teaching in Computer Science (ALT+CS) Lab – especially Dr. Nathan Bean and Russell Feldhausen – thank you for your immense support, contributions and ensuring that I have all the data needed for this research.

I am profoundly thankful to my loving and supportive wife, Mrs. Ojonemile Emmanuel, whose prayers, encouragement, patience and belief in my dreams sustained me at every moment, especially through the most challenging moments of this journey. Mentions must be made of my parents, Mr. & Mrs. James Ali and siblings, for their unwavering faith in my vision and constant prayers that have been a bedrock of strength for me.

I am especially grateful to Mrs. Barb Sanderson who has been like a mother to me and my family, for providing us with constant prayers, physical and spiritual care and encouragement that have made lasting impacts on our lives.

Finally, I extend my appreciation to every member of the Computer Science department, especially the office staff for their constant availability and timely responses to my requests.

Chapter 1 - Introduction

1.1 Background and Motivation

The world economies and societies have unarguably witnessed fundamental and remarkable transformation over the past few decades. This transformation comes with the evolution of technological advancements, which have become part and parcel of almost every endeavor of human lives, evident from the way we approach problems, make decisions and create interactions among ourselves. In fact, technological advancements have shaped the trajectory of human civilization throughout history – from the early days of material transformation in the Stone, Bronze and Iron Ages to industrial revolutions, leading to a dramatic shift from less than 1% of the world's stored information in digital format in the late 1980s to over 99% by 2012 [1].

The role of technological innovation in economic development and overall human progress cannot be overemphasized. According to economic theories (particularly the Solow-Swan model [2] [3]) long-run economic growth characterized by capital accumulation, labor and population growth, innovations in products, processes and businesses which brings about better standards of living and increases in overall productivity are largely driven by technological processes [4]. The technological advancements have also significantly shaped the dynamics of social life with the rapid growth of the internet and mobile technologies which have replaced the use of time-space constrained traditional approaches to global communication, thereby bringing about vast amounts of data [1]. In addition to the economic and social impacts, technological advancements play a crucial role in driving sustainable development. For instance, the 2030 Sustainable Development Agenda places a high emphasis on the role of digital transformation in achieving global sustainability goals [5].

Computer science has become the backbone of these advancements. The reliance of the society on technology has made the role of computer science education to evolve from a specialized field of study to a critical and broader component of education and learning in general as more emphasis on data-driven decision making and extensive adoption of Machine Learning/Artificial Intelligence (ML/AI) have led to innovations across diverse sectors – ranging from healthcare, finance, and entertainment to sports, security, and education.

With different aspects of computer science underpinning these advancements, there is a growing demand for the workforce in these sectors to gain expertise and become proficient in applying computer science skills, making computer science education a crucial part of today's academic programs.

In response to these demands, institutions of learning across the globe are deeply embedding computer science education in their programs to prepare students for the future. The introduction of computer science at an early stage is crucial for developing critical skills such as creativity, logical reasoning and problem-solving, which are requisite skills needed in a technology driven world [6]. These competencies have been demonstrated to have positive influence in the cognitive development of students and helps them keep pace with the rapidly changing teaching-learning process [7].

1.2 Problem Statement

Despite the significant advancement in computer science education, many students continue to struggle with programming, often without immediate support or intervention. Programming skills are critical skills needed to meet the demand of today's global technological advancements. In fact, programming skills are essential for understanding and applying complex concepts in data structures, algorithms, software engineering, machine learning and artificial

intelligence. However, given the rigorous nature of programming which requires a strong foundation of logical reasoning and problem-solving skills, there is a strong need to support students who struggle while writing computer programs particularly those who may not have had prior exposure in computer science or who have educational backgrounds that does not relate to computer science. While some students quickly grasp fundamental concepts and develop confidence in problem-solving within a short time, others face persistent difficulties that tend to hinder their progress. Challenges such as lack of understanding syntaxes, inability to debug errors, misunderstanding of control structures, amongst others, could create significant barriers to student learning. Detecting students' challenges early enough allows instructors to intervene in a timely manner [8]. This early intervention is particularly important in fundamental programming courses where the basic concepts are important for succeeding in the course. Without early support, students may continue to face difficulties grasping the keys concepts as many programmers, especially novices have been studied to face difficulties in understanding these fundamental concepts, debugging errors and developing efficient problem-solving strategies [9] [10].

Traditional assessment methods primarily rely on final exam scores, which provide limited insights into the iterative problem-solving processes students engage while coding their programming assignments [11]. As a result of this, students who struggle with the logic of coding, debugging or any time of compile-time or runtime errors may nor receive the necessary support early enough, which may lead to high dropout rates especially in introductory programming. Moreover, grades alone provide an incomplete picture of student learning as they only reveal the outcome but not the process of how students arrive at their solutions. This makes it difficult for educators to pinpoint specific areas where the students struggle and offer support.

For instance, a student who has regular attendance in classes and turns in assignments on time may have struggled while writing their programs. Without early identification of these students and providing the necessary support and intervention, students may lose confidence in their abilities and eventually drop out of the program, an outcome which not only affect the student, but also the institution as it will leads to a decline in student retention and overall effectiveness of the program.

In addition to supporting students, another important factor in the success of computer science education is the role of teachers. Teachers are instrumental in promoting an inclusive and effective learning environment [12], yet many educators, particularly those in rural or underserved areas, lack the necessary training and resources to teach computer science effectively [13]. Computer science (CS) education gained a significant amount of attention after the publication of a disturbing report titled "*Running on Empty: The Failure to Teach Computer Science in the Digital Age"* by [14], which revealed outrageously low figures for women in computing and highlighted that more than two-thirds of the country's population were lacking the standards for a comprehensive computer science program at the secondary school level [15]. Consequently, countries and academic scholars began to place high values and emphasis on the need for children to develop a strong foundation and gain fundamental understanding of computer science [16]. This includes the development of digital tools to aid computational thinking (a problem-solving approach associated with the field of computer science [17]) and collaborative learning from an early age to keep pace with the rapidly changing teaching-learning process [7].

The expansion of computer science education promises feasible results, given that computer programming has recorded profound positive effects on the meta-cognitive ability, reflectivity, and divergent thinking in young children [18], and when given age-appropriate technologies, curriculum and pedagogy, young children can actively engage in learning from computer programming and take the first steps in developing computational thinking [19].

The National Center for Educational Statistics (NCES) defines locales by four categories - Urban, Suburban, Town and Rural. Each of these categories are broken down further into sub-groups [20]. Although each grouping by NCES are widely recognized and used for educational research, each category presents unique challenges [21], specifically with respect to bringing computer science education to the rural schools, which is seen as a persistent CS educational challenge [22]. Access to computer science education is less pronounced in schools in the rural areas compared to their more urbanized counterparts [23]. Although Broadening Participation in Computing (BPC) education projects have been effectively implemented in some states in the United States of America, such as Maryland [24], California [25], and Utah [26], rural schools are not still within sufficient reach largely due to their geographical disadvantages [27]. An attempt to identify the implementation challenges for a new computer science curriculum in rural western regions of the United States also revealed that the concept of computational thinking and coding were foreign to the teachers whom required a pedagogical shift to teach CS [28]. Thus, computational thinking needs to be incorporated in rural education as it helps to build structural thinking, critical power and creativity [29].

In addition, understanding teachers' learning perspectives, identity confidence and measuring their commitment to teaching is crucial in designing and implementing impactful teacher training programs. Additionally, the disparities in computer science education between

the urban and rural regions highlight the need for specialized training initiatives that equip teachers with the skills and confidence to deliver high quality computer science instructions.

1.3 Research Objectives

This dissertation investigates these interconnected aspects of computer science education and seeks to advance computer science education by addressing the mentioned challenges through the following research objectives:

1. To develop an interactive tool that visualizes students' code progression and identifies error patterns using keystroke data.
2. To propose a rule-based system for detecting struggling programmers using students' reflective report as a baseline.
3. To assess the impact of teacher training on educators' professional identity, confidence, competence and commitment, particularly in rural areas and evaluate how structured training programs influence teachers' motivation and ability to effectively teach computer science.
4. To conduct in-depth thematic analysis of teachers' learning perspectives from teacher training program.

1.4 Dissertation Structure

This dissertation is structured into seven chapters and addresses specific research objectives while ensuring a cohesive discussion of findings and implications.

Chapter 2 presents a review of the key literature that informs the development of data-driven tools and frameworks aimed at improving student learning and teacher development in computer science education

Chapter 3 - explores the role of keystroke analytics in computer science education, reviewing existing literature on learning analytics and its applications. It proposes an analytical tool — *From Typing to Insights* — and presents the methodologies used in the study, including data collection, processing and system development of the tool. Key features of *From Typing to Insights* include code reconstruction, external code editor integration and error correction analysis. The chapter concludes by discussing how these insights can improve programming and student learning experiences, as well as how instructors can best provide early intervention to prevent massive failures or early dropouts.

Chapter 4 - builds on Chapter 3 and proposes a *TrackIt*, a rule-based detection system for identifying struggling programmers using keystroke data and incorporating students' survey reports as a baseline for validation. It details the engineering of key features constructed from keystroke data that serve as indicators of struggle for students in introductory programming courses, ultimately demonstrating how keystroke logs can provide instructors with early warnings about students at risk of dropping out or failing.

Chapter 5 - evaluates the impact of teacher training programs on teachers' identity, confidence, competence and commitment to the teaching profession in rural centric areas. It begins with an introduction and overview of related work on the scarcity of CS education in rural areas, details a theoretical framework covering identity, confidence, competence and commitment, and describes existing computer science teacher training programs. This is followed by detailed discussion of the data collection, research design and data analysis. The chapter concludes with presentation of results and highlighting the impact of the training programs on the key aspects of the teaching profession.

Chapter 6 - focuses on computer science teacher education by analyzing reflections from teacher training programs. By comparing human and automated analyses from Large Language Models, it uncovers key insights into teachers' challenges, motivations and strategies in teaching computer science concepts.

Chapter 7 - synthesizes the key findings from the dissertation and offers actionable recommendations for improving computer science education. It also outlines future research directions.

Chapter 2 - Literature Review

As mentioned in Chapter 1, this dissertation responds to both needs of ensuring that students receive timely support when they struggle, and equipping teachers with the tools, mindsets and pedagogical strategies to teach computing effectively. This chapter presents a review of the key literature that informs the development of data-driven tools and frameworks aimed at improving student learning and teacher development in computer science education. It explores core areas including learning analytics, keystroke dynamics, approaches to detecting student struggles, rule-based systems, and the evolving elements of effective computer science teaching — identity, commitment, confidence, and competence. The bodies of work reviewed are not only foundational to understanding the current landscape but also align with the overarching research objectives of this dissertation.

2.1 Overview of Learning Analytics

Learning Analytics is the measurement, collection, analysis, and reporting of data about learners and their contexts, for the purposes of understanding and optimizing learning and the environments in which it occurs [30]. 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 [31]. 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 towards providing intervention to struggling students [32]. It has developed into an essential tool used by educational institutions to enhance both student learning and how educators provide intervention through the use of data mining machine learning and visualizations [31]. 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 underperforming [33].

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), Naïve 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 [33]. 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 [34], eye-tracking data is useful for evaluating intersubjectivity in face-to-face collaboration, log data cand be used to predict student performance and contrition in team work for project-based learning, automated dialog data can be used to predict the coming of newcomers in online learning. Also, other forms of multiple forms of data such as audio, student logs, 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 [35].

Learning analytics has had significant impact in the ability to analyze students' learning outcomes by providing valuable insights into students' behavior and learning processes but has not yet been fully utilized beyond small-scale studies to cover larger and more complex education-related issues – for example, improving educational policies [36]. Although existing policies provide valuable supports for data privacy and ethical use as well as engagement of students, a more comprehensive policies that have more pedagogical applications [37].

Within the context of learning analytics, the development of the interactive tool *From Typing to Insights* —which aims to analyze students programming keystroke dynamics — aligns well with the overall goal of learning analytics by aiming at providing insights into identifying 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 Research

Keystroke dynamics also referred to as keystroke biometrics or typing dynamics, or typing biometrics refer to the collection of biometric information generated by key-press-related events that occur when a user types on a keyboard [38]. The main concept behind using keystroke dynamics as an authenticator is to detect the unique patterns that exist when a user types on a keyboard [39]. 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 [40].

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 [41]. In a similar fashion, Shrestha R. [42] explored the use of keystroke data from students in detecting struggling students by analyzing the students' thought processes, typing and programming patterns. The analysis of the pausing behavior, pause-frequencies of different lengths and the last keystroke action before pausing correlated with exam scores provided insight into identifying struggling students [42].

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 early in the course for effective intervention with keystroke data that can be collected without having to place unexpected burden on the instructor. This is done by analyzing keystroke data comprising time-on-task, typing speed and pauses [43]. The correlation between students' pausing behavior and learning outcomes obtained through keystroke dynamics also provides an understanding of students' coding strategies [44]. While keystroke data can be modeled to predict students' success, the models may not generalize well across different contexts and that it could be misleading to rely solely on a single semester data [45].

Keystroke dynamics transcends 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. [46] used a combination of keystroke data with survey 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 much 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 [47]. Keystroke data can provide information that signals typing patterns that are induced by stress which can be used to detect users' emotional states [48]. Specifically, emotional states such as confidence, hesitance, nervousness, relaxation, sadness and tiredness have been modeled using keystroke dynamics with a potential of identifying anger and excitement [49]. In addition, keystroke has been found to be useful in detecting engagement and boredom. Research conducted by Bixler and D'Mello

[50] which used keystroke data obtained from students typing three different essays from topics 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 Struggle Detection in Programming Courses

2.3.1 Survey/Interview-Based Detection

Surveys are widely used in educational research and have played a significant role in identifying struggling students in programming courses. By collecting self-reported data, surveys provide valuable insights into students' perceptions, challenges and learning behaviors. These surveys are either used independently or as complementary methods of identifying struggling students.

One of the earliest survey-based studies on programming struggles was conducted by Bennedsen and Caspersen [51]. Their research aimed to quantify the failure rates in introductory programming courses by collecting data through a web-based survey questionnaire where institutions reported pass, fail and dropout numbers along with details about course structure and evaluation methods. This study highlights the importance of survey-based struggle detection as it establishes a baseline for measuring student difficulties across different institutions. Self-reported factors such as prior academic background, self-perceived programming ability and comfort level has been used to investigate student struggle with programming courses, and students who rated their understanding of programming concepts were more likely to succeed where those with low self-efficiency were at greater risk of struggling [52].

Semi-structured interviews has shown that students frequently experienced frustration, anxiety and self-doubt while completing programming tasks, which significantly impacted their

ability to persist and succeed [53]. Surveys have been used to identify the specific programming concepts that students find most challenging – recursion, loops and object-oriented programming, often leading to frustration and poor performance [54].

A combination of survey data and real-time tracking of students' learning management revealed that many students struggle with time management, procrastination and ineffective help-seeking behaviors, all of which leads to poor performance [55]. Gorson et al. (2021) [56] examined the relationship between student perceptions and programming struggles and found that many students judged their abilities based on routine programming challenges such as encountering syntax errors and debugging issues.

Survey-based approaches to identifying struggling students enable the instructor to capture the psychological and emotional dimensions of struggle, providing a holistic view of student struggle through cognitive and behavioral data. In addition, by tracking students' experiences through longitudinal surveys, instructors can get valuable feedback on the effectiveness of teaching approaches which in turn helps instructors to refine course design and teaching methodologies. However, self-reporting bias is a significant concern, as students may either under-report or exaggerate their struggles. Moreover, surveys rely on periodic data collection, which may not provide the immediate insights that keystroke tracking or learning analytics can offer. By combining survey-based methods with other forms of struggle detection such as keystroke analysis or real-time feedback mechanisms, instructors can develop a more comprehensive approach to identifying and addressing students that are struggling.

2.3.2 Detection Based on Coding Behavior and Error Analysis

Error analysis has been studied as an important approach for identifying struggling students in programming courses. This method primarily focuses on compilation and runtime

errors, debugging behaviors and error resolution patterns, in order to detect students that exhibit struggling habits while writing their codes. Several studies have investigated different aspects of error-based struggle detection, which reveal insights into how students encounter and resolve errors in programming tasks.

Altadmri and Brown [57] conducted an intensive one-year study of 37 million compilations from over 250,000 students all over the world. By utilizing data from a diverse range of students and institutions, the study provided a perspective on error frequencies, time-to-fix errors and the repetition of specific mistakes. They found that syntax errors (missing semicolons, mismatched parenthesis, confusing the assignment and equality operators) were more pronounced among the students. They also observed that while these syntax errors decline over time, more complex errors from semantics emerged as the students made progress.

In a similar fashion, Lundberg [58] used data from 264 students in a programming class to develop a predictive model that could identify students who were at risk of failing the course. The study found that students who frequently encountered the same types of errors without resolving them were more likely to struggle.

Geng et al. [59] categorized students into different learning groups based on their error resolution patterns. Their study divided students into categories of “Quick Learners”, “Hardworking”, “Satisficing” and “Struggling”. The found that the “Struggling” group exhibited high frequencies of syntax and semantic errors and took longer time to resolve the errors and concluded that clustering students based on their debugging efficiencies and error repetition could provide valuable insights for early intervention.

James et al.. [60] leveraged the granularity of keystroke data to study students’ programming errors and time taken to fix errors by reconstructing their code snippets using the

keystroke data and running them against unit tests at different intervals within the duration of writing their programs. The study revealed that most introductory programming students struggle with syntax errors and take longer time to fix the bugs in their codes.

Understanding how students debug their code can also reveal indicators of struggle. Repeated failure of same unit tests for more than four consecutive submissions have been revealed as indicators of students struggling [61], with findings revealing that struggling students often engaged in trial-and-error debugging without fully understanding the underlying problems. Oliveira et al. [61] further found that students who did not modify their unit tests to explore different solutions were more likely to experience persistent struggles. These results highlight the importance of incorporating debugging behavior analysis in addition to error tracking.

In a similar study, Tabarsi et al. [62] developed a model for detecting struggle moments by analyzing students' code traces in open-ended programming assignments. Their model identified struggle indicators to include long periods of inactivity, frequent re-runs of unchanged code and repeated deletions and rewriting of code segments. However, they opined that using open-ended assignments are less effective ways of identifying struggling students and proposed the use of Interactive Development Environments (IDEs) with built-in analytics tools to track students' problem-solving processes.

While coding behaviors and error analysis provides indicators of struggle in programming courses, they are ladened with some limitations. McCall and Kolling [63] however examined the limitations of using compiler error messages as indicators of struggle in programming courses. They emphasized that compiler messages often fail to differentiate between different logical errors, thereby leading to inaccurate interpretations of students'

misconceptions. They further argued for a more detailed classification of errors that accounts for underlying cognitive difficulties rather than merely relying on recorded compiler messages.

2.3.3 Data Mining and Machine Learning Approaches

The challenge of identifying struggling students in programming courses has led to the adoption of machine learning approaches in an attempt to detect at-risk students early for the purpose of providing interventions. Various machine learning models – ranging from neural networks, decision trees to support vector machines (SVMs) and ensemble methods, have been developed to analyze students' data with the goal of prediction dropouts, low performance and the need for assistance.

Castro-Wunsch et al. [64] employed a shallow neural network model to analyze source code snapshots in CS1 programming courses, using the number of steps required to complete the assignment and code correctness (defined as the proportion of tests passed). The researchers found that the neural network models were pessimistic – they often over-identified students as struggling but resulted in fewer false negatives, where struggling students were being overlooked. They also explored a range of Bayesian, decision tree, rule learner, along with the neural network models and found that the performance of most of the models was fairly stable across contexts and, more importantly, across terms in the same course.

Lokhande [65] explored deep neural networks for predicting student performance by classifying students as low, medium and high performers. The study claimed to outperform traditional machine learning approaches, with an accuracy of 85.4%. However, the study does not capture any meaningful information regarding the type or source of data and does not offer clear reasons for grouping the students.

Long Short-Term Memory (LSTM) neural networks have also been employed for early identification of struggling students in programming courses. Vives et al. [66] utilized LSTM approach to predict the academic performance of 677 students' records during the seventh, eighth, twelfth and sixteenth week of the academic semester in order to identify students at risk of failing the course. Due to data class imbalance, Generative Adversarial Network (GAN) and Synthetic Minority Over-sampling Technique (SMOTE) were used to balance the data. They reported a classification accuracy of 98.3% and performed better than Deep Neural Network (DNN), Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), Support Vector Classifier (SVM), and K-Nearest Neighbor (KNN).

A study combining word embeddings (a Natural Language Processing technique – NLP) and LSTM recurrent neural networks based on the premise that students' early-stage code resembles natural language more than formal programming syntax was carried out to predict whether students will need assistance based on the structure of their code [67]. The research findings showed that deep learning models can achieve expert-level performance in predicting student struggles. However, the researchers highlighted that hand-crafted feature engineering models are more interpretable from a pedagogical point of view.

Research by Liao et al. [68] explored a data-driven linear regression methodology for early identification of at-risk CS1 students using clicker-based student response in Peer Instruction (PI) classrooms. The findings indicate that by analyzing clicker response data from Peer Instruction classes, the model is capable of classifying 70% of students as at-risk or not at-risk by the third week of the course with a false negative rate of 17%.

Despite significant advancements in the use of data mining and machine learning approach to identifying struggling students in programming courses, a critical research gap

remains in the ability to provide interpretable and adaptable struggle detection for students. While these models have demonstrated high accuracies in forecasting struggles and classifying students, they often act as black boxes, making it difficult for instructors to actually know why a particular student is flagged as struggling. Also, many of the machine learning models are trained on course-specific datasets and fail to effectively transfer their functionalities when applied to new datasets. Moreover, machine learning and data mining techniques rely heavily on extensive historical data for training. This may not always be available in new programming courses, which limit their applicability. When data is not sufficiently large, especially in cases of data imbalance, the researchers (for example [66]) would resort to the use of synthetically generated data to make up for the imbalances in the data which may not conform to the real world data.

2.4 Rule-Based Systems in Education

Several studies have explored the use of rule-based systems in computer science to assess students' competency levels for the purpose of providing guided feedback. Ulla et al. [69] developed a rule-based cognitive competency assessment system that evaluates students' programming skills using Bloom's Taxonomy – a guide for assessing learning objectives of a particular field across cognitive, effective and psychomotor domains [70]. The research automatically maps written codes to cognitive skills levels (such as analyzing, applying or understanding) and uses the mapping to quantify students' proficiency. They found that rule-based competency assessment aligns well with manual evaluations.

A rule-based error classification system that categorizes common programming errors across different experience levels was developed by Shirafuje et al. [71]. The study analyzed 95,631 code submissions from an online judge system and applied regular expressions and structured rules to classify errors into syntax, semantic and logic errors, and revealed that novice

programmers frequently made syntax-related errors while expert programmers made more logic errors.

Rule-based Online Judge (OJ) recommender system was introduced by Rahman et al. [72]. The system categorizes students based on submission correctness, efficiency and complexity by applying association rule mining, thereby identifying clusters of students and providing recommendations. Their findings reveal that rule-based clustering can effectively group students based on skill level.

Associative classification models based on rules performs comparatively to traditional machine learning classifiers, as revealed in a study that utilized associative classification model to predict student academic performance based on students' historical coursework [73].

2.5 Core Elements of Effective Computer Science Teachers

2.5.1 Teacher Identity

Teacher identity stands at the core of the teaching profession [74]. It is broadly defined as being recognized as a certain kind of teacher by self or by others [75] and can be perceived as a multifaceted construct that covers the beliefs, values, and perceptions of an individual about their role as an educator [76]. Teacher professional identity is dynamic and changes over time based on internal factors, such as emotion, and external factors, such as life experiences and exposure [77]. Not only does identity provide a great framework in understanding computer science teacher preparation and professional development [78], it is also one of the factors of consideration for entering into the profession of teaching computer science [79].

We can therefore conclude that teacher identity plays a key role in shaping the pedagogical approaches and overall effectiveness of a teacher. It is thus imperative to evaluate

the impact of the teacher training program on the unique identities of the teachers under study and how they develop.

2.5.2 Commitment

Teachers' commitment plays a central role in the expansion and, subsequently, the sustenance of computer science education, both on a rural and urban scale. Teacher training supports educators by boosting their commitment and confidence in their ability to teach computer science as well as leading students in completing course capstone projects [80]. Mentoring experiences have also been shown to build teachers commitment to computer science education [81]. Following an instructional coaching program, teachers in underserved rural areas looked beyond infrastructural challenge, such as wireless access, to exhibit very keen interest in the development of computer science master teachers [82]. Essentially, efforts to provide comprehensive professional teacher training programs are critical for ensuring teachers' commitment and motivation to delivering efficient computer science education to students.

2.5.3 Confidence and Competence

Many K-12 teachers are new to the teaching of computer science, which requires new disciplinary knowledge and skills. Teachers' confidence, or self-esteem, is one of the factors that affect teaching, and it is imperative for teachers to feel confident in their ability to deliver computer science education [83]. Understanding teachers' confidence level is so essential that it allows us for adjustment of the contents of the teacher professional development program in order to meet teachers' needs [84]. In addition, competence in teaching computer science plays a significant role and must be bolstered if expansion of computer science education is to be achieved. Studies reveal that insufficient number of trained and capable teachers is a common barrier to the broad-based adoption of computer science in secondary schools [85]. All of these

highlight the effectiveness of professional training on teachers' confidence and competence whilst preparing for a career in teaching computer science.

2.6 Teacher's Perspectives and Attitudes Toward Computer Science

A growing body of research highlights that teachers' learning perspectives during professional development are shaped not only by the structure and content of the training, but also by their pre-existing beliefs, attitudes and dispositions towards the subject matter – in this case, computer science. These attitudes do not only influence teachers' engagement with professional development, but also how they reflect on what they have learned and ultimately translate that learning into classroom practice.

Prior studies have shown that teachers' perceptions can significantly impact both the learning process, instructional outcomes as well as participation in the training programs [86]. Early research reveals that teachers often have concerns and anxieties regarding related to the effects of computer science instruction, particularly regarding their ability to grasp unfamiliar technologies and allocating time for learning and integrating them into their practice [87]. Many teachers, particularly those without a formal background in computer science, approach the subject with hesitations that may come from a perceived difficulty of the subject, a lack of professional preparation or misconceptions about who can or should participate in computing [88]. For example, findings from Margolis et al. [89] revealed that teachers often hold the belief that success in computer science requires innate intellectual or technical aptitude. This perception reinforces the limiting assumption about who belongs in the field and can indirectly restrict inclusive teaching practices.

Studies have also highlighted how gendered and racialized perceptions of computer science influence teacher attitudes. Goode [90] pointed out that some educators unconsciously

reproduce systematic barriers by assuming that students from underrepresented backgrounds may not be interested in or capable of excelling in computing. These assumptions, though often implicitly reflected, can limit access for students from underrepresented backgrounds by influencing classroom expectations and curriculum choices.

Positive attitudes, on the other hand, are associated with teacher participation in professional development programs. When teachers receive scaffolded exposure to computer science concepts – including participation in coding workshops and communities of practice – their self-efficacy increases, leading to improved pedagogical confidence and a willingness to experiment their knowledge in their classrooms [91]. Moreso, when teachers are supported across multiple academic years, their attitudes tend to evolve over time. A longitudinal study [92] reveals that prospective teachers' attitudes towards computer technologies – particularly comfort and perceived usefulness – improved significantly over time with structured training and hands-on experience, highlighting the need for continuous and consistent teachers professional development programs.

These studies underscore the importance of understanding how teachers perceive computer science during and after professional development programs.

2.7 Thematic Analysis in Education Research

Thematic analysis is one of the most common forms of analysis within qualitative research. It emphasizes identifying, analyzing, and interpreting patterns of meaning (or "themes") within qualitative data [93]. This implies that it is a suitable and appropriate technique for learning from student experiences, curriculum evaluation, and teacher reflections by uncovering themes in textual data. In fact, Court et al. [12] emphasized the significance of nurturing reflective abilities among teachers undergoing professional training as it would help

the teachers reflect on their learning in a systematic manner [12]. Essentially, thematic analysis can be done in six steps – familiarization with the data, generating initial codes, generating themes, reviewing the themes, naming or defining the themes, and producing a final report [93]. In a similar study, the reflective practices of pre-service teachers were analyzed using thematic analysis which led to the development of improved self-awareness and teaching competences [94].

Thematic analysis of reflective feedback is applicable to Engineering education. Ghosh and Verma utilized thematic analysis in examining feedback in courses that are programming-heavy, the result which demonstrated the potential of thematic analysis in helping with course design and enhancing the learning outcomes of students, especially in technical fields [95]. In a similar manner, thematic analysis has been applied to students' reflection on pair programming in a CS1 course where affective, cognitive, and social dimensions of the experiences of the students were identified [96].

The increase in the volume of datasets generated from educational research has brought about the use and combination of computational methods with the traditional thematic analysis methods. For example, Gauthier and Wallace introduced the Computational Thematic Analysis Toolkit, which combines computational methods and conventional qualitative techniques to analyze a large online dataset [97]. Similarly, in [98], a system that incorporates computational tools with thematic analysis was designed to identify textual contents through thematic analysis of large datasets, the result of which stresses the significance of thematic analysis in handling big dataset in research on education.

Thematic analysis goes beyond analyzing reflective feedback and teacher reflections. It has been utilized in curriculum evaluation by assessing how well the contents of educational

programs prepare the students to become competent. Rahman et al. achieve this feat through the use of natural language processing to analyze a software testing course curriculum [99].

Chapter 3 - Leveraging Keystroke Data for Code Visualization

This section is adapted from a poster [8] *presented in SIGCSE 2025 and* [100] *accepted to ASEE 2025*

3.1 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 [101]. 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 [102] – 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 [103]. Computer science educators often resort to assignments and examination scores to assess students' level of understanding [104], 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 [105]. This is particularly important for large classes, where individualized attention is practically impossible [106].

To address this problem, we develop and evaluate an analytical tool *From Typing to Insights* that leverages keystroke data to identify struggling students by reconstructing and visualizing 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 processes such as insertions and deletions, offers 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 chapter therefore seeks to provide answers to the following research questions:

1. How can keystroke data be effectively utilized to understand analyze students' programming behaviors and identify indicators of struggling for early intervention?
2. How can code execution logs and error reports enhance 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.

3.2 Comparison of *From Typing to Insights* with Existing 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 [107]. 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 upon 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. [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.

3.3 Methodology

3.3.1 Data Collection and Description

The data was collected from Codio [108], 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 and strategies, behaviors and struggles for the entire duration of writing the programs. Table 3.1 shows the attributes of the Codio data while Figure shows a sample of the data along with skeletal starter code generated by Codio.

Table 3.1. Keystroke Data Attributes

Column	Description
Date	Contains the exact timestamp in the ISO 8601 format (for example 2022-09-05T00:13:53.698Z) and records when an event occurred.
Position	Shows the cursor position where character insertion or deletion happens.
Insert	Defines the character or lines inserted at the specified position.
Delete	Defines the number of characters deleted by the student at the specified position.
Version	An integer value that represents the version of the code at that timestamp.
User	An identifier for each student that works on the program. There is an "internalReload" user that represents an automatically generated action provided by Codio.

date	position	delete	insert	version	user
2022-08-27T05:02:10.110Z	0	0	import sys	0	internal#reload
2022-08-27T05:02:36.400Z	96			1	Student 1
2022-08-27T05:02:38.510Z	105		P	2	Student 1
2022-08-27T05:02:38.577Z	105	1		3	Student 1
2022-08-27T05:02:39.655Z	105		p	4	Student 1
2022-08-27T05:02:39.655Z	106		r	4	Student 1
2022-08-27T05:02:40.364Z	107		i	5	Student 1
2022-08-27T05:02:40.364Z	108		n	5	Student 1
2022-08-27T05:02:40.364Z	109		t	5	Student 1
2022-08-27T05:02:42.003Z	110		()	6	Student 1
2022-08-27T05:02:42.502Z	111		""	7	Student 1
2022-08-27T05:02:42.580Z	112		H	8	Student 1
2022-08-27T05:02:42.680Z	113		e	9	Student 1
2022-08-27T05:02:42.773Z	114		l	10	Student 1
2022-08-27T05:02:43.542Z	115		l	11	Student 1
2022-08-27T05:02:43.542Z	116		o	11	Student 1
2022-08-27T05:02:44.643Z	117			12	Student 1
2022-08-27T05:02:44.643Z	118		W	12	Student 1
2022-08-27T05:02:44.643Z	119		O	12	Student 1
2022-08-27T05:02:44.643Z	120		r	12	Student 1
2022-08-27T05:02:45.732Z	120	1		13	Student 1
2022-08-27T05:02:45.732Z	119	1		13	Student 1

```
import sys
class HelloWorld:

    @staticmethod
    def main(args):
        # Your code below


        # Your code above
if __name__ == "__main__":
    HelloWorld.main(sys.argv)
```

Figure 3.1. Sample Keystroke Data and #InternalReload Skeletal Code

3.3.2 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.

3.3.3 System Development and Design

The development of the *From Typing to Insights* 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.

3.3.3.1 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 types 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.

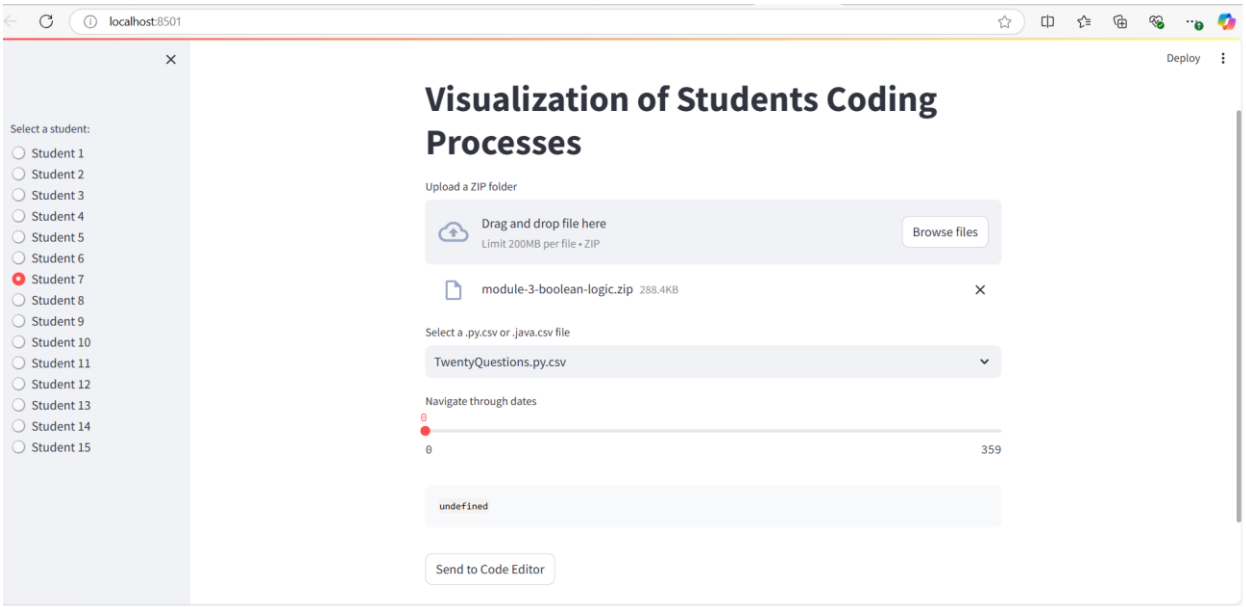


Figure 3.2. Interface Showing File Upload and Slider

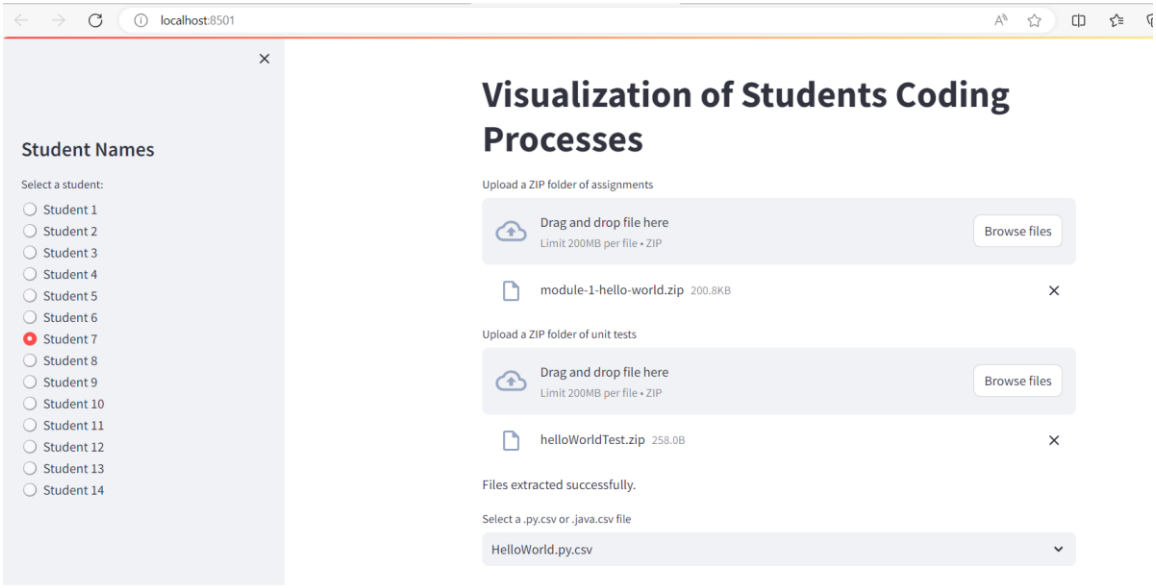


Figure 3.3. Interface Showing Unit Test Upload Functionality

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 students' solutions and to pinpoint the exact moments when the individual student encountered challenges or appeared to be struggling.

The constructed code snippets are displayed using Prism.js for syntax highlighting, providing a visually accessible representation of the code. 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 (timestamps, 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, revealing changes line by line and observing the students' approaches and how it shifts throughout their entire problem-solving process as shown in Figures 3.4 and 3.5.

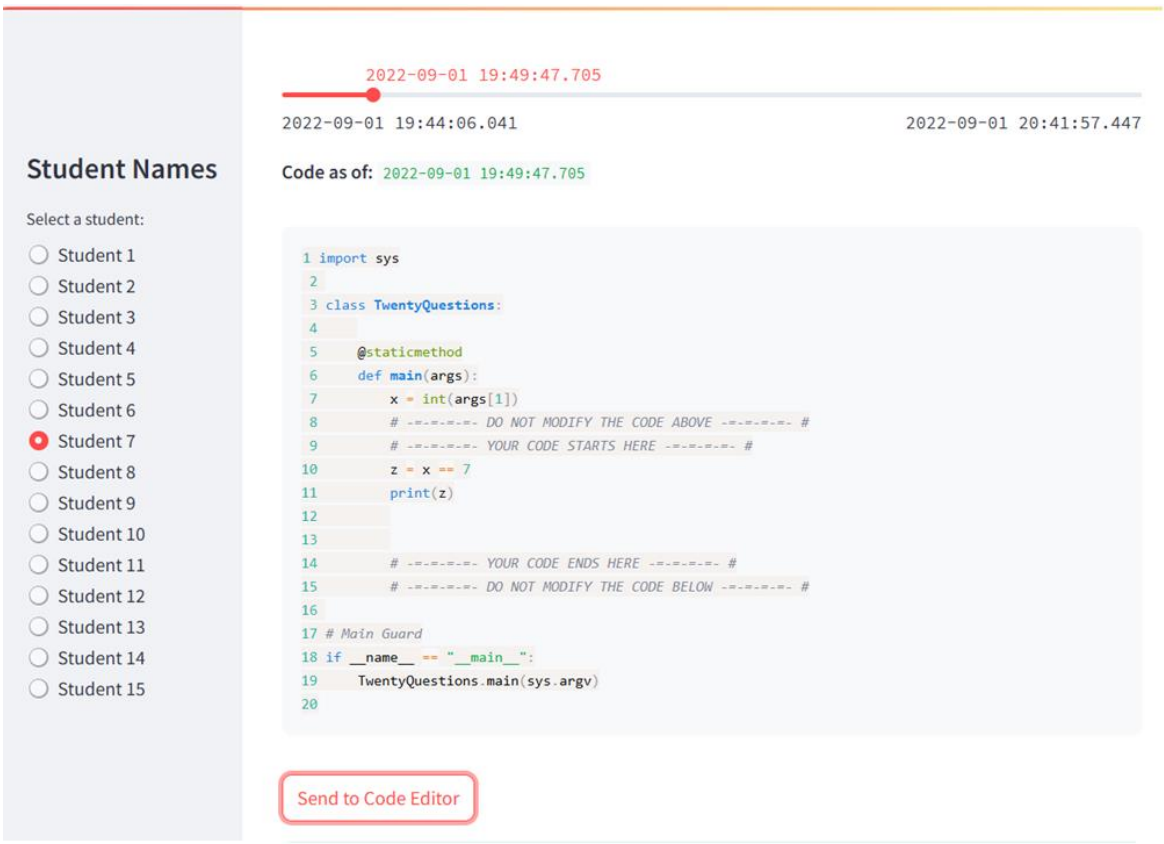


Figure 3.4. Interface Showing Slider Movements and Corresponding Code Snippets

3.3.3.2 Interfacing with External Code Editor

The application integrates error detection by interfacing with an external code editor execution environment shown in Figure 3.5. 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. 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.

The user interface as shown in Figure 2.4 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.

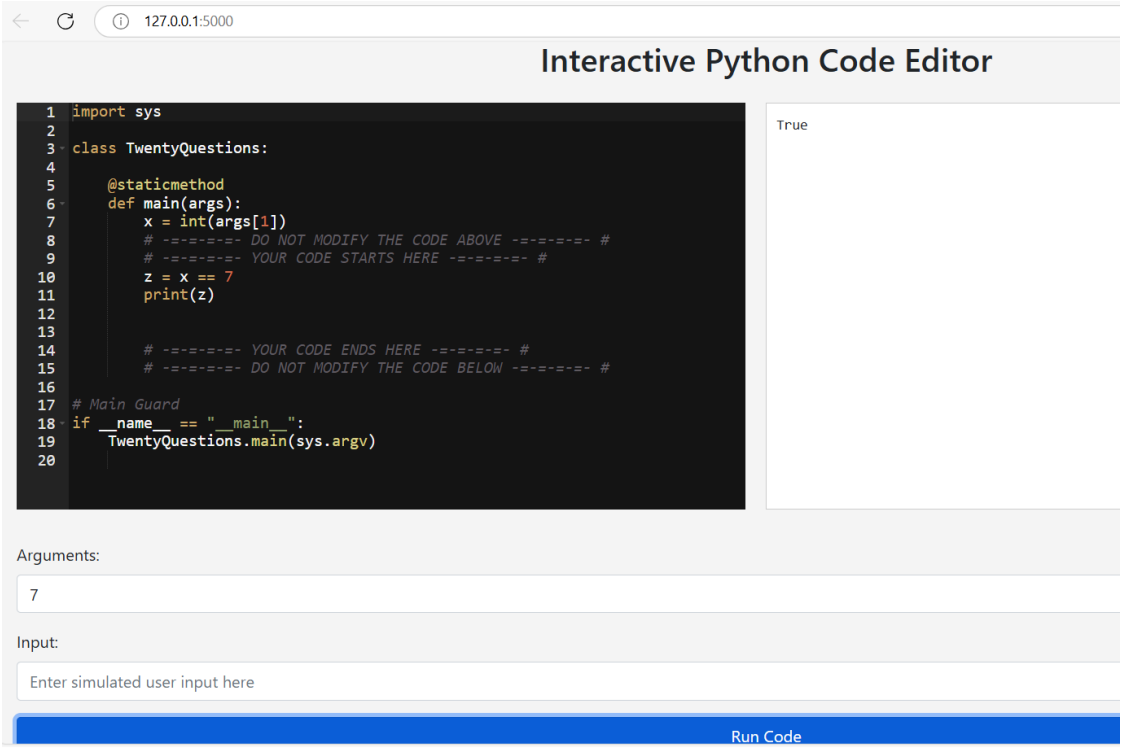


Figure 3.5. Interaction with Code Editor

3.3.3.3 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 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 3.3) 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.

Also, as part of the process, the system continuously tracks the pass/fail status of each test case and generates detailed execution logs that provides useful feedback as shown in Figure 3.6. Runtime errors, syntax errors or logical mismatches are detected during execution.

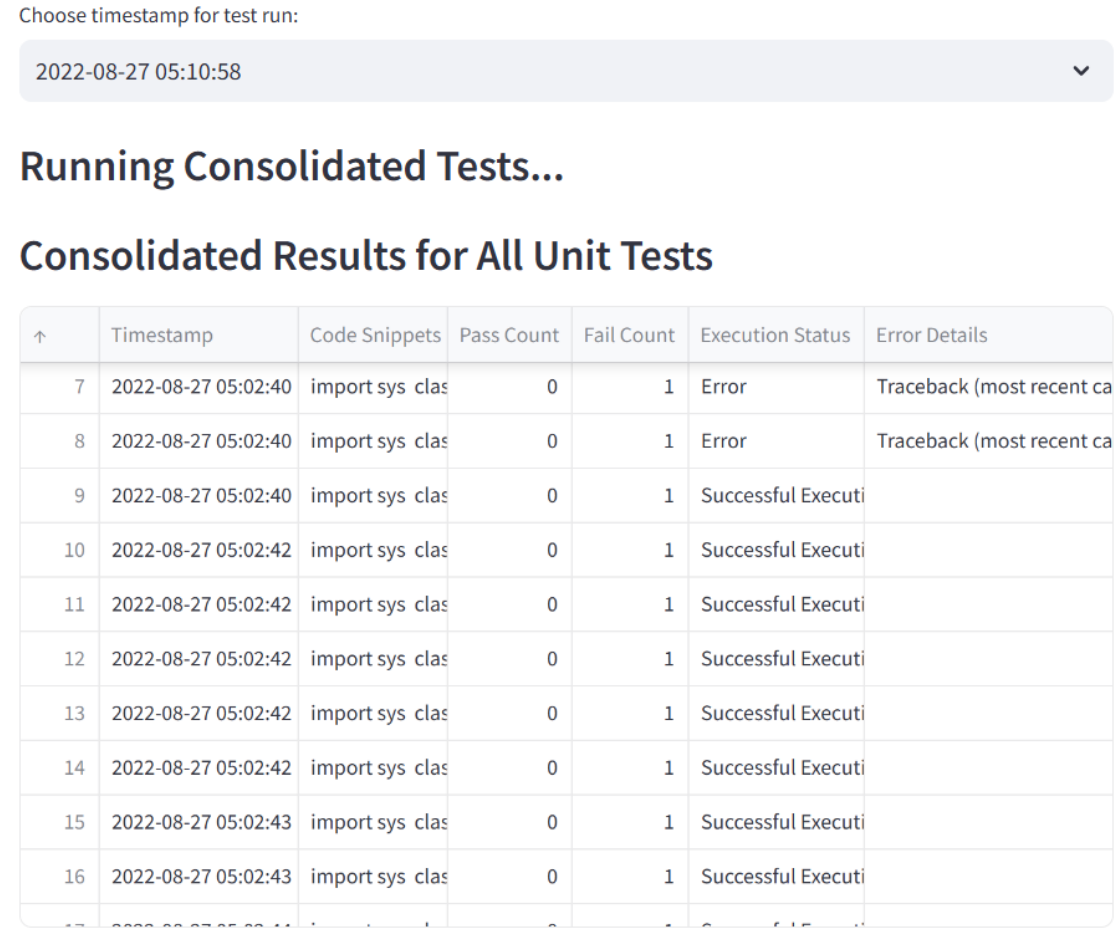


Figure 3.6. Execution Logs with Unit Tests

3.3.3.4 Error Correction Times

The application includes a feature that tracks error correction times by recording when each error occurs, when it is resolved, the type of error, and the duration taken to fix it — offering a detailed time of the debugging process. For example, as shown in Figure 3.7, 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.

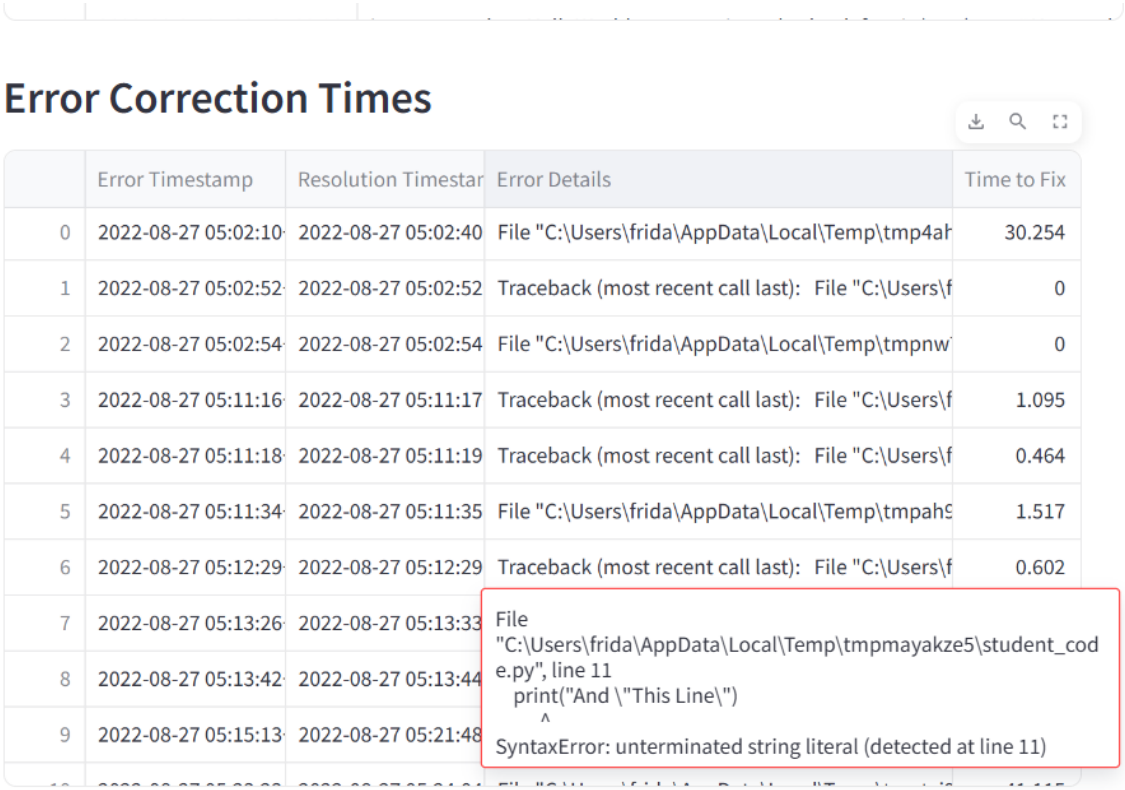


Figure 3.7. Error Correction Times

3.4 Discussion of Results

3.4.1 Understanding Students' Programming Behaviors and Potentially 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 code.

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 helps 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.

3.4.2 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 spend 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 by the tool as displaying 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 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.

3.5 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 ensures data security and prevents 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 purposes only. The instructors utilizing this tool will have access to student information to be able to provide early intervention where necessary.

3.6 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. 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.

3.7 Limitations

While From Typing to Insights provides valuable insights into students' coding behavior — inferring potential struggle through visual observation of patterns such as frequent deletions, prolonged pauses, repeated errors or long error correction times — it does not, in isolation, capture whether these struggles stem from external distractions or non-cognitive factors. To address this limitation, the next phase of this dissertation, as detailed in Chapter 4, integrates

keystroke data with self-reported reflections from students. This combined dataset serves as a baseline for a rule-based detection system, offering a more holistic assessment of struggle.

3.8 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 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.

The next chapter builds directly on this foundation by introducing *TrackIt* — a rule-based system that leverages both keystroke analytics and self-reported reflections to more robustly detect and understand student struggles.

Chapter 4 - *TrackIt*: A Rule-Based System for Detecting Struggling Programmers

4.1 Introduction

This chapter proposes a rule-based system, *TrackIt*, that integrates keystroke data with self-reported student survey responses to identify struggling students. Keystroke data has emerged as a powerful approach for capturing coding behaviors and identifying students who have difficulties, based on 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 the student struggles or confidence. The survey serves as a baseline to assess the key dimensions of student experience, including the time taken to complete a programming assignment, self-reported struggles, specific areas of difficulty and perceived confidence while writing their codes. By combining these reports with keystroke-based indicators, the tool aims to automatically identify struggling students and group them into meaningful categories.

This chapter therefore builds upon the interactive code visualization framework introduced in Chapter 3 by focusing on struggle detection, thereby providing a more comprehensive methodology for designing and implementing a struggle detection system. It explores the interplay between keystroke dynamics and self-reported student experiences, the development of rule-based criteria for struggle classification and the validation of the system's effectiveness in correctly identifying struggling students.

This approach will equip the instructors with a data-driven tool for identifying and supporting struggling students as the system proposes a holistic approach to improving the learning outcomes in programming courses. This chapter therefore aims to provide answers to the following research questions.

1. What patterns in keystroke behavior are the most significant indicators of struggling students?
2. How does self-reported student feedback compare to keystroke-based indicators in identifying struggling students?
3. What are the major areas/concepts in fundamental or introductory level programming courses that are most challenging to students?

4.2 Rationale for Adopting Rule-Based System Over Machine Learning

Detecting struggling programmers using keystroke data requires a carefully selected methodology that aligns with the nature of the available data. Although machine learning and deep learning techniques have been widely applied in educational data and learning analytics in general, these methods typically rely on large, well-structured and labelled datasets. In this work however, the dataset is relatively small, making a rule-based system a preferred choice. Unlike the machine learning models which depend on large amount of training data to identify patterns, the rule-based approach can effectively work with limited data by leveraging domain expert knowledge and related work to detect struggling behaviors in programming assignments.

An important advantage of using the rule-based approach is its adaptability across different coding platforms. Keystroke behaviors such as frequent deletions, long pauses and exhibition of repeated errors remain consistent indicators of struggle, irrespective of the Integrated Development Environment (IDE) being used. If the keystroke data is available in a

compatible format, the same rule-based logic can be applied across multiple platforms without any major modifications. This ensures that the system can be expanded to other widely used platforms such as Visual Studio Code, PyCharm, Jupyter Notebook etc.

4.3 Methodology

This section provides details on the research design, data collection methods, implementation and analysis techniques employed in the design and evaluation of *TrackIt*. To establish the reliability of the tool, its struggle detection and classification is compared with student self-reported survey responses to understand its performance and limitations.

4.3.1 Research Design

This study uses a mixed-method approach that integrates quantitative analysis of the keystroke data and both quantitative and qualitative analysis of the survey responses. The goal of combining these two data sources is to provide a more holistic understanding of student struggles and 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 pre-defined rules to classify students into struggle levels by using a weighted scoring system.

4.3.2 Participants and Course/Assignment Context

The study used an introductory programming course designed for students with little to no prior programming experience in python, with about 40 students enrolled in the course. The course focused on fundamental programming concepts, with the programming assignments consisting of structured Lab-based activities where students wrote, tested and debugged Python programs in Codio interactive coding environment, as well as and more detailed homework assignments (Tables 4.1 and 4.2). Each Lab assignments consist of one or more exercises.

Table 4.1. Lab Assignments and Coverage

Lab Assignment	**Description/Coverage**
Lab 1: Basic Python	Variables, printing and code tracing
Lab 2: Numbers and Math	Data types and math operators
Lab 3: Strings and Input	String operators, formatting, numerical and string input, complex statements
Lab 4: Booleans and Conditionals	Boolean operators, Boolean comparators, if, if-else, general branching
Lab 5: Nested Conditionals	Chaining and nested conditionals, block and scope
Lab 6: Loops	For and while loops
Lab 7: Nested Loops	Nested For and While loops
Lab 8: Functions	Basics of functions, parameters and arguments, return statement
Lab 9: Lists	Loops with list, lists slicing and strings as lists
Lab 10: Dictionaries	Loops with dictionaries, functions with dictionaries

Table 4.2. Homework Assignments and Coverage

Homework Assignment	**Description/Coverage**
Homework 1: A simple compound interest calculator	Variables, Data Types, User Input, Math Operators, String formatting and Output
Homework 2: A simple cash register application to manage transactions, calculate tax and make changes	Boolean Logic, Conditional Statements and Mathematical Operators
Homework 3: Building a Finite State Machine that simulates a real-life vending machine	Boolean Logic, Conditional Statements, Loops, Nested Loops and User Input
Homework 4: Decision Tree – Identifying creatures in a garden	Nested Conditional Statements, Handling User Input, Functions, Parameters, Argument and Return
Homework 5: Implementing a Shift Cipher to encode and decode texts/messages	Lists, Nested Loops, User Input, Functions, Parameters, Argument and Return

4.3.3 Data Collection and Description

4.3.3.1 Baseline Survey Data for Self-Reported Struggles

To establish a ground truth for struggle classification, a structured survey was administered immediately after the students completed their assignments. The survey aimed to capture the students' experiences of difficulty, struggle, confidence and specific areas of struggle, while also accounting for the time taken to complete their assignments. The survey included both quantitative and qualitative questions to assess multiple dimensions of struggle (Table 4.2).

Table 4.3 Survey Structure and Questions

Survey Section	Description	Scale/Format
Name	Identifies the student (anonymized for analysis)	Open-ended text entry
Perceived Difficulty Rating	How difficult was the assignment?	1(Very Easy) – 5 (Very Difficult)
Completion Time	Time taken to complete the assignment	Numeric Entry (in minutes)
Struggle Rating	How much did you struggle?	1(Not at all) – 5 (Struggled a lot
Most Struggled Area	Which part of the assignment was most difficult?	Open-ended text entry
Confidence Level	How confident are you in completing the assignment?	1(Not confident at all) – 5 (Very confident)

Each survey question was designed to capture different aspects of student struggle. The *perceived difficulty rating* serves as an indicator of the perceived cognitive load associated with the programming task and aims to provide insights into whether the students experienced the assignment as manageable or overwhelming. The *struggle rating* captures the actual engagement and problem-solving efforts of the students and reports whether the students find themselves stuck, making repeated errors or having to consult external resources and assistance. The *completion time* measures the time commitment to solving the assignment and indicates whether the students complete the assignment quickly with minimal struggle – suggesting understanding and efficiency. The *confidence level* further provides insight into students' self-efficacy.

Comparing this with the struggle rating and completion times determines whether struggling students tend to lack confidence in their work.

In addition to these quantitative measures, the survey included open-ended responses to gather qualitative insights on the specific areas of the assignments that students found most challenging. This allows for deeper thematic understanding and analysis of common programming obstacles and is essential in identifying whether students struggle more with syntax, logic, conceptual understanding or have trouble with the programming platforms. By incorporating these diverse parameters into the survey, the study constructs a comprehensive baseline for student struggles, which is used to validate the performance of *TrackIt,* ensuring that the tool correctly identifies students who report high struggle levels

4.3.3.2 Keystroke Data

Keystroke data was collected from Codio, the learning platform where students write their codes. Description of the attributes of the keystroke data is as discussed in Chapter 3, Section 2.4.1. 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.

4.3.4 Data Processing

4.3.4.1 Preprocessing of Keystroke Data

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.

4.3.5 Features Extraction for Keystroke Data Analysis

In order 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. This enables the tool to infer how frequently a student is writing codes, how often they pause, how frequently they correct errors and ultimately, whether they exhibit any signs of struggle or complete disengagement from writing the programs. The features extracted are grouped into time-based features, pause classification features and typing and editing behavior features.

4.3.5.1 Time-Based Features

- **Total Duration:** This represents the entire period a student spends writing their code, from the first recorded keystroke to the last. It includes active typing, pauses, debugging efforts, deletion actions and all keystroke activities related to the completion of the programming task.

$$Total\ Duration = T_{end} - T_{start}$$ .....**Equation 4.1.** Total Duration

Where:

T_{start} is the timestamp of the first recorded keystroke

T_{end} is the timestamp of the last recorded keystroke

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. In the same way, a long duration could suggest that the student spent time hesitating, debugging difficulties or excessive trial-and-error attempts.

- **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.

$$Total\ Pauses = \sum(T_i - T_{i-1}) \quad \text{.....} \textbf{Equation 4.2.}\ \text{Total Pauses}$$

Where:

$T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

Here, only pauses greater than 2 seconds are considered, for the purpose of classifying pauses. A high total pause time relative to the duration of completing the task could suggest that the student is spending significant time deliberating or struggling.

- **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.

$$\boldsymbol{Idle\ Time\ Ratio\ (ITR)} = \frac{\boldsymbol{Total\ Pause\ Time}}{\boldsymbol{Total\ Coding\ Session\ Duration}} \ldots \textbf{Equation 4.3.}\ \text{Idle Time Ratio}$$

Where:

$Total\ Pause\ Time$ is the cumulative time during which no keystrokes were recorded as defined in *Equation 4.2*

$Total\ Coding\ Session\ Duration$ represents the total time from the first keystroke to the last keystroke in the session as defined in *Equation 4.1*

4.3.5.2 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 [109], which were based on studies by [110], [111], [112] 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.

  $\boldsymbol{Micro\ Pauses} = \sum(\boldsymbol{T_i - T_{i-1}}),\ \boldsymbol{2 \leq (T_i - T_{i-1})} < 15$…**Equation 4.4.** Micro Pauses

  Where:

  $T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

  $Only\ pauses\ between\ 2\ and\ 15\ seconds\ are\ counted$

  While a moderate number of micro pauses are expected in fluent programming, an excessive number of micro pauses may indicate lack of familiarity with syntax or uncertainty in writing the code.

- **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.

  $\boldsymbol{Mild\ Pauses} = \sum(\boldsymbol{T_i - T_{i-1}}),\ \boldsymbol{15 \leq (T_i - T_{i-1})} < 120$… **Equation 4.5.** Mild Pauses

  Where:

  $T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

  $Only\ pauses\ between\ 15\ and\ 120\ seconds\ are\ counted$

A few mild pauses are normal especially when debugging. However, a high frequency of mild pauses may indicate lack of understanding or difficulty recalling programming concepts.

- **Short Pauses:** Short pauses refer to gaps between 2 and 3 minutes, often resulting from students strategizing or rethinking their approach to the problem.

$$\boldsymbol{Short\ Pauses} = \sum(\boldsymbol{T_i - T_{i-1}}),\ \boldsymbol{120 \leq (T_i - T_{i-1})} < 180 \ldots$$ **Equation 4.6.** Short Pauses

Where:

$T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

$$Only\ pauses\ between\ 120\ and\ 180\ seconds\ are\ counted$$

A few short pauses may be beneficial as they indicate thoughtful coding. On the other hand, too many short pauses may suggest difficulty in structuring the programming solution.

- **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 (for example, searching through Stack Overflow, course materials, referring to previous lab solutions etc).

$$\boldsymbol{Mid\ Pauses} = \sum(\boldsymbol{T_i - T_{i-1}}),\ \boldsymbol{180 \leq (T_i - T_{i-1})} < 600 \ldots$$ **Equation 4.7.** Mid Pauses

Where:

$T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

$$Only\ pauses\ between\ 180\ and\ 600\ seconds\ are\ counted$$

While a couple of mid pauses may indicate debugging efforts, frequent mid pauses are strong indicators of struggle.

- **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.

  $$\boldsymbol{Long\ Pauses} = \sum(\boldsymbol{T_i - T_{i-1}}),\ (\boldsymbol{T_i - T_{i-1}}) \geq 600 \ldots$$ **Equation 4.8.** Long Pauses

  Where:

  $T_i\ and\ T_{i-1}$ are consecutive keystroke timestamps

  $Only\ pauses\ that\ last\ at\ least\ 600\ seconds\ are\ counted$

  A single long pause is concerning, especially if it is not followed by any significant code edits. Multiple long pauses indicate severe struggle or disengagement.

4.3.5.3 Typing and Editing Behavior Features

Typing and editing behavior features focus on how students interact with the code editor, including how frequently they type, how often they delete characters and how efficiently they make corrections.

- **Total Insertions:** This indicates the number of characters added to the code during the session and includes both the new code and rewritten code after deletions, not considering characters that come from instructor-provided starter codes.

  $$\boldsymbol{Total\ Insertions} = \sum \boldsymbol{C_{insert}} \ \ldots$$ **Equation 4.9.** Total Insertions

  Where:

  C_{insert} is the number of characters inserted at each keystroke

  While writing a short body of functional code might indicate a good and concise programming habit, a dangerously low number of insertions suggests lack of engagement, hesitation or attempting to copy and paste solutions from external sources.

Also, a high number of insertions indicate active coding but nevertheless, excessive insertions with frequent deletions may suggest trial-and-error behavior.

- **Total Deletions:** Total deletions count the number of characters removed from the code.

  $\boldsymbol{Total\ Deletions} = \sum \boldsymbol{C_{delete}}$ … **Equation 4.10.** Total Deletions

  Where:

  C_{delete} is the number of characters deleted at each keystroke

  A low deletion count may indicate fluency in coding while high deletion count might suggest frequent trial-and-error attempts or uncertainty in coding decision. It could also be an indication of frequent mistakes that are being corrected.

- **Deletion Ratio:** The deletion ratio compares the number of deletions to insertions, measuring how often students remove and rewrite their code. It measures the proportion of the inserted characters that were deleted.

  $\boldsymbol{Deletion\ Ratio\ (DR)} = \frac{\boldsymbol{Total\ Deletions}}{\boldsymbol{Total\ Insertions+1}}$ … **Equation 4.11.** Deletion Ratio

  Where:

  $Total\ Deletions$ is the total number of characters removed from the code (defined in 10)

  $Total\ Insertions$ represents the total number of characters added to the code (defined in 9)

  $The\ addition\ of\ 1\ to\ the\ Total\ Insertions\ is\ to\ avoid\ divition\ by\ zero$

- **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.

  $\boldsymbol{Insert-Delete\ Ratio\ (IDR)} = \frac{\boldsymbol{Total\ Insertions}}{\boldsymbol{Total\ Deletions+1}}$ ...**Equation 4.12.** Insert-Delete Ratio

Where:

$Total\ Deletions$ is the total number of characters removed from the code (defined in 10)

$Total\ Insertions$ represents the total number of characters added to the code (defined in 9)

The addition of 1 to the total deletions is to avoid division by zero when no deletions are recorded.

- **Correction Frequency/Speed:** This measures how frequently a student corrects errors, which could indicate debugging behavior or difficulty in making progress in writing the code.

$$\boldsymbol{Correction\ Speed} = \frac{\boldsymbol{Total\ Deletions}}{\boldsymbol{Total\ Coding\ Session\ Duration(minutes)}} \ldots \textbf{Equation 4.13.}$$ Correction Speed

Where:

$Total\ Deletions$ is the total number of characters removed from the code (defined in *Equation 4.10*)

$Total\ Coding\ Session\ Duration$ represents the total time from the first keystroke to the last keystroke in the session as defined in *Equation 4.1*

Since the correction frequency measures how frequent the student makes corrections per minute, it is expected to be low – indicating careful methodical coding. A high correction frequency could be indicative of frequent mistakes and struggle.

- **Typing Speed:** Typing speed measures how quickly a student is entering characters into the code editor.

$$Typing\ \boldsymbol{Speed} = \frac{\boldsymbol{Total\ Insertions}}{\boldsymbol{Total\ Coding\ Session\ Duration(minutes)}} \ldots \textbf{Equation 4.14.}$$ Typing Speed

Where:

$Total\ Insertions$ is the total number of characters removed from the code (defined in *Equation 4.9*)

$Total\ Coding\ Session\ Duration$ represents the total time from the first keystroke to the last keystroke in the session as defined in *Equation 4.1*

Typing speed itself is not a direct measure or indicator of struggle, but extreme variations in typing speed can reveal some behavioral patterns. For example, as sudden spike in the number of characters inserted could indicate a copy-paste event and the student did not manually type in the code.

- **Active Typing Segment:** An active typing segment represents the average number of keystrokes a student makes before pausing.

$$\boldsymbol{Active\ Typing\ Segment} = \frac{\sum \boldsymbol{C_{insert}}}{\sum \boldsymbol{P_i}}$$ … **Equation 4.15.** Active Typing Segment

Where:

$\sum C_{insert}$ is the total number of characters inserted (defined in *Equation 4.9*)

$\sum P_i$ represents a pause event

Short segments of active typing before a pause could indicate frequent hesitation and struggle whereas long segments of active typing could suggest fluency and confidence.

4.3.6 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. 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. Below are the rules set to govern the assignment of struggle scores:

4.3.6.1 Idle Time Ratio Contribution

The *Idle Time Ratio* contributes to the struggle level classification, as it measures the proportion of total time that the student spent pausing rather than actively typing. Table 4.3 shows the threshold values for the *Idle Time Ratio*:

Table 4.4. Idle Time Thresholds

Idle Time Ratio	Struggle Score	Struggle Level
< 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

An *Idle Time Ratio* less than 0.3 indicates that the student spends at least 70% of the session actively coding, indicating a period of rare hesitation, hence no struggle points are added. An *Idle Time Ratio* between 0.3 and 0.5 indicates that the student spends between 30% and 50% of the session pausing, which is suggestive of occasional hesitation but not enough to indicate major struggle hence a slight struggle score of +2 is assigned. An *Idle Time Ratio* between 0.5 and 0.7 means that the student spends more than half of the session pausing, indicating a more frequent hesitation possibly due to lack of confidence, debugging difficulties or challenges in planning the code structure. A struggle score of +4 is assigned. If the *Idle Time Ratio* is at least 0.7, the student spends more time (at least 70%) pausing than coding. This suggests that the student is struggling significantly or disengaged. A moderate to high struggle score of +6 is assigned.

4.3.6.2 Pause-Based Struggle Contribution

A student with frequent long pauses is likely to struggle significantly, whereas mid-length pauses may indicate uncertainty but not complete disengagement from writing the program.

Table 4.5. Pause-Based Struggle Thresholds

Pause Type	Struggle Score	Struggle Level
Micro Pauses (2 – 15 seconds)	0	No impact
Mild Pauses (15 – 120 seconds)	0	No impact
Short Pauses (120 – 180 seconds)	+3 if > 5 occurrences +6 if > 10 occurrences	Slight Struggle Moderate Struggle
Mid Pauses (1800 – 600 seconds)	+3 if > 5 occurrences	Moderate Struggle
Long Pauses (> 10 minutes)	+6 if > 2 occurrences	High Struggle and Complete disengagement from coding

4.3.6.3 Insert-Delete Ratio Contribution

The relationship between inserted characters and those deleted could determine the struggling status of the students.

Table 4.6. Insert-Delete Ratio Thresholds

Insert – Delete Ratio	Struggle Score	Struggle Level
> 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

If the *Insert – Delete Ratio* > 3.0, the student types three times more than they delete, meaning they are confident in their code structure. This suggests confidence and minimal struggle; hence zero points are added. If the *Insert – Delete Ratio* is between 1.5 – 3.0, the student deletes a moderate portion of their code but still types significantly more than they delete. This suggests minor hesitations but does not strongly indicate struggle; hence a struggle score of +2 is assigned. An *Insert – Delete Ratio* between 1.0 and 1.5 means that the student is deleting nearly as much as they are typing, suggesting continuous correction, revisions or trial-and-error coding. This is a strong indicator of struggle and therefore a struggle score of +4 is assigned. If an *Insert – Delete Ratio* is less than 1.0, the student deletes more than they insert, meaning they are constantly deleting and making corrections – a very strong indicator of severe struggle leading to an assignment of +6 struggle score.

4.3.6.4 Deletion Ratio Contribution

The *Deletion Ratio* is another measure of editing struggle – measuring how much what was typed was later removed.

Table 4.7. Deletion Ratio Thresholds

Deletion Ratio	Struggle Score	Interpretation
< 0.3	0	Few corrections
0.3 – 0.7	+3	Slight struggle
> 0.7	+6	Excessive backspacing (Moderate/High struggle)

A *Deletion Ratio* less than 0.3 indicates that the student only deletes a small portion of their code and most of the codes remain intact, suggesting confidence and fluency – leading to a zero-struggle point. If *Deletion Ratio* is between 0.3 and 0.7, the student is deleting a moderate portion of their code, which may indicate editing and debugging but not necessarily a strong struggle, hence a struggle score of +3 is assigned. A *Deletion Ratio* that is greater than 0.7 indicates that the student deletes more than 70% of their inserted codes, meaning they are erasing continuously. This is a strong indicator of severe struggle leading to a struggle score of +6.

4.3.6.5 Correction Frequency Contribution

How frequently does a student tend to delete the codes they already typed within one minute? This could define struggle levels, with the following threshold values:

Table 4.8. Correction Frequency Thresholds

Correction Frequency	Struggle Score	Struggle Level
< 10 deletions per minute	0	Confident typing
10 – 30 deletions per minute	+2	Some struggle
> 30 deletions per minute	+5	Frequent mistakes (Moderate to High struggle)

If the *Correction Frequency* is less than 10 deletions per minute, the student makes very few mistakes, leading to no struggle points. A *Correction Frequency* between 10 and 30 deletions per minute means that the student is making corrections at a reasonable rate, suggesting some debugging activity but no extreme struggle, leading to the assignment of +2 struggle score. A *Correction Frequency* that is greater than 30 deletions per minute indicates that the student is frequently deleting their code, suggesting that they struggle with code structures or uncertainty. A struggle score of +5 is therefore assigned.

4.3.6.6 Active Typing Segment Contribution

The average length of keystrokes entered before a significant pause also contribute to identifying struggling students.

Table 4.9. Active Typing Segment Thresholds

Average length Active Typing Segment	Struggle Score	Struggle Level
> 10 keystrokes	0	Confident coding with no struggle
5 – 10 keystrokes	+3	Some Struggle
< 5	+6	Frequent hesitations, leading to Moderate to High Struggle

If the student has an average code segment greater than 10 keystrokes before a major pause, the student tends to type in long uninterrupted bursts therefore no struggle points are added. An average code segment between 5 and 10 keystrokes before a major pause indicates that the student is pausing intermittently, suggesting moderate hesitation, leading to a struggle score of +3. A student is pausing extremely frequently if the average keystroke segment entered before a significant pause is less than 5 keystrokes. This is suggestive of severe hesitation and difficulty, leading to an assignment of +6 struggle score.

4.3.6.7 Copy-Paste Struggle Contribution

In programming assignments, students tend to copy and paste codes from various sources. While minor pasting can be part of normal coding behavior (especially if copied within already written codes), unusually large pasting from external sources may indicate struggle, lack of understanding or an external dependency.

Table 4.10. Copy-Paste Struggle Thresholds

Paste Type	Struggle Score	Struggle Level
Any Large Paste (50 + characters	+2	Slight Struggle
Any Very Large Paste (100 + characters)	+4	Moderate Struggle
Frequent Large Pasting (50+ characters, greater than 3 times)	+6	High Struggle

If a student pastes a moderately large chunk of code (more than 50 characters), it might indicate slight struggle, leading to a struggle point of +2. A very large paste of code from an external source greater than 100 characters may suggest copying an entire function or multiple lines from an external source, leading to a moderate struggle score of +4. Frequent large pasting of codes greater than 50 characters more than 3 times signifies a high dependency on external help, which might indicate a high-level struggle in writing the code, hence a struggle score of +6 is assigned.

4.3.6.8 Struggle Classification Based on Struggle Score

Once the struggle score is computed, students are classified into one of five categories. Table 4.10 shows the range of values of the struggle score and the struggle categories.

Table 4.11. Struggle Classification

Struggle Score	Struggle Category
0 – 4	Not at all
5 – 8	Slight Struggle
9 – 12	Some Struggle
13 – 16	Moderate Struggle
> 16	Struggled a lot

4.4 Analysis and Results

4.4.1 General Survey Data Analysis

4.4.1.1 Descriptive Statistics

Tables 4.11 and 4.12 show the summary statistics of the perceived difficulty, struggle and confidence levels 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 4.12. Summary Statistics for Lab Assignments

	Difficulty Level	Completion Time (Minutes)	Struggle Rating	Confidence Level
count	292	279	292	292
mean	3.34	129.48	3.35	2.74
std	0.98	110.95	1.14	1.10
min	1	10	1	1
25%	3	60	3	2
50%	3	100	3	3
75%	4	180	4	3
max	5	900	5	5

Table 4.13. Summary Statistics for Homework Assignments

	Difficulty Level	Completion Time (Minutes)	Struggle Rating	Confidence Level
count	129	129	129	129
mean	3.62	152.09	3.60	2.74
std	0.97	121.69	1.17	1.20
min	1	30	1	1
25%	3	70	3	2
50%	4	120	4	3
75%	4	180	5	4
max	5	840	5	5

4.4.1.2 Correlations Across Quantitative Measures

The correlation heatmaps (Figures 4.1 and 4.2) 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.

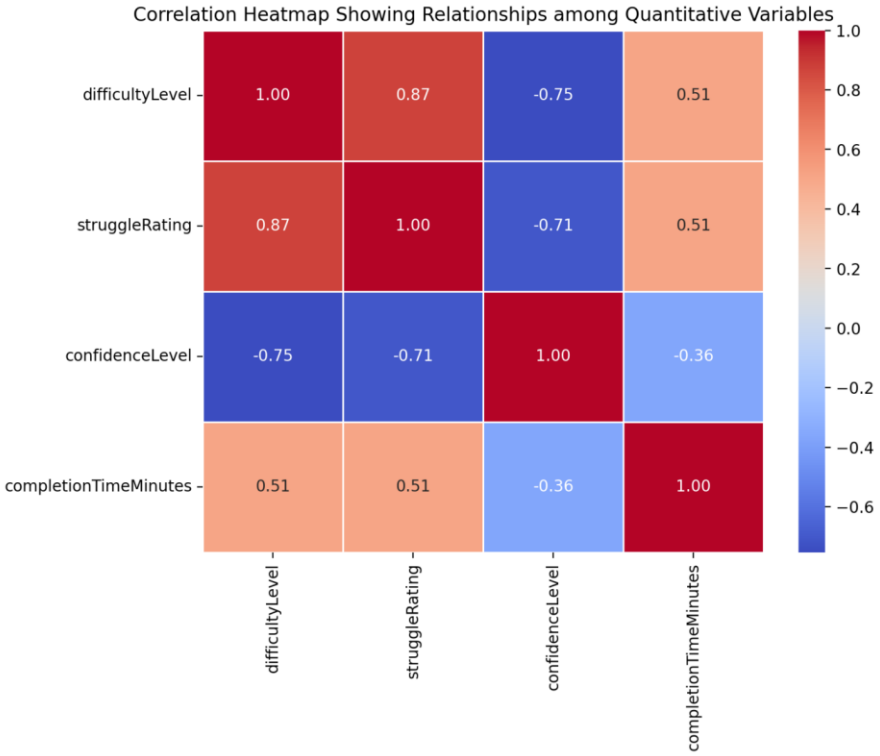


Figure 4.1. Correlation Heatmap: Lab Assignment

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.

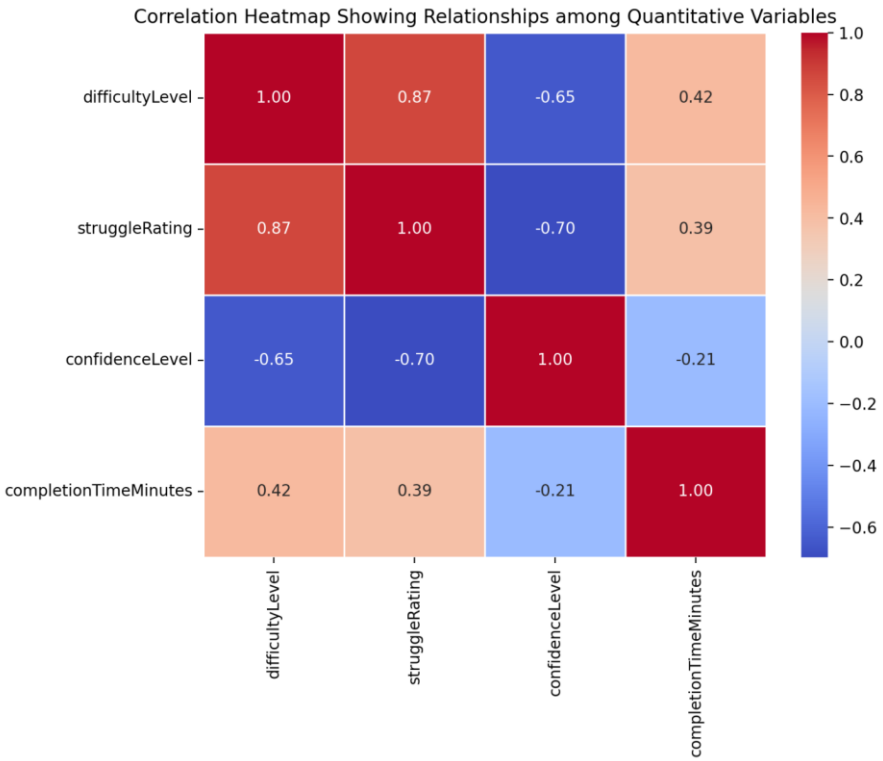


Figure 4.2. Correlation Heatmap: Homework Assignments

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 the homework assignments late and had 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.

4.4.1.3 Identifying the Most Challenging Assignments

The lab assignment analysis chart (Figure 4.3), reveals that Lab 5, covering Nested Conditionals, is the most challenging of all lab assignments.

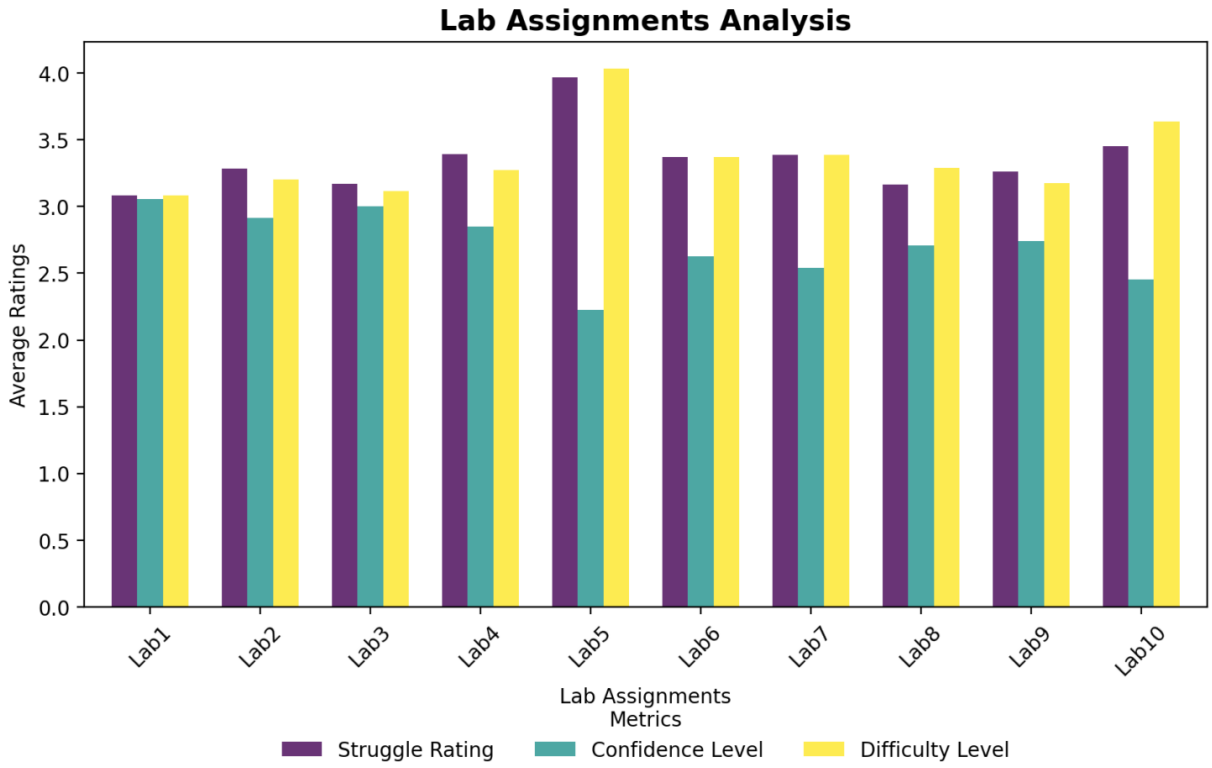


Figure 4.3. Lab Assignments Chart

This is evidenced by the highest struggle rating and difficulty level, while the confidence level is significantly lower than in other lab tasks. Lab 7 and Lab 10 also stand out as difficult assignments, as both exhibit relatively high struggle ratings and difficulty levels while maintaining lower confidence levels compared to other labs. On the other hand, Lab 1, Lab 2 and

Lab 3 appear to be among the least challenging, with lower struggle ratings and moderate difficulty levels, while confidence remains higher in the more difficult labs.

The homework assignments chart (Figure 4.4) reveals that Homework 3, which required building a Finite State Machine that simulates a real-life vending machine by demonstrating familiarity with Boolean Logic, Conditional Statements, Loops, Nested Loops and User Input, stands out as the most challenging assignment.

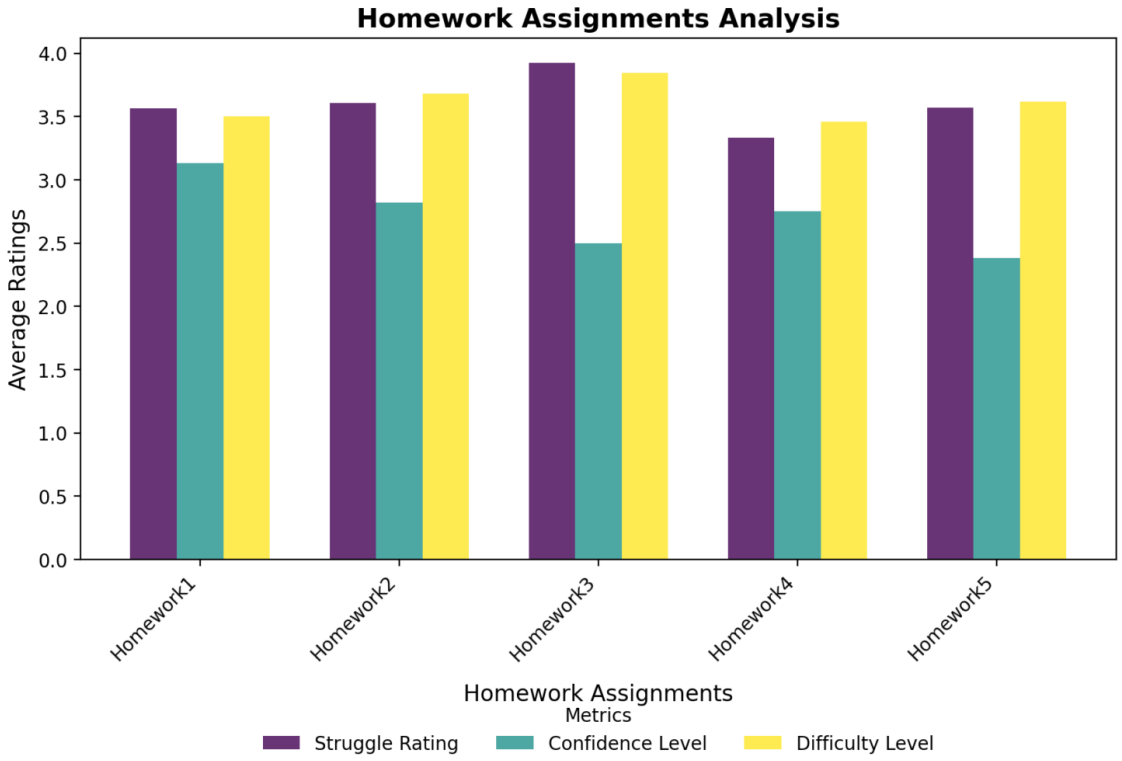


Figure 4.4. Homework Assignments Chart

This is indicative of the highest average struggle rating among all homework tasks, coupled with a relatively lower confidence level. Homework 5 also presents some challenges, with a relatively high struggle rating and difficulty level while confidence level remains lower compared to other homework assignments. Although the struggle ratings for Homework 1 and Homework 2 appear moderately high, the confidence levels are relatively higher.

4.4.2 Keystroke Data Analysis and *TrackIt* Features

4.4.2.1 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 keys 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, TrackIt 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 copy-paste detection discussed in more detail in the following section. This is important as the differentiation between students' keystroke and pasted code from external sources is crucial in flagging behaviors associated with struggle.

4.4.2.2 Copy-Past Detection Functionality

One of the standout features 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 4.5) and another student who pasted texts from reddit – an online interactive platform (Figure 4.6). The keystroke tracked and revealed a sudden influx of 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.

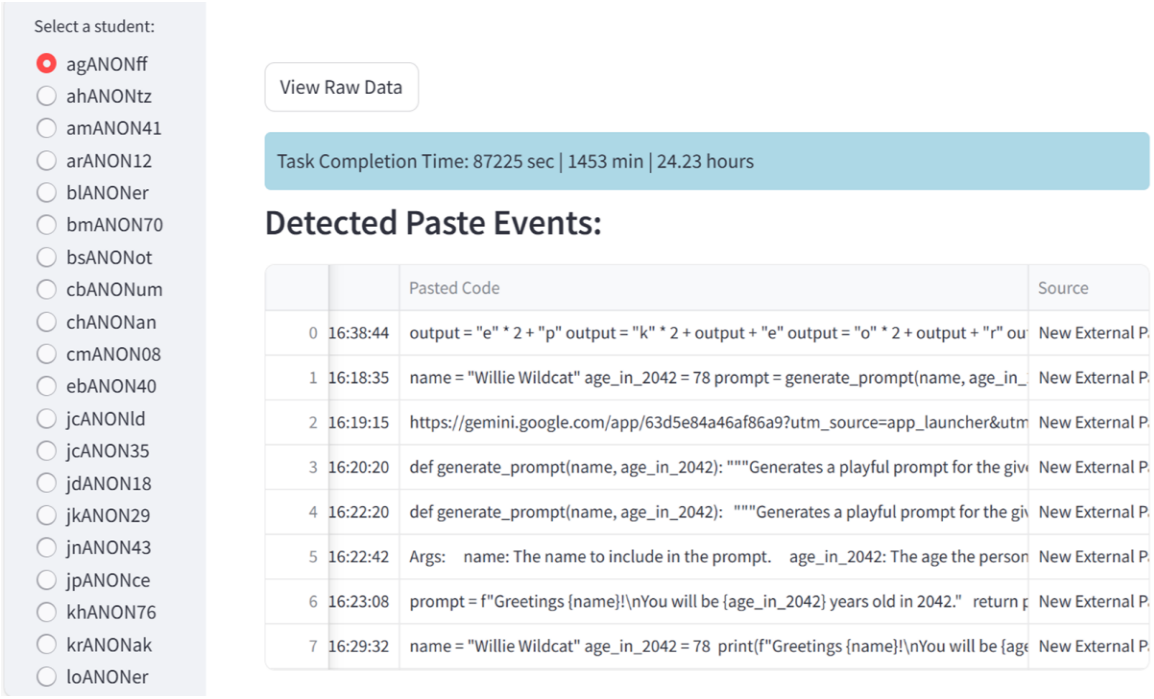


Figure 4.5. Copy-Paste Detection - I

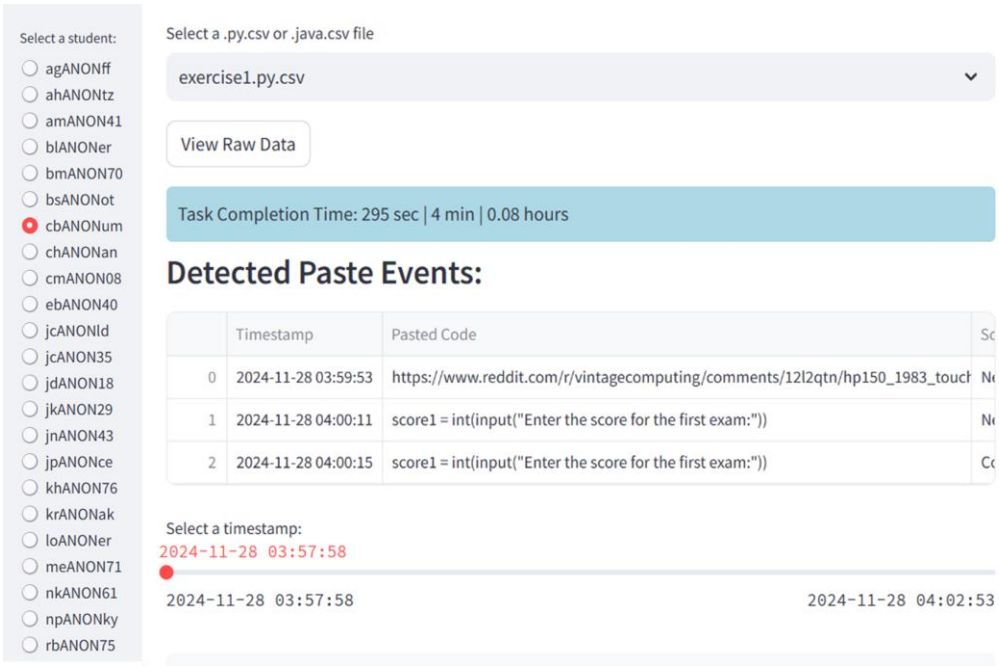


Figure 4.6. Copy-Paste Detection II

4.4.2.3 Identifying Most Difficult Assignments (Versus Survey Report)

A striking observation from the comparison of struggle ratings (Figures 4.7 and 4.8) between *TrackIt's* automated analysis and student self-reported surveys is the strong alignment in identifying the most difficult assignments. 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. This is particularly fascinating because it aligns with the results from section 4.4.1.3 (Figures 4.3 and 4.4) where confidence level, struggle rating and difficulty rating combined to identify the most challenging assignments. 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.

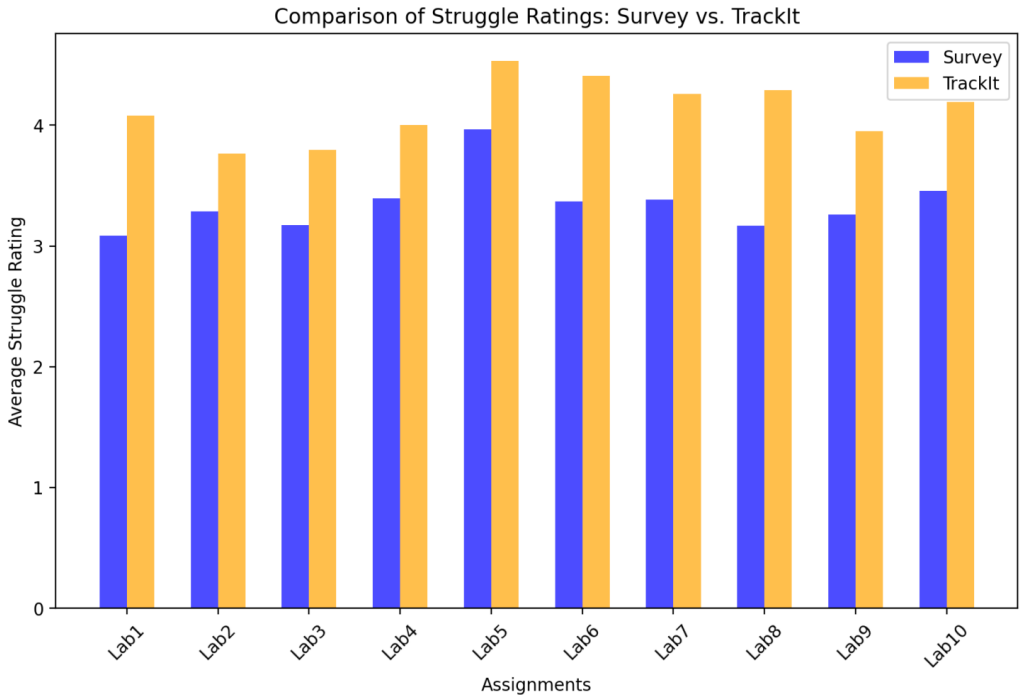


Figure 4.7. Identifying the Most Challenging Lab

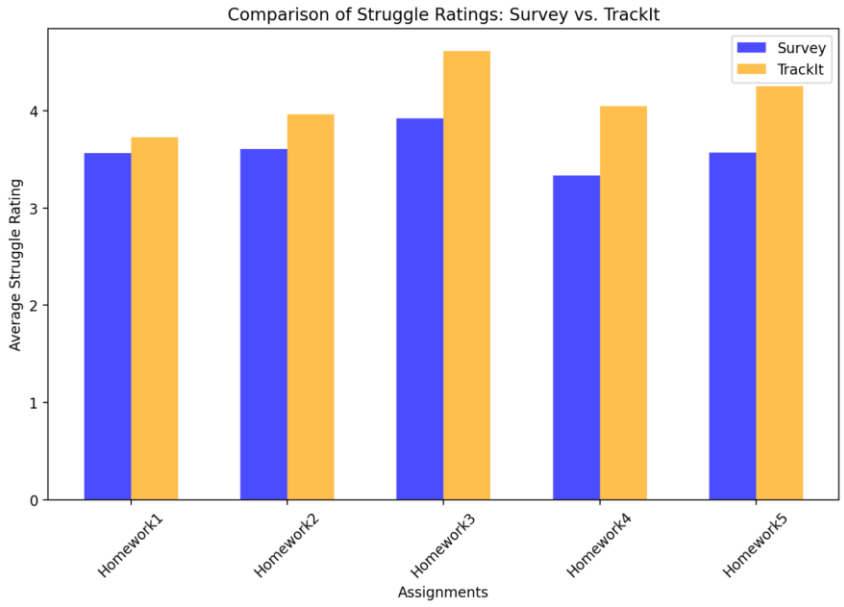


Figure 4.8. Identifying the Most Challenging Homework

4.5 Performance Evaluation of *TrackIt*

4.5.1 Collapsing Struggle Ratings

The five-point struggle rating scale used in this study was collapsed into three broader categories – Low Struggle (1), Medium Struggle (2 & 3) and High Struggle (4 & 5). This is to enhance interpretability and maintain the integrity of the result. According to Allen and Seaman [113], while collecting data on a more granular Likert-scale can be valuable, collapsing the categories during analysis is acceptable when it aids clearer reporting, provided the original scale has sufficient range.

Furthermore, traditional agreement indices such as Cohen's Kappa [114] are known to penalize even small discrepancies in ordinal ratings. A marginal difference between adjacent ratings such as 3 versus 4, is treated as a disagreement. Although weighted variants of Kappa such as linear and quadratic weighting, attempt to account for these marginal differences, they still remain sensitive to rating distribution and category boundaries. This has been studied in inter-rater reliability research, especially in the work of Marasin et al. [115], who highlighted that small rating differences can artificially deflate agreement statistics. Initial analysis for this study suffered from this artificial deflation effect due to *Track It*'s slight overestimation of struggle. Collapsing the scale mitigated this problem.

4.5.2 Agreement Between Student Reported Survey and TrackIt

The effectiveness of *TrackIt* in identifying students' struggle was examined by comparing its automated outputs against students' self-reported ratings collected through post-assignment surveys. This comparison was conducted across both Lab and Homework assignments, and the results measured using Cohen's Kappa (unweighted and weighted) as well

as raw percentage agreement. These metrics were analyzed before and after collapsing the original 5-point struggle rating scale into broader categories of Low, Medium and High struggle.

Across the lab assignments, the agreement between *TrackIt* and student surveys was generally low when 5-point scale was maintained. As shown in Figure 4.9 and Figure 4.10, before collapsing, unweighted Kappa values hovered near zero, with occasional negative coefficients in the homework assignments. Weighted Kappa values performed slightly better but still suffered from the strictness of Kappa's interpretation that even a small one-point deviation is considered disagreement.

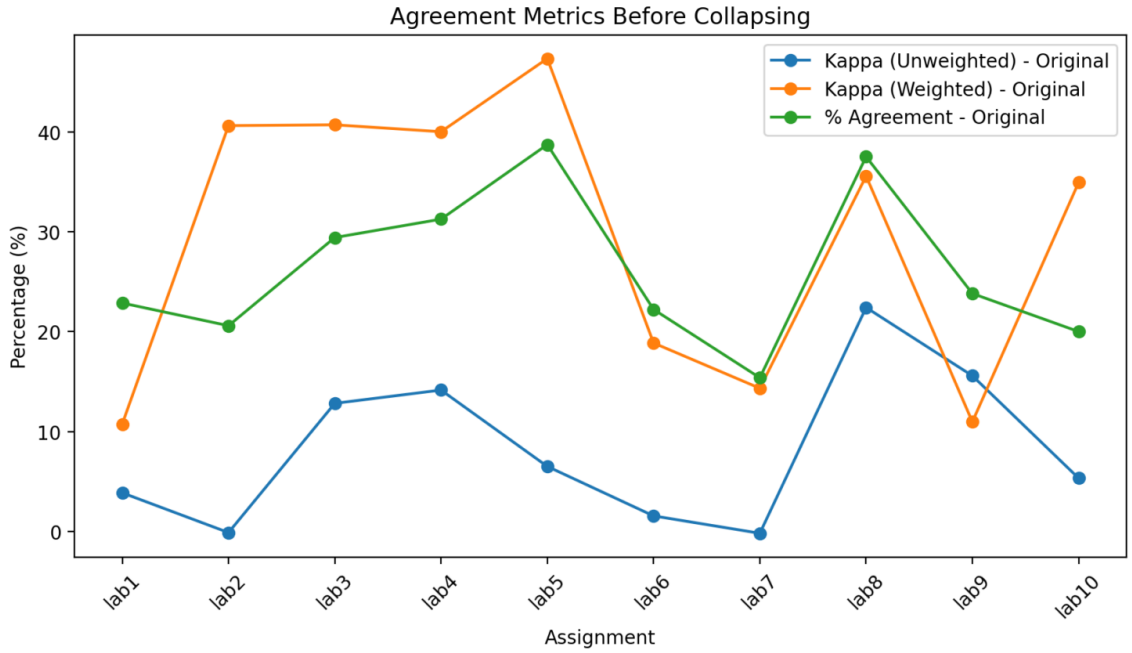


Figure 4.9. Lab Agreement Indices Before Collapsing Struggle Ratings

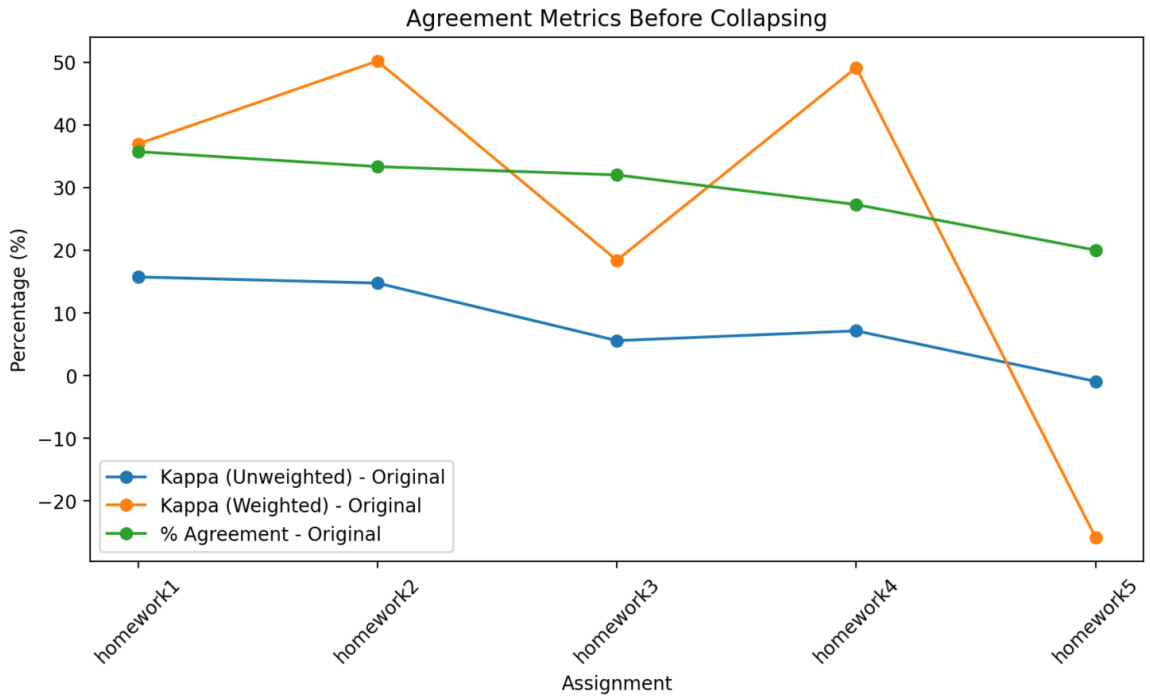


Figure 4.10. Homework Agreement Indices Before Collapsing Struggle Ratings

However, once the ratings were collapsed into three broader struggle levels, the agreement between *TrackIt* and student surveys improved markedly across both types of assignments (Figures 4.11 and 4.12). In the Labs, unweighted Kappa values rose in several instances – notably in Lab 8 and Lab 10 – reaching over 30% and 40% respectively. Weighted Kappa values also increased, reflecting that many of the original disagreements were marginal in nature and became more aligned after scale collapsing. The percentage agreement after collapsing consistently improved across all Labs, with values peaking around 71%. The homework assignments mirrored this trend with the percentage agreements consistently high across all tasks – above 55% in most cases with homework 3 peaking above 60%. Even where unweighted Kappa remained low, the rise in percentage agreement suggests that TrackIt often correctly captured the general direction of perceived struggle even if it didn't match the exact numerical rating.

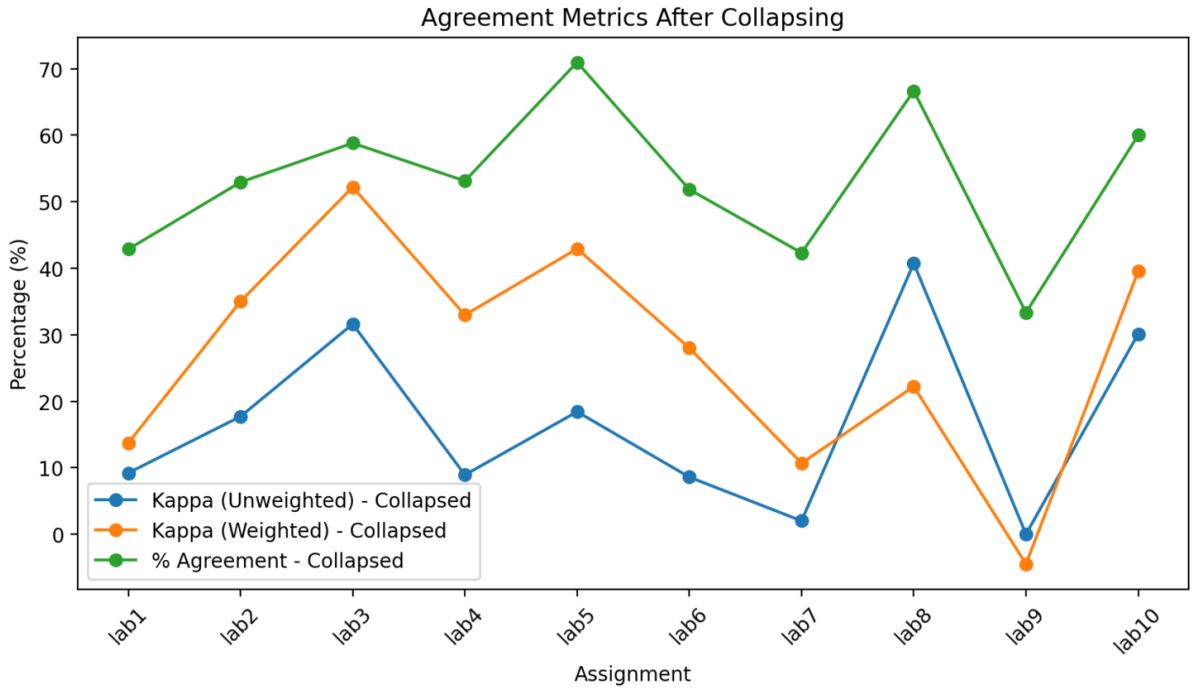


Figure 4.11. Lab Agreement Indices After Collapsing Struggle Ratings

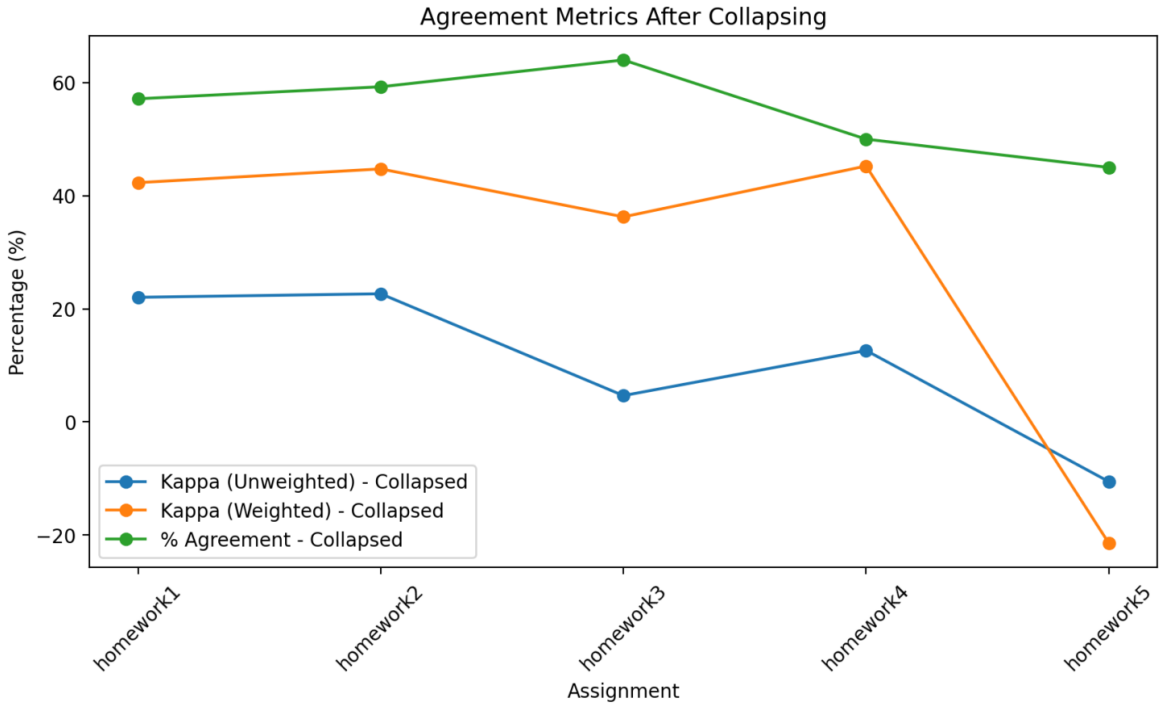


Figure 4.12. Homework Agreement Indices After Collapsing Struggle Ratings

4.6 Addressing the Research Questions

The keystroke data analysis revealed a rich and time-stamped trail of student interactions during coding assignments. Several recurring patterns emerged as strong indicators of student struggle. Most notably were prolonged idle time or inactive periods – revealed by the frequency of pauses, frequent deletions relative to the insertion of keystrokes leading to high deletion ratio or low insert-delete ratio and copy-paste behaviors.

Periods of long pauses without keystrokes were often captured prior to a sudden surge of pasted codes. These pauses likely reflect moments of cognitive struggle or students seeking external help. Copy-paste events were particularly important. When not followed by typing patterns that depict finding solution to the problem, copy-paste events, especially large pastes can be signs of unresolved struggle. Frequent deletions, including removal of large blocks of codes, further emphasized the struggling nature of the students. Unlike simple corrections, large deletions often indicated that the student had a fundamental misunderstanding of the concepts having moments of confusion or frustration.

A core goal of this chapter was to evaluate the alignment of *TrackIt's* behavioral indicators and students' self-reported struggle. Section 4.5.2 provides detailed breakdown of the agreement scores between *TrackIt* and the survey responses from the students.

Analysis of both the keystroke behaviors and the self-reported surveys revealed patterns about which programming concepts were most difficult for the students. 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 4.3 and 4.4, 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. This is also in agreement with *TrackIt's*

capability to identify the most challenging problem. Moreso, we see that the agreement indices between the survey analysis and that of *TrackIt* were highest in these assignments – Lab 5 and Homework 3.

4.7 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 indicator 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.

4.8 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, *TrackIt* 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 codes that are pasted from external ssources and those copied from already created block of codes by the student – repeated or untimely copy-paste behavior often signaled disengagement.

4.9 Limitations

Despite its strengths, *TrackIt* is not without limitations. One major concern is that students often choose to write their code in external environments – probably Visual Studio Code, PyCharm and only paste the completed solution into Codio. This behavior leads *TrackIt* to record a long delay followed by a paste action, which is then interpreted as a copy-paste

triggered by struggle. In such cases, TrackIt may falsely flag confident or well-prepared students as struggling simply because they worked outside the Codio environment.

Another limitation stems from the lack of contextual awareness. *TrackIt* interprets behaviors such as long pauses without understanding the reason behind them. A pause could result from distraction, or external interruptions that don't necessarily emanate from struggle. Because *TrackIt* relies purely on keystroke data, it cannot capture emotional states such as anxiety, frustration or boredom that students may capture in self-reported surveys. The current version of *TrackIt* also processes data only after an assignment has been completed. As such, it cannot provide real time immediate intervention.

Chapter 5 - Building Rural Computer Science Capacity Through Teacher Development

This chapter is adapted from [116] *accepted by ASEE 2025*

5.1 Introduction

In the previous chapters, we explored the development and application of tools designed to detect and support struggling students as they engage in programming tasks. An important insight from that works was the role of timely guidance and knowledgeable mentorship in reversing the route of struggle and frustration, particularly for novice programmers. However, the effectiveness of such interventions ultimately hinges on the presence of well-prepared teachers who can interpret students' needs and leverage support systems appropriately. This realization draws attention to the broader issue of teacher capacity, especially in underserved areas such as rural areas — where access to computing education and qualified mentors are limited. This chapter shifts focus to understanding how teacher training programs can empower rural teachers by enhancing their identity, commitment, confidence, and competence in computer science instruction.

If a significant expansion of CS education is to be equitably achieved; it becomes highly imperative to understand the inequalities in access to computing tools and human resources. Despite a notable increase in enrollment in CS majors since 2009 [117], there is still a marked disparity between rural and urban areas with respect to access to computer science education. A study of 1,537,073 students in Texas showed an under-representation disparity index (which is measured by the quotient of the rate or proportion for the target group over the rate or proportion for the comparison group) of 0.84 for rural areas and an over-representation disparity index of

1.19 for their urban-suburban counterparts [118]. In fact, only 57.5% of public high schools in the United States offer foundational computer science.

Although this is an increase from 53% in the previous year, there are still disparities as rural and smaller schools are less likely to offer computer science foundation [119]. These disparities pose a systematic and national challenge and are created largely by patterns of residential segregation and socioeconomic disadvantage [120]. The integration of computer science into almost every discipline creates lucrative jobs and promising career opportunities. However, the field is still underpopulated and under- represented [121]. Specifically, one of the significant challenges and bottlenecks in the expansion of computer science education is the inaccessibility of highly qualified teachers in rural areas [82]. To help address the CS teacher deficit, Morrissey and Koballa et al. [82] developed a preservice CS certification pathway, a testing option for CS professionals who want to transition from industry into teaching, and a CS endorsement for teachers who are certified in other teaching areas to obtain CS certification but very few of the CS endorsement program providers target rural, high-need school systems. The remoteness of rural areas often leads to unique challenges in expanding access to CS education, including limited technological infrastructure such as the high-speed Internet, fewer opportunities for professional development, and difficulties in recruiting and retaining qualified teachers [122]. Teacher preparation programs in under-resourced institutions of learning in the United States will need to inculcate CS education in order to foster their teacher preparation and professional development efforts [123]. We therefore seek to provide answers to the following research questions, in order to assess the impact of teacher training on educators' professional identity, confidence, competence and commitment, particularly in rural areas and evaluate how

structured training programs influence teachers' motivation and ability to effectively teach computer science:

1. How does participation in the computer science training program influence rural teachers' professional computing identities?
2. How does the training program affect teachers' commitment to teaching computer science?
3. How does participation in the teacher training program affect confidence and teaching competence of rural teachers?

5.2 Background

5.2.1 Computer Science Teacher Training Programs

The lack of access and implementation of computer science education in rural areas has led to increased efforts to broaden the participation in CS education through various teacher training programs. Computer science is seen as one of the most segregated disciplines in the United States, highlighting the necessity of teacher training in developing the knowledge and practices that would broaden participation in computing [124]. Well-designed teacher training programs help build computational thinking skills in teachers. An online STEM-based activity Computer Science Teacher Training (CSTT) was put in place to develop pre-service teachers' Computational Thinking (CT) skills measuring problem decomposition, algorithms, pattern recognition and abstraction, and revealed a 13.58% improvement in the CT test mean scores [125].

Quality teacher training programs targeting K-12 teachers have the tendency of reaching more students [126]. For example, WeTeach CS designed a teacher training program to certify teachers to teach high school CS in Texas, leading to an increase in the number of certified

teachers in rural areas [13]. A comprehensive study on the computer science K-12 outreach teacher training programs of eleven universities demonstrated the effectiveness of these training programs in making computer science accessible to teachers [126].

Professional development has been shown to be among the key factors that contribute to the turnover and retention of STEM teachers in rural areas [127]. Susie and Thomas et al.. [82] developed a project that highlights a mechanism that has the potential of increasing computer science learning opportunities for students in rural, high-need school systems by using well-trained set of teachers. Rural teachers are able to exhibit creative ways of incorporating Computational Thinking into their subjects, following teacher workshops [128]. There are several tools and techniques used in expanding computer science in rural areas. These include Modeling and Simulation [129] as well as robotics, game design and culturally responsive teaching models [130]

5.3 Methodology

We engaged 64 participants, primarily high school teachers and a few others in a comprehensive teacher training program was conducted by the computer science department of a Midwest university in the United States. The program aimed at equipping teachers with the requisite skills needed to deliver effective computer science education particularly focusing on participants from rural and under-represented areas, with the overall goal of integrating computer science into high school curriculum.

5.3.1 Research Design

A mixed-method design was employed, combining both quantitative and qualitative approaches. The rationale behind the use of both approaches is to provide a comprehensive evaluation of the impact of the teacher training on teachers' identities, perceptions and

commitments. The quantitative approach utilizes pre- and post-surveys measured on a 5 -point Likert scale, while the qualitative method integrates teachers' self-reflective journals to gather information regarding teachers' motivation, years of teaching experience, and prior computer science knowledge.

5.3.2 Research Participants

To recruit teachers into the program, we emailed multiple teachers and lists of schools managed by our university and the state Department of Education, with the aim of reaching a wide range of participants. Teachers were invited to complete a brief survey to enter into the program. All teachers who signed up were accepted as long as they were involved in education in some capacity - this included elementary teachers, a librarian, a substitute teacher, an unemployed teacher, and two recent graduates who showed interest in advancing careers in education.

We had a total of 64 participants join our professional development program. These were primarily high school teachers, though we also had two middle school teachers, one junior high teacher, a librarian, and one pre-professional. This mix of participants ensured that exposure to computer science education is extended beyond high school to middle and elementary school levels. The program covered a wide geographic area by involving 34 unique school districts, out of which 27 were classified as rural. This 79.41% representation of rural teachers reflects the program's success in targeting under-served areas. A total of 24 teachers completed both the pre and post surveys, and comprised 13 teachers from town, 8 from rural areas and 3 from suburban locales

5.3.3 Data Collection

The participants were asked to complete pre- and post-training surveys built into the program at the beginning and ending of each course, respectively. The surveys cut across three major areas – teacher and computing identity; rural identity and teacher mindset; and teaching perceptions and computational thinking. The data used for the research covered Spring 2023 to Fall 2023 semesters.

5.3.3.1 Survey Instruments

The survey instruments used for this research were constructed using frameworks in existing literature, all of which were targeted towards evaluating the impact of a teacher training program on teachers' identities, perceptions and sense of belonging amongst other related parameters.

The survey instrument includes items from Computing Identity Framework/Model [131], which reflects interests in computing topics and practices, recognition in computer science with respect to being computer savvy, and performance/competence that highlights how people feel they could perform or understand computing topics and practices.

We adopted a survey construct from Ni et al.'s Teachers' Professional Identity in Computer Science [78], covering self-identification, community/sense of belonging, interest and value of teaching computer science, learning/striving to teach well, confidence in teaching computer science, and commitment to teaching computer science.

We utilized the Rural Identity Scale (RIS) [132] which proved essential for understanding the unique identities of the teachers in rural areas. It measures teachers' perceptions about rural life, activities and behaviors as well as relationships with people in the rural community. The RIS showed an acceptable internal reliability of $\alpha = 0.72-0.83$ which boasts of its effectiveness

in capturing rural identity. The teacher mindset survey, carved out of [133] and [134], was a vital instrument in supplying the valuable insights into diverse aspects of teachers' mindsets. It measures parameters such as concerns on social comparison, self-efficacy, comfort being oneself, measurement of task value, as well as the perceived costs of participating in the training program. Each survey item was measured on a 5-point Likert scale, with 1 being "strongly disagree" to 5 being "strongly agree," which eventually helped in measuring teachers' attitudes in the role of being computer science teachers.

Lastly, the survey incorporated items from Teachers' Self-Efficacy in Computational Thinking (TSECT), which is meant to capture a sense of students' self-efficacy in utilizing programming and computational thinking [17]. All these instruments were put together to provide a comprehensive evaluation of the impact of the teacher training program in expanding Computer Science Education.

5.3.3.2 Teachers' Autobiographies

We analyzed the teachers' autobiographies which provided qualitative data on their personal experiences, challenges and aspirations related to CS education. Each of the teachers reflected on their early exposure to technology, professional growth and their motivations for incorporating CS into their teaching practices. They represented different educational backgrounds and levels of teaching, from elementary to high school and included both STEM-focused and general teachers.

5.4 Ethical Consideration

Participants in the study were informed prior to the commencement of the program about the purpose of the study, what it entails and their right to opt out at any time. Identifiable information was collected for the purpose of merging the pre-survey and the post-survey

responses. This was stipulated on the Institutional Review Board (IRB) document approved prior to the commencement of this research. All responses were anonymized during the data analysis to protect the identity of the participants. Participation in the research was voluntary and had no intended penalties for non-participation.

5.5 Training Programs

5.5.1 Content and Delivery Method

The teacher training program covered foundational computer science courses packaged into 3 graduate-level courses delivered by our College of Engineering. An additional 3-credit hour graduate course focused only on CS pedagogy was delivered by the College of Education; however, the focus in this paper will be on the CS courses, and also align with the content in typical CS0 and CS1 courses aligned with AP CS Principles and AP CSA, with the addition of pedagogical content, activities to create lesson plans, reflective journals and discussions. The following is an overview of the curriculum coverage:

1. Introduction to Computing for Educators (2 credit hours)
    - Overview of history of computing and modern impact on the society
    - Theories of Computational Thinking
    - Pre K-12 Standards
2. Computer Education Programming Fundamentals (1 credit hour)
    - Concept knowledge crucial for developing and teaching programming
    - Practice reading and writing basic program codes
    - Basic concepts of conditionals and looping constructs
3. Computer Programming for Educators (4 credit hours)
    - Basic concepts of programming (variables, control flow, functions, objects)

- Interactive lessons and engaging projects reinforce new skills
- Exploring pedagogical strategies for teaching programming

The training was delivered through a hybrid model that combined online modules and in-person workshops. The online modules leverage the power of digital learning by incorporating learning and content delivery through Codio learning platform [135]. This was particularly impactful, as it allowed for a wider range of access, especially for participants residing in rural areas.

5.5.2 Teacher Training Workshop

We conducted two-day in-person workshops, which were a blend of theoretical learning and hands-on activities that allowed for interaction with the participants. They included sessions on problem-solving, where teachers were partitioned into groups to discuss how problem-solving fits into their respective content areas and how it is being incorporated into their pedagogical styles. Participants were also engaged in coding sessions using block coding for elementary levels. The workshop also included collaborative learning through discussions and joint problem-solving activities, which brings about synergy among the participants thereby enhancing their professional development.

5.6 Data Analysis

5.6.1 Quantitative Data Analysis

A statistical paired t-test was used to determine the significance of the difference in the mean pre- and post-survey responses. Beyond statistical significance represented by p-values, the expression of results with effect sizes and confidence intervals provides a more comprehensive method of statistical results [136]. We therefore computed the effect sizes and confidence bounds to complement the p-values in measuring the effect of the training program. Hedge's g [137]was

calculated as a variation of Cohen's d [138]. This is because Hedge's g includes a correction factor that reduces the bias on small sizes, the results from Cohen's d.

5.6.2 4.6.2 Qualitative Data Analysis

A thematic analysis approach was used to identify recurring patterns and themes within the teachers' autobiographies. Each autobiography was read by multiple researchers to capture key narratives. Key phrases such as early exposure, student-centered goals, apprehension about CS, self-taught, desire to help students learn CS, no background in CS, etc. were identified and coded. The themes were cross-referenced to ensure consistency.

5.7 Results

The impact of the teacher training program on rural, town and suburban populations is shown in Table 4.1. A total of 24 teachers completed both the pre and post surveys, and comprised 13 teachers from town, 8 from rural and 3 from suburban locales. The data was filtered such that only the teachers that completed both the pre and post surveys were included in the analysis. Also, the results provide an understanding of the teachers' experiences, challenges and aspirations as revealed through their autobiographies. Key themes emerged from the thematic analysis that captured the diverse journeys of the teachers as they navigate the learning and teaching of computer science, which goes a long way to support the role of professional development in equipping teachers to prepare the next generation of CS education professionals. The following subsections provide specific details of the impact of the teacher training program.

5.7.1 Thematic Results from Teachers' Autobiography

The teachers' initial interactions with technology varied widely and are reflective of the state of technological development during their times. Early exposures ranged from using TRS 80s and Commodore 64 to working with gaming tools like Atari and Nintendo. Early exposures

Table 5.1. Data Analysis Results

Item	Locale	Effect Size	P Value	Lower Bound	Upper Bound
Teacher Identity					
I truly enjoy teaching Computer Science	Rural Suburban Town	0.4954 0.0000 -0.0712	0.1704 0.0000 0.8078	0.2044 0.0000 -0.1099	0.7956 0.0000 -0.0441
I see myself as a computer science teacher	Rural Suburban Town	0.1824 0.2469 0.3755	0.5630 0.4226 0.0532	0.1022 -0.2522 0.2643	0.3978 0.9189 0.6588
I have actively looked for opportunities to teach computer science	Rural Suburban Town	0.4322 -0.2066 0.3574	0.0331 0.4226 0.1511	0.2044 -0.9188 0.3084	0.7956 0.2522 0.7685
Commitment/Striving to Teach Computer Science					
I work hard to be the best computer science teacher that I can be	Rural Suburban Town	0.7221 0.6532 0.1878	0.1705 0.4226 0.5345	0.2044 -0.2521 0.1322	0.7956 0.9189 0.3294
I attend professional development to help me keep up with the latest developments in [CS] teaching	Rural Suburban Town	0.3383 0.9237 0.0000	0.3506 0.1835 1.0000	0.1533 -0.5044 0.0000	0.5967 1.8377 0.0000
I advocate for more students to take courses in computer science	Rural Suburban Town	0.2832 0.6532 0.1758	0.4512 0.4226 0.5845	0.1022 -0.2522 0.0881	0.3978 0.9189 0.2196
Confidence in Teaching Computer Science					
How well can you implement alternative strategies in your computer science classroom?	Rural Suburban Town	0.8092 0.4131 0.0642	0.1114 0.1835 0.8193	0.3066 -0.5044 0.0441	1.1934 1.8377 0.1098
To what extent can you gauge student comprehension of what you have taught?	Rural Suburban Town	0.5970 0.3939 0.1399	0.2443 0.1835 0.7112	0.3066 -0.5044 0.0881	1.1934 1.8377 0.2196
How much can you do to adjust your lessons to the proper level for individual students?	Rural Suburban Town	0.3611 0.2066 0.0000	0.4869 0.4226 1.0000	0.2044 -0.2522 0.0000	0.7956 0.9189 0.0000
How much can you do to get students to believe they can do well in computer science?	Rural Suburban Town	0.7221 0.0000 0.3273	0.0062 1.000 0.3905	0.3577 0.0000 0.2203	1.3923 0.0000 0.5490
How much can you do to help your students value learning computer science?	Rural Suburban Town	0.7649 0.0000 0.2906	0.0.0062 1.0000 0.3665	0.3577 0.0000 0.1762	1.3923 0.0000 0.4392
How much can you do to foster student creativity in computer science?	Rural Suburban Town	0.3708 0.0000 0.1869	0.0331 1.0000 0.5696	0.2044 0.0000 0.1322	0.7956 0.0000 0.3294

were limited to basic tasks, leaving gaps in foundational knowledge of programming and problem-solving skills. Educators frequently encounter challenges when learning or teaching CS, including limited prior knowledge, fear of failure, and difficulty adapting to new tools and programming concepts.

These challenges highlight and importance of providing structured learning opportunities and support system to build educators' confidence and teaching capabilities. A recurring them was the teachers' motivation to grow professionally by learning computer science. Many of the teachers viewed professional development as an opportunity to enhance their teaching practices and prepare students for careers in technology. The teachers consistently emphasized their desire to inspire and empower students through CS education. A particular teacher shared success using Code.org and Scratch to engage students in collaborative problem solving.

Many of the teachers highlighted enthusiasm for applying computer science concepts to real-world problems such as robotics, web development and practical coding projects.

5.7.2 Bolstered Professional Identity

The training program impacted rural teachers' professional identity, evident from the moderate effect size of 0.4954 in Table 5.1 where teachers report enjoyment of teaching computer science. Although the p-value of 0.1704 suggests a statistically insignificant change, the 95% confidence bound ranged from 0.2044 to 0.7956, indicating the true effect size lies within that range – signifying a meaningful effect. Similarly, the training gave the teachers strong motivation to actively engage in activities that provide opportunities to teach computer science, thereby bolstering their professional identity.

5.7.3 Commitment to Professional Growth

The program instilled a sense of commitment to advancing the teaching of computer science among rural teachers, indicated by an effect size of 0.7221 for striving hard to be the best computer science teacher. In addition, the training led to a positive shift in teachers' willingness to participate in professional development programs to keep pace with current developments in computer science teaching as shown in Table 5.1.

5.7.4 Increased Confidence and Competence

Table 5.1 shows that the rural teachers demonstrate increased confidence — they are more able to gauge students' level of comprehension as indicated by an effect size of 0.5970. Moreso, an effect size of 0.8092 for implementing alternative teaching strategies highlights a boost in the confidence of the teachers and enhance competence in teaching computer science. Trailing closely behind the ability to measure students' level of comprehension and building alternative teaching strategies is the rural teachers' confidence in their ability to adjust lessons for individual students and foster students' belief in their ability to become successful computer science students, evident by effect sizes of 0.3611 and 0.7221, respectively. Rural educators are more ready to engage students in activities that foster creative thinking after the program than they were before the program.

5.8 Discussion

The findings from the teacher training program highlights the potential for professional development programs to transform rural educators and equip them for advancing computer science education. Given the unique challenges faced by rural teachers, which includes a shortage of qualified teachers and limited access to resources, the outcome of the program is significant.

A profound impact of the program is on teachers' identities with a moderate effect size which indicates meaningful impact despite a statistical insignificance. Teachers reported greater enjoyment in teaching computer science after the program and are more motivated to seek out opportunities to teach computer science.

The program also had a positive impact on teachers' commitment to professional growth, evident by willingness to participate in professional development programs that would keep them up to date with advancements in computer science education.

The teachers' demonstration of increased ability to gauge students' level of understanding, adjust lessons to meet students' needs and implement alternative teaching strategies is an indication that the program effectively equipped the teachers with the requisite skills and confidence in teaching computer science in rural areas.

5.8.1 Broader Impact of The Teacher Training Program

With the goal of achieving a broad impact, we initially attempted to reach at least 50 teachers in 50 different schools. However, while this target was not met, we successfully reached 34 unique school districts - out of which 27 were rural.

About 18 of the school districts reached by our program have committed to offering either AP CS Principles-aligned course or our AP CS A-aligned course, with 8 of those districts planning on offering the associated AP exam (see Table 5.2). More schools reported that they were going to integrate at least parts of this curriculum into their classrooms. Roughly 363 students will be taking a CS course designed by our program, and around 2,000 students will be reached through teachers who completed at least some part of the professional development provided as a result of this program.

Table 5.2. Impact by Numbers

Group	Total
Teachers who completed at least 1 course	51
Teachers who completed 10 credit hours*	22
School Districts Reached	34
Rural	27
Urban	7
Schools offering the Courses Fall 2023	18
Schools integrating some part of the Courses	25
Number of new AP CS A courses offered	3
Number of new AP CS Principles courses offered	7
Est. Students Reached By Our Curriculum Fall 2023	363
Est. Students Reached By a Teacher Trained by this Program	2052

* 12 actively working to finish

5.9 Limitation of the Study

The findings of this study are limited by small sample sizes, which reduces the ability to generalize the results to the rural populations. This reduces the statistical power of the study and in turn makes it difficult to obtain significant effects and draw solid conclusions.

More so, the sample has a lack of diversity which makes it difficult to represent a wide range of experiences and viewpoints found in rural areas, thereby missing out on important points. As a result of this, it is essential for future research to involve larger and more diverse samples in order to guarantee their applicability across rural areas.

Currently, there are about 120 teachers in the program and there are plans to recruit more later this year. This will create room for additional data to be collected more quickly, especially as we will be using similar survey instruments to explore how rural students are impacted by our curriculum.

Chapter 6 - Evaluating Learning Perspectives in Teacher Training Programs

6.1 Introduction

The increasing evolution of computer science education has prompted urgent calls for its inclusion across all levels of K-12 education [6], [15]. As computational thinking and programming become foundational skills in computer science education, preparing teachers to deliver high quality computer science education has emerged as a critical challenge. Teacher professional development programs play a crucial role in shaping the pedagogical practices and learning attitudes of the teachers. These programs are not merely means through which teachers acquire skills in theory and practice but also serve as spaces where teachers reflect on their beliefs, develop new understanding with the goal of reshaping their professional roles as teachers [139].

The learning perspectives of teachers enrolled in computer science training programs offer a valuable lens for understanding how educators interpret, internalize and apply new knowledge within their various unique teaching domains [139]. Unlike traditional content delivery models, effective teacher training especially in evolving discipline like computer science relies heavily on learner-centered and reflective approaches that help to empower teachers to express their understanding through experience, collaboration and critical reflection [140]. Consequently, exploring how teachers perceive their learning through professional development programs provide key insights into their evolving identities, motivations and teaching practices [141].

Computer science teacher training is particularly complex because it demands both cognitive and emotional shifts to the teachers. Teachers may not only acquire new knowledge of

the contents being taught, such as programming, data structures and algorithms, but also adopt new ways of thinking - thinking computationally [15]. In addition to this, many teachers encounter challenges in aligning their established teaching philosophies with the open-ended, problem-solving nature of computer science. This difficulty often emerges from the contrast between traditional instructional methods and the exploratory, inquiry-based approaches that computer science demands. For instance, Yadav et al. [142] found that teachers often face difficulties when incorporating computational thinking into their teaching practice, as it requires a shift from traditional teaching methods to more problem-solving approaches. Professional development programs must therefore address not only conceptual learning but also teachers' confidence, self-efficacy and pedagogical beliefs, if the teachers must be at their best to deliver high quality computer science teaching.

Several studies have evaluated the outcomes of computer science training programs by examining not only the content knowledge teachers acquire – such as programming concepts, computational thinking and computer science pedagogies – but also how effectively they transfer that knowledge into classroom practices [143]. However, few of these studies have focused on the qualitative dimensions of teacher learning, especially how teachers articulate their experiences, struggles, growth and identities during the learning process. By investigating teachers' own words through reflective feedback, this study aims to illuminate the complexity of teacher learning in computer science professional development and contribute to a deeper understanding of what supports meaningful and sustained professional growth in this rapidly expanding field.

This chapter therefore presents a thematic analysis of reflective journals collected from participants in computer science focused professional development courses, with goals:

1. To identify core themes in how teachers perceive and articulate their learning journeys
2. To explore the emotional and cognitive dimensions of teacher development
3. To inform the design of future professional development programs that are responsive to the challenges and aspirations of computer science teachers

6.2 Background

Understanding teachers' learning perspectives is essential for designing and implementing effective professional development. By examining the benefits of understanding these learning perspectives, we can gain valuable insights into enhancing professional development programs.

6.2.1 Benefits of Understanding Teachers' Learning Perspectives

Understanding teachers' learning perspectives plays a crucial role in the design, implementation and evaluation of professional development programs. The experiences, challenges and reflections of teachers are useful in providing insights into how teacher training and professional development programs can be improved in order to meet their needs and by extension, enhance students' learning outcomes. According to Avalos [139], as much as professional development programs play vital roles in equipping teachers, the unique learning needs of the teachers must be put into consideration. This is because learning in a professional context is most effective when the teachers are involved in emotional and cognitive engagements which encourage them to reflect on their learning journeys [139]. Moreso, when reflective conversations are combined with feedback from teacher training programs, if promotes a deeper understanding of the teaching practices [144] while intensive reflection in teacher training promotes critical thinking among the teachers and improve their self-awareness [145]. Teachers have reported that reflection along learning journey allowed them to identify the areas where

they need to make improvements and informed decisions regarding their learning strategies [146] and influence how they eventually apply the knowledge and practices they acquire in their classrooms [141].

Professional development initiatives are more successful when teachers are motivated internally to participate and apply what they learn. Research shows that understanding intrinsic motivators such as curiosity, sense of accomplishment or relevance to students can help professional development program designers to create more meaningful experiences for the teachers [147]. For instance, professional development that aligns with teachers' values and classroom goals can promote a stronger commitment to instructional adjustments for further benefits of the teachers [140].

Another significant benefit of understanding teachers' learning perspectives is the feedback it provides for iterative refinement of professional development programs. Qualitative insights obtained from teacher reflective journals, surveys and interviews can help researchers identify which components of the professional development programs are effective, confusing or misaligned with the teachers' needs [148]. Specifically, teachers' reflections on challenges with computational thinking can guide the development of targeted workshops or online learning modules that address specific misconceptions [15].

Understanding the learning perspective of teachers also serves as a bridge between theoretical frameworks and classroom realities. Teachers often interpret professional development contents through the lens of their own classroom experiences, cultures and student populations [149]. Therefore, without incorporating teachers' perspectives, there is a risk that professional development will remain abstract and devoid of practical challenges that the teachers face.

6.3 Methodology

6.3.1 Research Design

This study employs a qualitative research design to investigate the learning perspectives of teachers engaged in a computer science professional development program. The core objective is to examine the content of these reflective narratives to uncover themes that are related to cognitive, emotional and pedagogical transformations during the teacher training programs. To enhance the robustness of the thematic analysis, this study adopts a dual analytic framework involving manual human annotation and automated thematic tagging using a Large Language Model (LLM), specially OpenAI's ChatGPT. A comparison of human tagging versus LLM-based tagging is performed, both visually and using ROUGE metric. This allows the study to remain grounded in qualitative educational research while exploring the potential automated methods for future use in professional development evaluation and feedback systems.

6.3.2 Data Collection and Description

The data used for this study was collected from 50 teachers who participated in professional development courses with a focus on a course titled "Introduction to Programming for Educators" — the first in a sequence of computer science professional development courses designed specifically for K-12 teachers. This course serves as a foundational introduction to programming concepts, pedagogical strategies and computational thinking. It is modeled after the CS0 curriculum, focusing on basic programming constructs such as variables, loops, conditionals, functions, and simple data structures. The course content also aligns conceptually with introductory modules of AP computer science principles, emphasizing problem-solving, algorithmic thinking and the relevance of computing in educational contexts.

The teachers were required to provide reflective journals of no more than two pages on their experiences. Each journal entry requirement correlates with the modules that were being covered at that point in time, with a focus on exploring their experiences with computing and their general thoughts on teaching computer science. The journal entry reflection questions include:

- ***Computing Autobiography:*** Teachers were required to reflect on their early encounters with computing, their attitudes towards computer science over time, the way they currently utilize computer science in their daily lives both personally and professionally as well as the classes they have taken. They were also required to discuss their goals and expectations from the course.
- ***Navigating the Beginning:*** Teachers were asked to reflect on their experiences after completing the initial phase of the course which included an introduction to the field of computer science and their first few programming assignments. They were encouraged to share how they currently feel about their learning journey – where they excited, confident, overwhelmed, or uncertain – and to describe specific experiences that have contributed to these emotions. Teachers were also asked to reflect on the challenges and breakthroughs they have encountered, the strategies they have found helpful and how these early experiences have shaped their perceptions of teaching computer science.
- ***Learning Theories:*** Teachers were asked to reflect on learning theories they had come across in the course and how they might apply these theories in their respective teaching. They were also required to discuss how to engage with a diverse audience of students and strategies for equitable participation in computer science education.
- ***Learning Ecologies:*** The teachers were asked to describe the learning opportunities that are available to their students as well as the disparities in computer science education.

They were also required to identify potential strategies that would enable students to learn computer science effectively.

- ***Finding Your Teaching Style/Approaches in Computer Science:*** The teachers were asked to reflect on the various approaches to teaching computer science introduced throughout the course, both in reading and content lessons. They were encouraged to consider which methods resonated with them most as learners, and which they envision using in their own teaching practices. Teachers were also invited to reflect on the strategies they currently use in their classrooms that could be adapted to effectively teach computer science.

6.3.3 Human Annotation Procedure

The study employed a manual thematic coding procedure using the Taguette qualitative research tool. Taguette is an open-source, web-based platform designed specifically for researchers conducting qualitative text analysis. It enables users to upload textual documents, highlight relevant excerpts and assign user-defined thematic tags to each segment, thereby facilitating systematic and traceable coding. It also facilitates the generation of structured outputs for further analysis or comparison with other annotators or LLM-generated tags in the case of this research.

6.3.4 Large Language Model-Based Thematic Tagging

To examine the reliability of the thematic analysis, the same set of teacher journal entries was subjected to thematic tagging using OpenAI's ChatGPT. Each journal was submitted to the model with a structured prompt designed to simulate the role of a qualitative researcher. The prompt formulated are shown below:

Prompt:	*"You are an expert in qualitative research. The following text consists of reflective journal entries written by teachers participating in a computer science professional development program. Conduct a thematic analysis by identifying all the main themes emerging from the reflections. Use concise, meaningful descriptions that capture emotional, cognitive and pedagogical dimensions of the teachers' experiences. Include the frequencies of how many individual highlights support that theme.*

6.3.5 ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a widely used metric for evaluating the quality of generated texts by comparing it to a set of reference summaries or keywords. It is originally developed for summarization tasks [150], useful for assessing overlap between automatic and human-generated texts.

ROUGE-1 refers to the overlap of unigrams (single words) between the generated and reference texts, indicating basic word-level similarity. ROUGE-2 measures bigram (two-word sequence) overlap, capturing more contextual agreement while ROUGE-L evaluates the longest common subsequence, reflecting the overall alignment between two sets of texts. In this study, ROUGE was used to compare the thematic labels generated by ChatGPT (LLM-based) with those generated by human coder.

The interpretation of ROUGE scores varies based on the task but generally, a ROUGE-1 score above 0.5 is considered high, indicating strong lexical similarity; scores between 0.3 and 0.5 suggest moderate similarity while scores below 0.3 reflect low similarity. ROUGE-2 scores are typically lower than ROUGE-1 because bigram matches are harder to achieve; values above 0.3

are strong, between 0.1 and 0.3 are moderate and below 0.1 are considered low. ROUGE-L, which captures sequential structure often lie between ROUGE-1 and ROUGE-2.

6.4 Analysis and Results

6.4.1 Teachers' Autobiography

From Figures 6.1 and 6.2, the human-coded analysis places strong emphasis on broader terms like "Attitude Towards CS", "K-12 Experience", and "Care for Students", with particularly high counts for institutional elements such as "Programming Expectations" and "Occupation.

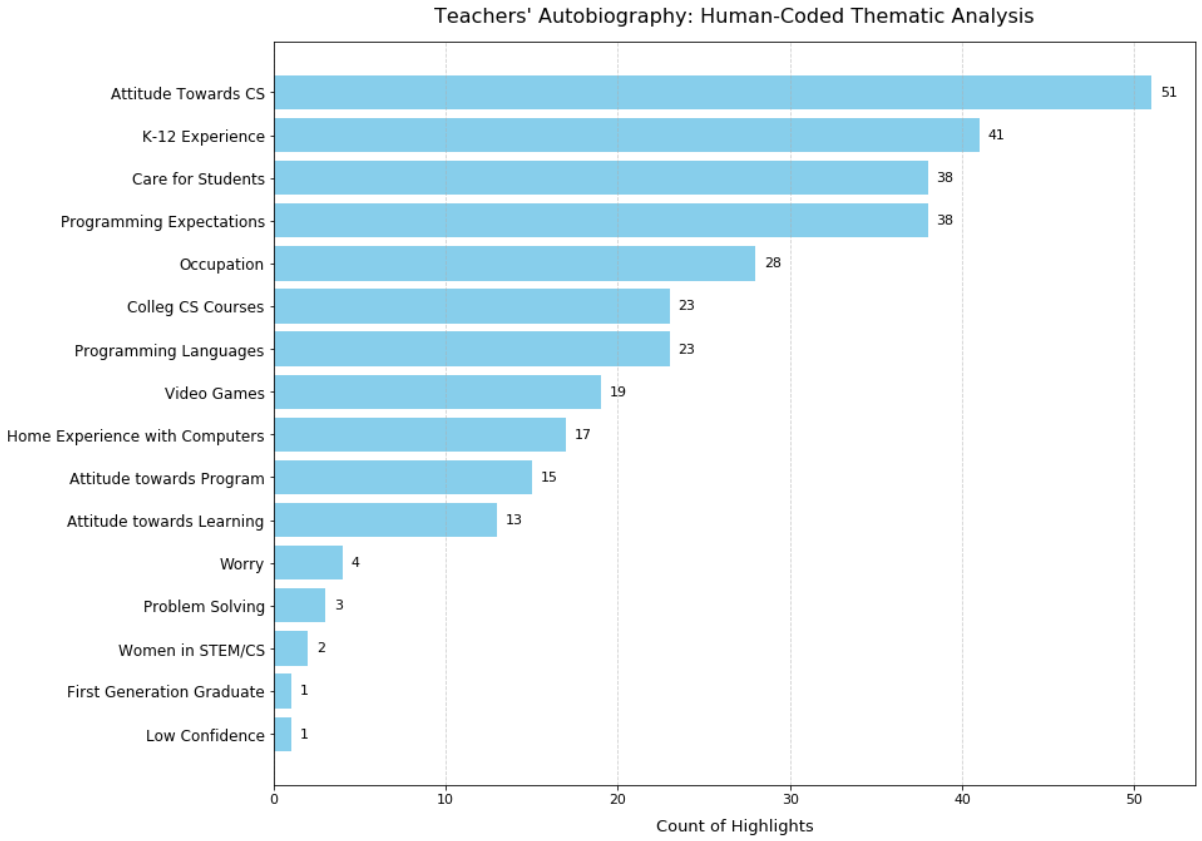


Figure 6.1. Teachers' Autobiography: Human Coded

On the other hand, the LLM-coded themes center heavily around personal and emotional journeys into computing, such as "Early Encounters with Computers", "Teaching Aspirations and Integration", and "Growth Mindset and Learning Attitudes". These themes capture

motivational factors in teachers' learning trajectories. Themes such as "Video Games" and "Home Experience with Computers" appear in both analyses, though with differing emphasis because LLM focuses on Gaming and Educational Software, while human highlights Video Games more generally.

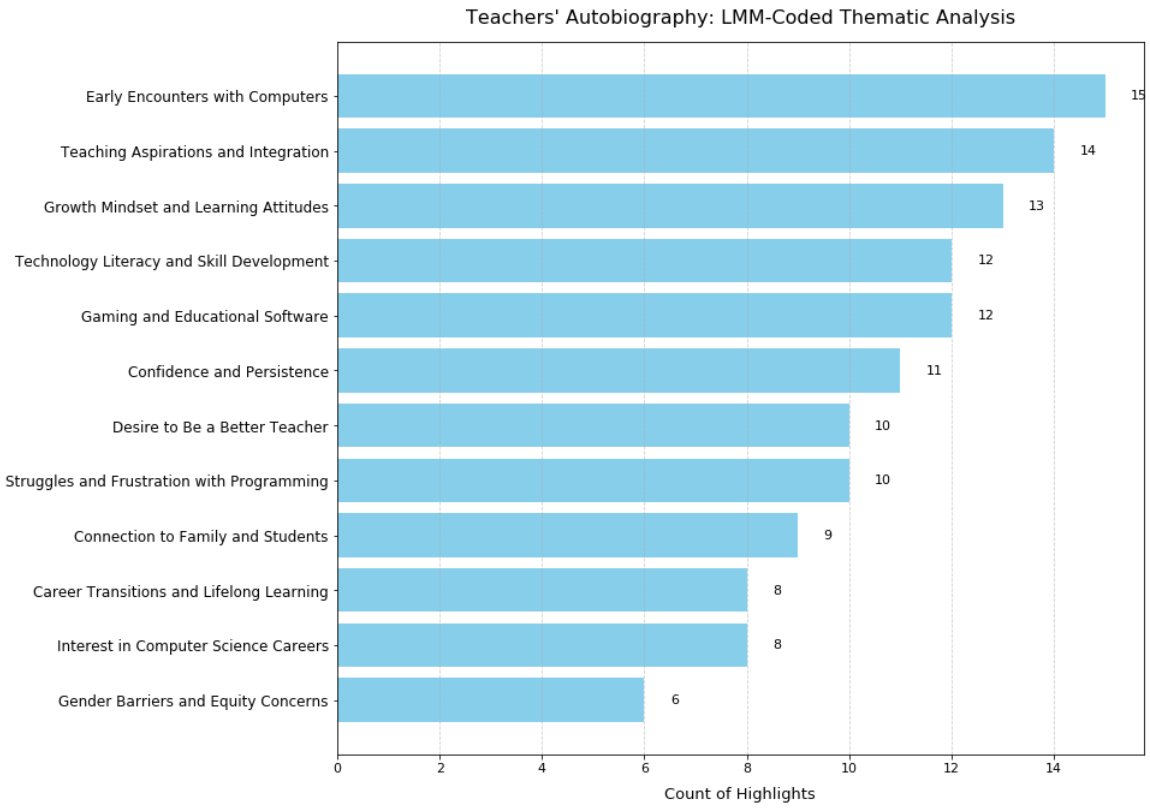


Figure 6.2. Teachers' Autobiography: LLM-Coded

Despite some conceptual alignments such as learning attitudes, confidence and technology experience, the thematic analysis differ substantially, as reflected in the relatively low ROUGE scores: ROUGE-1 at 0.143, ROUGE-2 at 0.022 and ROUGE-L at 0.130. These low overlap scores suggest that while both approaches captured relevant themes, the LLM and human coders often presented them in different ways. This underscores the need for complementary insight when combining both coding approaches.

6.4.2 Navigating the Beginning

The comparison between the human-coded themes (Figure 6.3) and the ChatGPT-generated themes (Figure 6.4) reveals a strong alignment in the overall focal areas of teacher reflections, though differences in granularity and phrasing are apparent.

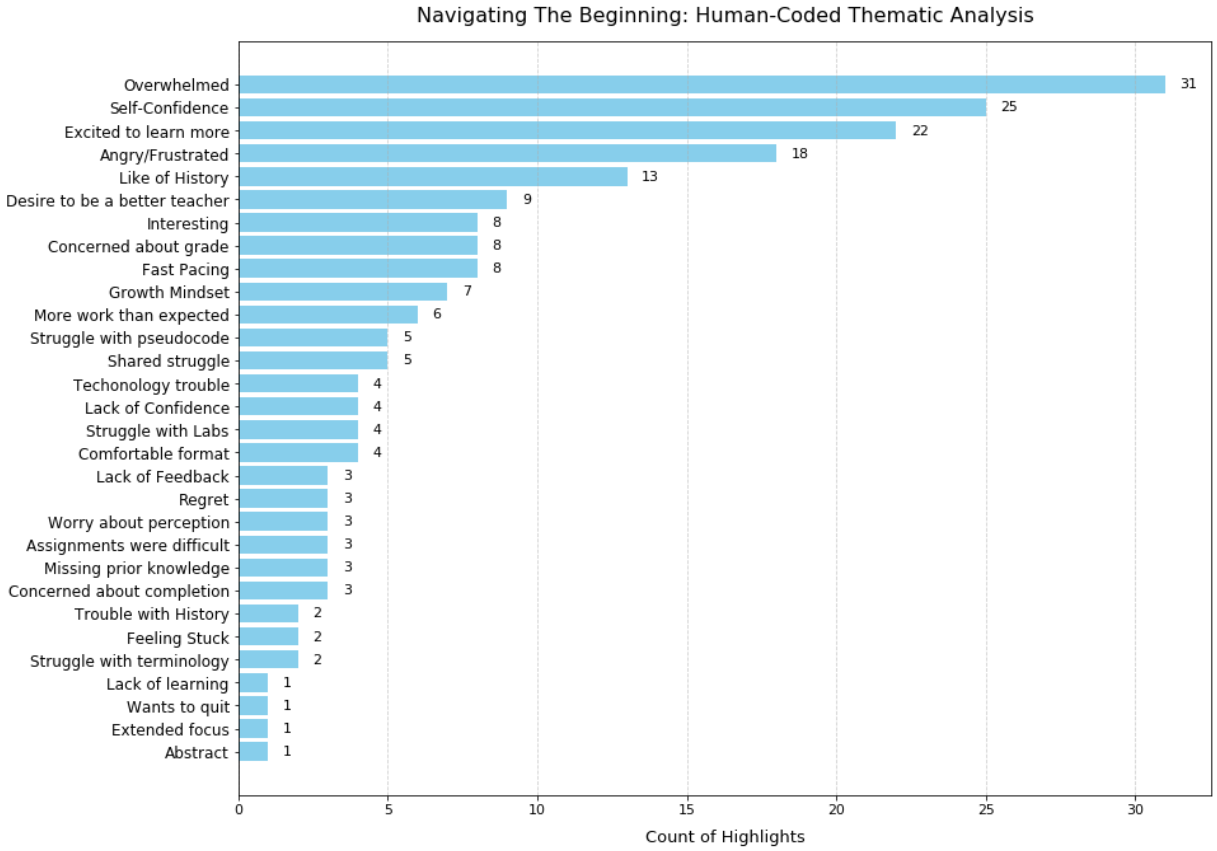


Figure 6.3. Navigating The Beginning (Human Coded)

Both analysis highlight “Overwhelm” or “Overwhelm and Time Pressure” as the most dominant theme, which reflects the intensive demands of the course. Themes such as “Excited to Learn More” (human) and “Excitement and Curiosity” (ChatGPT), along with “Growth Mindset” (Human) and “Growth Mindset and Perseverance” (ChatGPT), show semantic and conceptual overlap, suggesting that the AI captured key emotional and motivational patterns expressed by teachers. Similarly, ChatGPT’s “Empathy Toward Student Struggles” echoes the human-coded

"Shared struggle" and "Struggle with Labs/Pseudocodes", capturing the dual role of teachers as learners and future facilitators of computer science instruction.

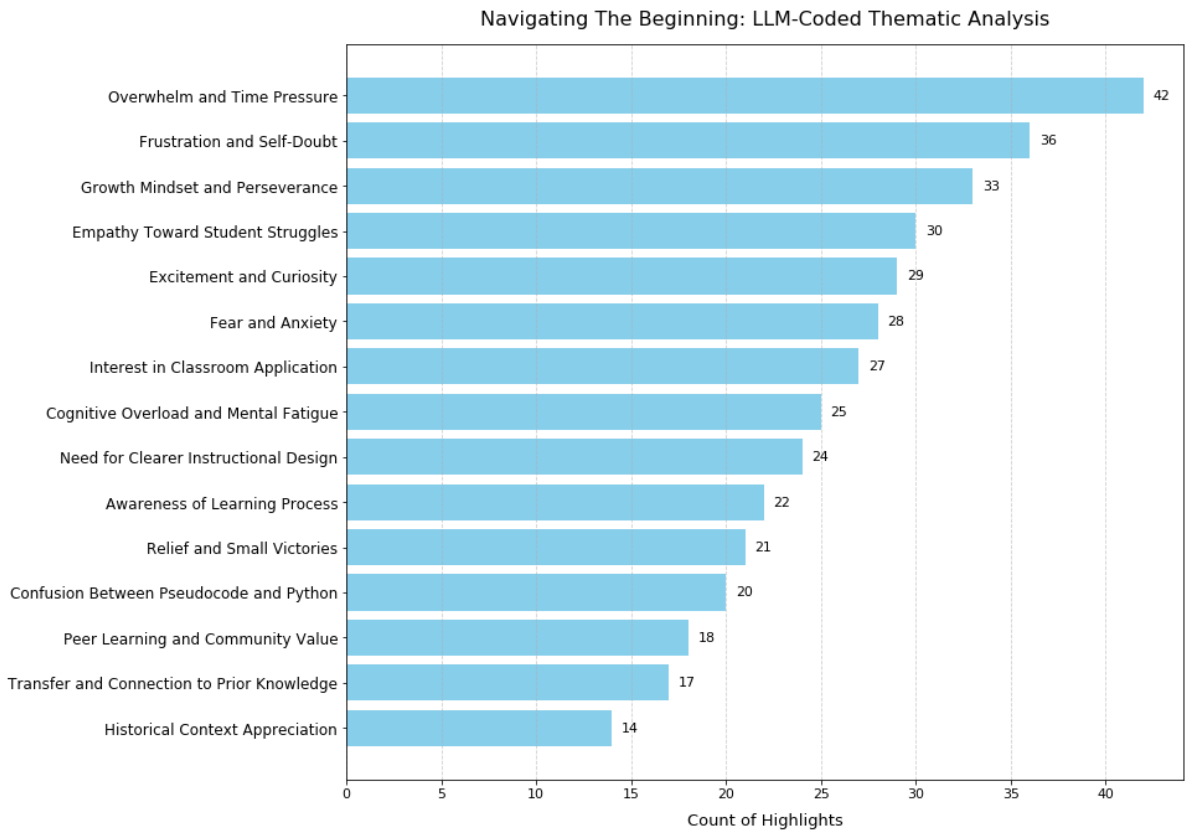


Figure 6.4. Navigating The Beginning (AI Coded)

However, the divergence in labeling specificity, with human coding using narrower codes like "Fast Pacing" or "Trouble with History" while ChatGPT uses broader, synthesized themes contributes to a moderate ROUGE-1 and ROUGE-L scores of 0.2623 and a low ROUGE-2 score of 0.0794. This indicates that while the AI's wording overlaps with the human-coded content at the word and concept level, its phrasing and structure differ. Overall, LLM-generated analysis reasonably aligns with human thematic analysis of the initial teachers' learning experiences, particularly in emotional and cognitive dimensions.

6.4.3 Learning Theories

The human-coded and LLM-coded thematic analysis reveals a marked contrast in the teachers' reflections on learning theories.

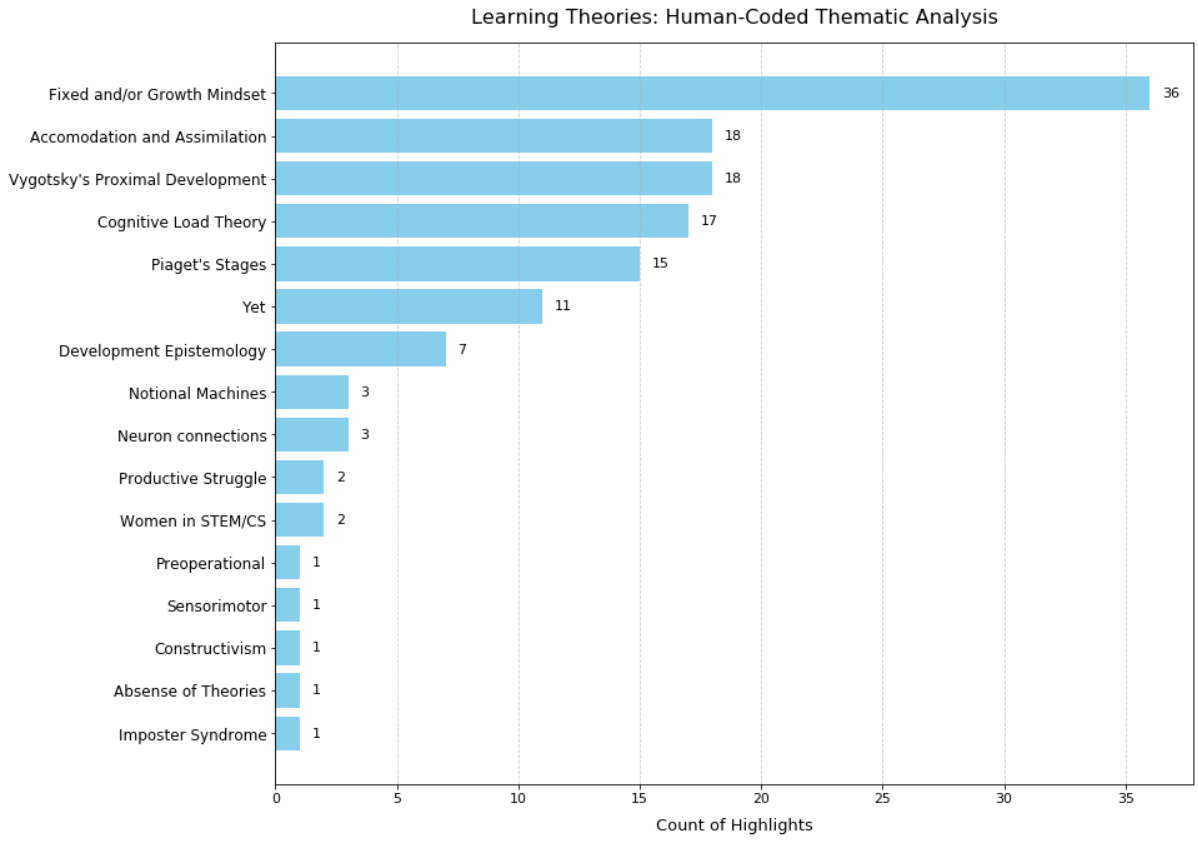


Figure 6.5. Learning Theories: Human Coded

The LLM-coded graph (Figure 5.4) shows a broader, more abstract categorization of themes. For example, it identifies high-level pedagogical and emotional constructs such as "Growth Mindset and Perseverance", "Overwhelm and Cognitive Load" while integrating theoretical anchors like Bloom's Taxonomy, Piaget/Cognitive Development and Vygotsky's ZPD, presenting them as structured conceptual frameworks that emerged meaningfully from the reflections.

In contrast, the human-coded graph (Figure 5.3) presents a more granular and literal interpretation. The themes are often smaller conceptual units or phrases taken directly from the text, such as "Yet", "Neuron connections" or "Women in STEM/CS". This suggests that the

human coder was guided by a close reading of the text, capturing the nuance and terminology specific to the course content.

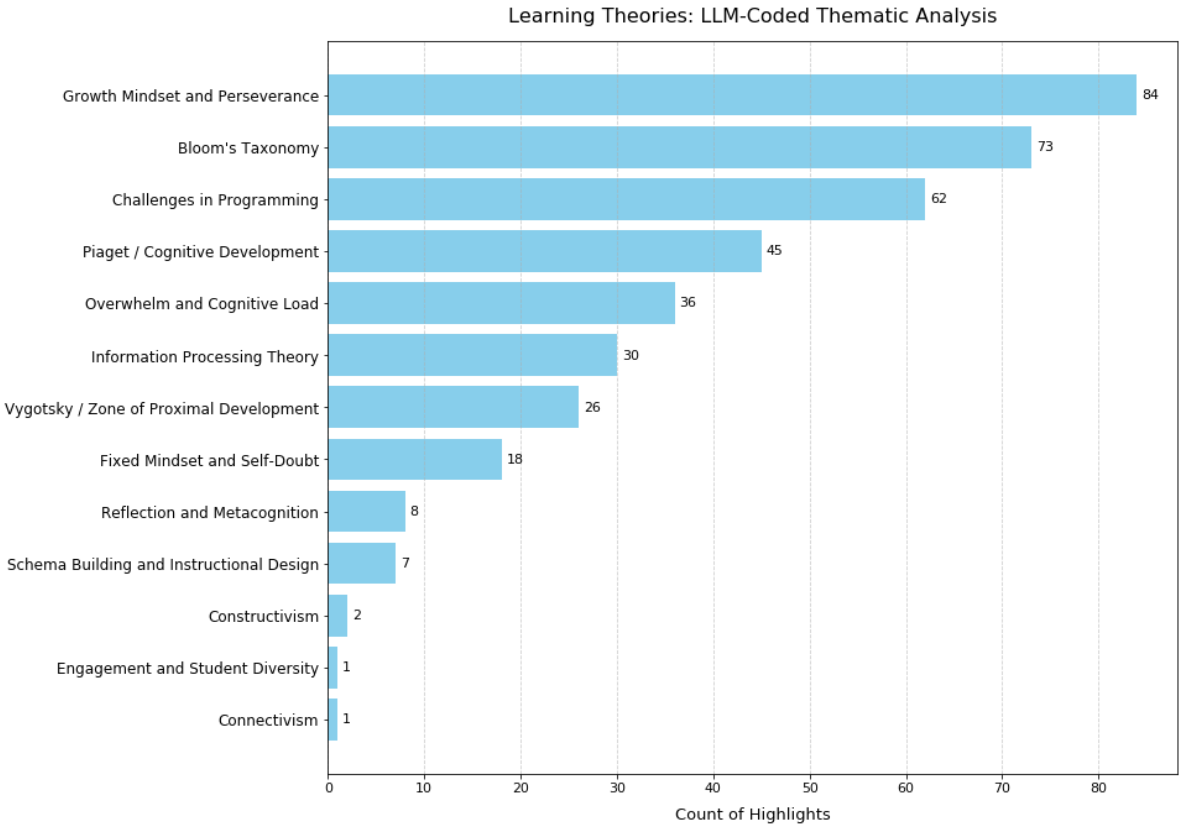


Figure 6.6. Learning Theories: LLM-Coded

There is a moderate degree of overlap in the individual word usage and longest common subsequence between the LLM and human themes – evident by the ROUGE-1 and ROUGE-L scores of 0.44 and 0.41 respectively. This suggests that many of the same concepts were recognized, although with different levels of abstraction or phrasing. However, the ROUGE-2 score is low at 0.07, reflecting that the two sets of themes diverged significantly in how those concepts were expressed in multi-word phrases.

6.4.4 Learning Ecologies

Figure 6.7 and Figure 6.8 show the human-coded and LLM-coded charts of the thematic analysis respectively. The comparison of the two approaches reveals both overlapping insights and distinct focal points. The LLM-coded analysis emphasizes themes such as “Limited Access and Opportunities”, “Gender and Racial Disparities”, and “Teacher Training and CS Integration”, with high frequencies of 18, 16 and 14 respectively.

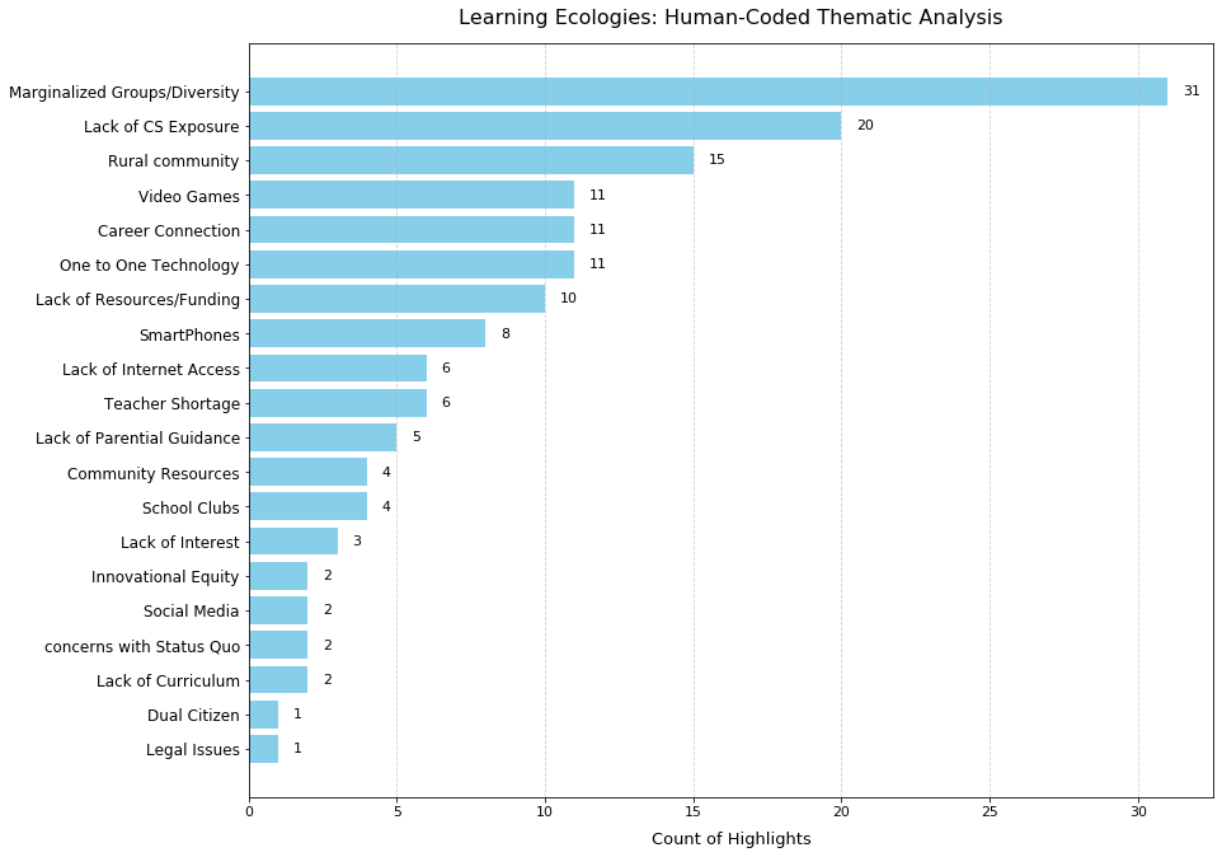


Figure 6.7. Learning Ecologies: Human Coded

These reflect a view of structural and instructional challenges in equitable access to computer science education. In contrast, the human-coded themes focus heavily on “Marginalized Groups/Diversity” (31 highlights) and “Lack of CS Exposure (20 highlights), which resonate with similar concerns but use different wording and granularity.

A notable area of comparison is technology use. The LLM-coded theme “Device Access vs Productive Use (13 highlights) aligns with the human-coded entries “One-to-One Technology (11) and “Smartphones (8). However, the human coders also identified Video Games (11) as a theme, suggesting a more nuanced attention to specific student behaviors and their influence on learning ecologies, a detail not captured by the LLM.

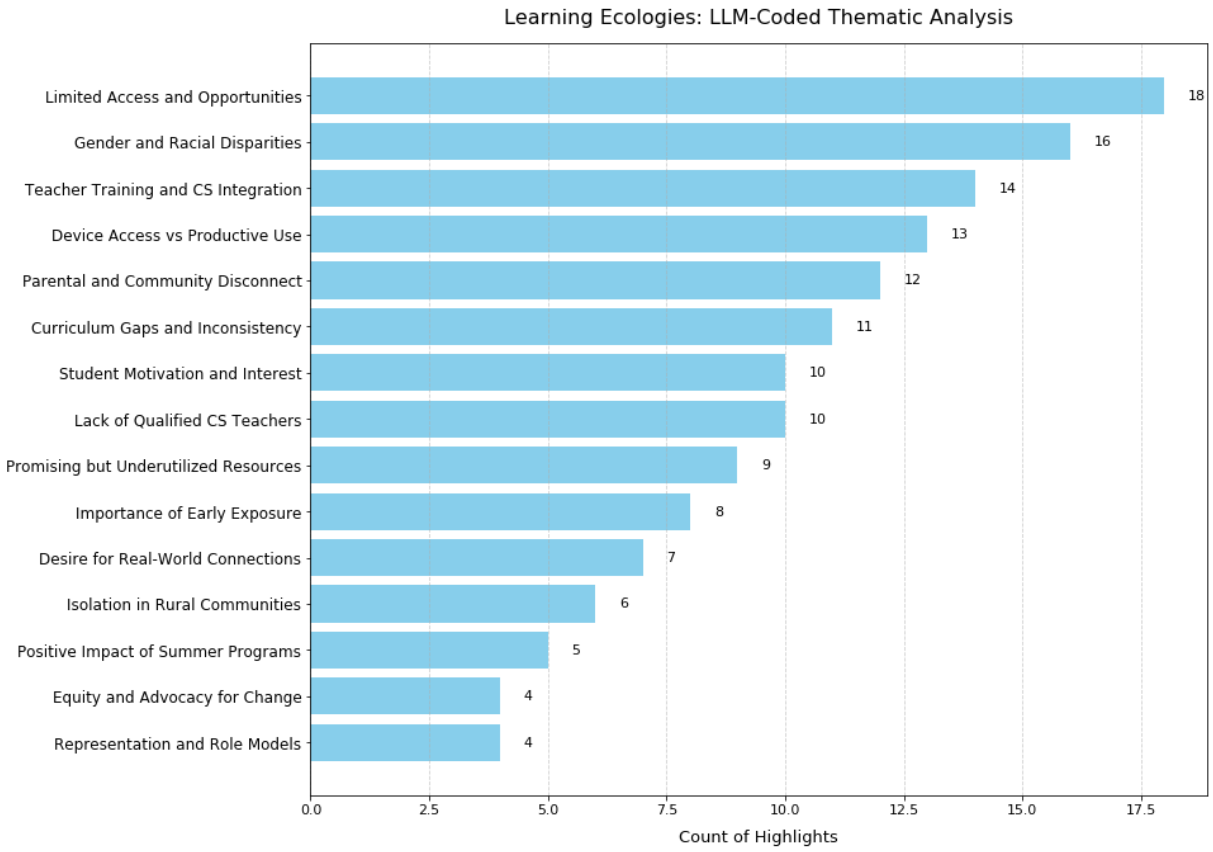


Figure 6.8. Learning Ecologies: LMM-Coded

Additionally, “Lack of Qualified CS Teachers” in the LLM-coded results (10 highlights) aligns with “Teacher Shortage (6) from the human-coded side, although it is less prominent in the human data. Other themes like “Parental and Community Disconnect” in the LLM-based results roughly correspond to “Lack of Parental Guidance” and “Community Resources” in the human-coded data, although with less frequency (5 and 4 respectively).

The overall ROUGE scores – ROUGE-1: 0.31, ROUGE-2: 0.056, and ROUGE-L: 0.31 suggest moderate lexical overlap but highlight semantic differences. This indicates that while LLM is capturing many of the same concerns at the top level, the human analysis includes a wider range of context-specific themes. While LLM offers breadth and generalization, human coded analysis captures fine-grained, context sensitive nuances.

6.4.5 Teaching Approaches in Computer Science

The comparison between the LLM-based and human-coded thematic analysis of teacher reflections on teaching approaches in computer science reveals both alignment and divergence (Figure M and Figure N)

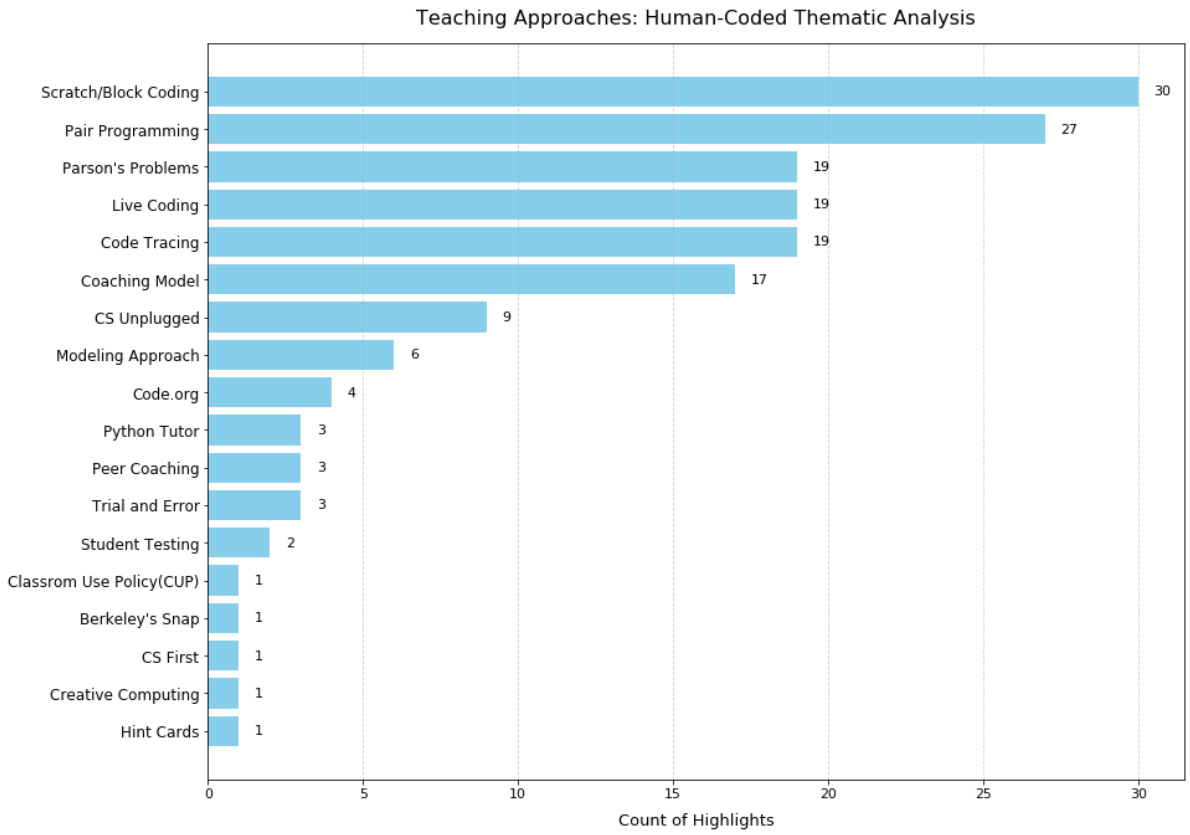


Figure 6.9. Teaching Approaches: Human-Coded

The LLM-coded chart emphasizes themes such as “Peer Programming and Collaboration” (22 highlights), “Live Coding and Modeling (20 highlights), and “Code Tracing and Debugging” (18 highlights) as dominant strategies recognized by teachers, closely followed by “Coaching and Guided Support” and “Parsons Problems”. These themes reflect a strong emphasis on interactive, student-centered approaches that support learning.

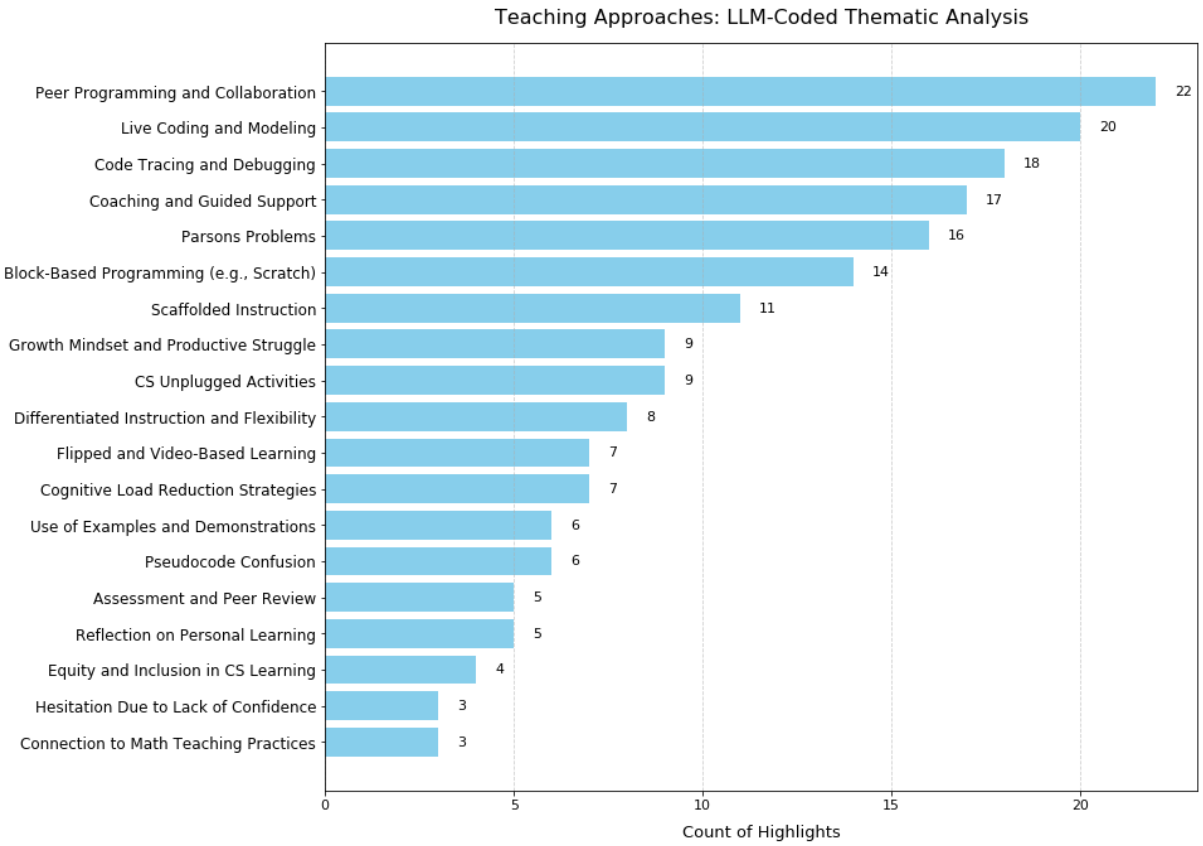


Figure 6.10. Teaching Approaches: LLM-Coded

On the other hand, the human-coded chart presents “Scratch/Block Coding” (30 highlights) and “Pair Programming (27 highlights) as the most prominent strategies followed by “Parson’s Problems”, “Live Coding”, and “Code Tracing” (all with 19 highlights each). While both analysis capture core pedagogical strategies like block-based programming, pair work and modeling, the human-coded analysis appears to be more granular in tool-specific coding (by mentions of Code.org, Python Tutor and Berkeley’s Snap), while the AI-coded version

aggregates strategies into broader concepts like “CS Unplugged Activities” and “Cognitive Load Reduction Strategies”.

The ROUGE scores further underscore this pattern of partial alignment, The ROUGE-1 and ROUGE-L scores both stand at 0.241, indicating a fair lexical overlap between the two sets of themes. However, the ROUGE-2 score is 0.0, suggesting limited phrase-level alignment. This could be due to the AI’s compound phrasing compared to the shorter and tool-specific labels used by the human code.

6.5 Discussion and Pedagogical Implications

The thematic analysis of teacher reflections conducted in this study reveals important insights into the learning journeys of educators in computer science professional development programs. These findings demonstrate that teacher learning is not merely a technical process of acquiring programming skills or content knowledge but a multi-dimensional experience that includes cognitive, emotional and pedagogical growth. Teachers described moments of both struggle and victory, with themes such as “Overwhelm and Cognitive Load”, “Confidence and Persistence”, “Growth Mindset and Learning Attitudes”, suggesting that emotions play a crucial role in the learning process. Professional development must therefore be designed to provide emotional support and foster resilience beyond technical instruction.

The teachers’ autobiography reflections supports the diverse trajectories that teachers bring into the learning process. Some shared stories of early exposure to technology and strong motivation, while others conveyed anxiety, self-doubt and identity issues. For teacher training programs to be effective and efficient, they must acknowledge and support these diverse experiences and aspirations.

The cognitive dimension of teacher professional development was seen in the reflections that engaged with foundational learning theories such as Bloom's Taxonomy, Vygotsky's Zone of Proximal Development and Piaget's Cognitive Development. The presence of these theories in both human and LLM coded data indicates that teachers are meaningfully engaging with conceptual frameworks and seeking to integrate them into practice. Pedagogically, teachers expressed a desire to adopt and adapt active, student-centered strategies such as Peer Programming, Live Coding and Scratch Programming. All of these reflect a shift from traditional approaches to a more interactive classroom teaching.

6.6 Limitations

This study, while informative, is not without its limitations. The sample size of 50 teachers, though sufficient for exploratory thematic analysis, constrains the generalizability of the findings to broader populations. The human-coded analysis was conducted by a single researcher, introducing the possibility of subjective interpretation and limiting inter-rater reliability. ROUGE as a metric provides insight into textual overlap but does not fully capture conceptual or semantic equivalence. As a result, some valid thematic similarities may be undervalued.

Chapter 7 - Conclusion

7.1 Summary and Review of Contributions

This dissertation explored strategies to advance computer science education by addressing two crucial challenges: Identifying and supporting struggling students and equipping teachers, especially those in underserved communities, with the knowledge, confidence and identity needed to teach computer science effectively. Drawing from empirical data, innovative tools and teacher narratives, the work presents a comprehensive, data-driven framework that bridges the technical and pedagogical dimensions of CS education.

The work introduced a novel keystroke analytics tool – *From Typing to Insights* – which reconstructed students' code progression from raw keystroke logs and visualized the evolution of their programming attempts. It further integrates automatic code execution and error detection mechanisms using unit tests. This has the potential to allow instructors to view the coding process, detect common patterns of errors, identify excessive backtracking or confusion and make timely, and informed intervention decisions. The tool shifts the focus from final submissions to the process-oriented view of student learning, emphasizing the journey of code development over its static end product.

Building this foundation, *TrackIt*, a rule-based detection system that classifies students' programming behaviors into varying levels of struggle by combining keystroke analytics with self-reported survey data was developed. *TrackIt* introduced multiple behavioral indicators such as pause time analysis, deletion frequency, correction patterns and copy-paste behaviors, to flag students at risk of struggling. The tool's validation against self-reported struggle surveys revealed important correlations and gaps between perceived and observed struggle, thereby

offering instructors actionable insights. Moreso, *TracktIt* identified difficult assignments and modules, enabling curriculum refinement and targeted pedagogical support.

While the first two chapters addressed student support, the dissertation switched gears into teacher development, focusing on teachers from rural and underserved communities where disparities in computer science education are most pronounced. Through structured training programs, this work examined how participation influenced teachers' professional identity, confidence, competence and commitment. The results show significant gains in self-efficacy, clearer computing identity formation, and increased dedication to implementing computer science concepts in classrooms. The study illuminated how well-designed professional development programs could bridge the gap in access and readiness to teach computer science.

Extending that line of study, the dissertation provided a deeper understanding of teachers' learning experiences by conducting a thematic analysis of reflective journals. Using both human coding and Large Language Model (LLM) tagging, the study uncovered emotional, cognitive and pedagogical themes, ranging from fear and motivation to instructional challenges and evolving teaching approaches. The findings emphasized that understanding teacher's reflections can yield nuanced perspectives on their growth and reveal insights not easily captured by surveys alone. It validated those qualitative feedback/reflections if rigorously analyzed, proves indispensable for computer science education.

In sum, the four core areas of research provide a coherent narrative, starting from identifying struggling students through their programming interactions, progressing to building intelligent tools for classification and then expanding to empower teachers who are crucial to students' success. Each chapter builds upon the insights of the previous, transitioning from the

level of students to the level of teachers, thus forming a holistic strategy for advancing computer science education.

7.2 Recommendation

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 to 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 apply data 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 explore ways to integrate the tool with existing educational technologies such as embedding keystroke analysis and error reporting features into commonly used learning management systems like Canvas.

7.3 Future Directions

Future enhancements to the keystroke analytics tools (*From Typing to Insights* and *Trackit*) will center around deepening their ability to support students by identifying when they are struggling and offering timely interventions. A key area of development will involve gathering massive datasets (including online courses) and integration of machine learning models that can automatically detect behavioral patterns indicative of struggle. Since the machine learning models rely on extensive feature engineering to build predictive systems that not only detect current difficulty but also anticipate when a student is likely to experience problems, challenges such as false positives and computational efficiency will be addressed in

the development process to ensure the system is both accurate and scalable. Real-time analytics is another major frontier, allowing students and instructors to receive alerts as soon as struggle begins to surface during a programming session. For instance, if a student pauses for a long period, frequently pasting code without prior typing, or shows inconsistent editing behaviors, the system could offer automated feedback or notify instructors for just-in-time interventions.

As the capabilities of *TrackIt* expand, future versions may function more like a virtual coach, offering supportive messages, reflective prompts or encouragement during moments of hesitation. This would transform TrackIt from a retrospective assessment tool to a proactive learning companion. Beyond individualized feedback, connecting the system to broader learning analytics dashboards would allow instructors to examine class-wide trends and determining which assessments or concepts are most challenging. Future multi-modal data sources — such as audio, video, or eye-tracking — could be incorporated to improve the accuracy of struggle detection and offer a more holistic view of student engagement. Broader platform integration is also a crucial direction, particularly embedding the tool within popular development environments like Visual Studio Code, Jupyter Notebooks, and learning management systems such as Canvas. These integrations will make real-time analytics more accessible and actionable for instructors managing large-scale programming courses.

In parallel with student support, future work will continue to build on the success of the teacher development initiatives. The thematic analysis of teachers' reflective journals has already provided meaningful insights into emotional, cognitive and pedagogical dimensions. Teachers have also shared their excitement and fears. Future efforts will build on this to focus on scaling these insights by analyzing larger datasets across broader teacher populations, thereby validating emerging patterns and ensuring their generalizability.

A crucial direction will involve linking these rich reflections to actual classroom practices and student learning outcomes. Doing so will help determine how shifts in teacher identity, confidence, and pedagogical strategies translate into real-world impacts. Longitudinal studies that follow teachers beyond the initial training — tracking how their perspectives, practices, and challenges evolve — could shed light on the sustainability of professional development efforts. Moreover, involving multiple human coders in the thematic analysis will enhance the reliability of findings and establish a stronger benchmark for comparing the outputs from large language models. To optimize LLM-based analysis, future work may experiment with varied prompt strategies to reduce potential bias and deepen thematic consistency.

In sum, by bridging keystroke-based learning analytics with in-depth teacher reflections, this dissertation offers a dual approach to improving computer science education: supporting students in real-time as they learn, and empowering teachers as they grow in their teaching journey.

Chapter 8 - Bibliography

[1] M. Hilbert, “Digital technology and social change: The digital transformation of society from a historical perspective,” *Dialogues Clin. Neurosci.*, vol. 22, no. 2, pp. 189–194, 2020, doi: 10.31887/dcns.2020.22.2/mhilbert.

[2] T. M. I. T. Press and T. Q. Journal, “A Contribution to the Theory of Economic Growth Author (s): Robert M . Solow Source : The Quarterly Journal of Economics , Vol . 70 , No . 1 (Feb ., 1956), pp . 65-94 Published by : The MIT Press Stable URL : http://www.jstor.org/stable/1884513,” *Growth (Lakeland)*, vol. 70, no. 1, pp. 65–94, 2010.

[3] T. W. Swan, “Economic Growth and Capital Accumulation,” *Econ. Rec.*, vol. 32, no. 2, pp. 334–361, 1956.

[4] Simon Kaggwa, Deborah Idowu Akinwolemiwa, Samuel Onimisi Dawodu, Prisca Ugomma Uwaoma, Odunayo Josephine Akindote, and Stephen Osawaru Eloghosa, “Digital transformation and economic development: A review of emerging technologies’ impact on national economies,” *World J. Adv. Res. Rev.*, vol. 20, no. 3, pp. 888–905, 2023, doi: 10.30574/wjarr.2023.20.3.2541.

[5] M. A. Dayioğlu and U. Türker, “Digital transformation for sustainable future-agriculture 4.0: A review,” *Tarim Bilim. Derg.*, vol. 27, no. 4, pp. 373–399, 2021, doi: 10.15832/ankutbd.986431.

[6] J. M. Wing, “Computational thinking,” *Commun. ACM*, vol. 49, no. 3, pp. 33–35, 2006, doi: 10.1145/1118178.1118215.

[7] Y. A. C. González and A. G. V. Muñoz-Repiso, “Development of computational thinking and collaborative learning in kindergarten using programmable educational robots: A teacher training experience,” *ACM Int. Conf. Proceeding Ser.*, vol. Part F1322, pp. 1–6, 2017, doi: 10.1145/3144826.3145353.

[8] F. E. James, N. H. Bean, J. Weese, D. S. Allen, and M. Friend, “From Typing to Insights : An Interactive Code Visualization for Enhanced Student Support Using Keystroke Data,” in *ACM International Conference Proceeding Series*, 2025, pp. 1493–1494. doi: 10.1145/3641555.3705188.

[9] A. Ahadi, S. Lal, R. Lister, and A. Hellas, “Learning Programming, Syntax Errors and Institution-specific Factors,” in *ACM International Conference Proceeding Series*, 2018, pp. 90–96. doi: 10.1145/3160489.3160490.

[10] A. Luxton-Reilly *et al.*, “Developing assessments to determine mastery of programming fundamentals,” *ITiCSE-WGR 2017 - Proc. 2017 ITiCSE Conf. Work. Gr. Reports*, vol. 2018-Janua, pp. 47–69, 2018, doi: 10.1145/3174781.3174784.

[11] M. M. Villamor, “A review on process-oriented approaches for analyzing novice solutions

to programming problems," *Res. Pract. Technol. Enhanc. Learn.*, vol. 15, no. 1, 2020, doi: 10.1186/s41039-020-00130-y.

[12] Z. Fund, D. Court, and B. Kramarski, "Construction and application of an evaluative tool to assess reflection in Teacher-Training courses," *Assess. Eval. High. Educ.*, vol. 27, no. 6, pp. 485–499, 2002, doi: 10.1080/0260293022000020264.

[13] J. R. Warner, R. Torbey, C. L. Fletcher, and L. S. Garbrecht, "Increasing capacity for computer science education in rural areas through a large-scale collective impact model," *SIGCSE 2019 - Proc. 50th ACM Tech. Symp. Comput. Sci. Educ.*, pp. 1157–1163, 2019, doi: 10.1145/3287324.3287418.

[14] M. Wilson, Cameron and Sudol, Leigh Ann and Stephenson, Chris and Stehlik, *Running on Empty: the Failure to Teach K--12 Computer Science in the Digital Age*. New York: The Association for Computing Machinery and the Computer Science Teachers Association, 2020.

[15] S. Grover and R. Pea, "Computational Thinking in K-12: A Review of the State of the Field," *Educ. Res.*, vol. 42, no. 1, pp. 38–43, 2013, doi: 10.3102/0013189X12463051.

[16] M. Başaran, Ş. Metin, and Ö. Faruk, "coding education : A comprehensive review," pp. 20795–20822, 2024.

[17] N. Bean, J. Weese, R. Feldhausen, and R. S. Bell, "Starting from Scratch: Developing a Pre-Service Teacher Training Program in Computational Thinking," *Proc. - Front. Educ. Conf. FIE*, vol. 2014, pp. 1–8, 2015, doi: 10.1109/FIE.2015.7344237.

[18] D. H. Clements and D. F. Gullo, "Effects of computer programming on young children's cognition," *J. Educ. Psychol.*, vol. 76, no. 6, pp. 1051–1058, 1984, doi: 10.1037/0022-0663.76.6.1051.

[19] M. U. Bers, L. Flannery, E. R. Kazakoff, and A. Sullivan, "Computational thinking and tinkering: Exploration of an early childhood robotics curriculum," *Comput. Educ.*, vol. 72, pp. 145–157, 2014, doi: 10.1016/j.compedu.2013.10.020.

[20] N. C. for E. S. (NCES), "NCES Locale Classifications and Criteria." Accessed: Jun. 16, 2024. [Online]. Available: https://nces.ed.gov/surveys/annualreports/topical-studies/locale/definitions

[21] A. D. Cicchinelli, Louis F and Beesley, "Introduction: Current state of the science in rural education research," *Rural Educ. Res. United States State Sci. Emerg. Dir.*, pp. 1–14, 2017, doi: 10.1007/978-3-319-42940-3_1.

[22] C. Broneak and J. Rosato, "Experiences of Rural CS Principles Educators," *2021 Res. Equity Sustain. Particip. Eng. Comput. Technol. RESPECT 2021 - Conf. Proc.*, 2021, doi: 10.1109/RESPECT51740.2021.9620685.

[23] G. I. and G. Inc., "Computer Science Learning: Closing the Gap: Rural and Small Town

School Districts," 2017.

[24] M. desJardins and S. Martin, "CE21--Maryland: The State of Computer Science Education in Maryland High Schools," in *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, ser. SIGCSE '13.*, 2013, pp. 711–716. doi: 10.1145/2445196.2445402.

[25] G. Bernier, D and Stephenson, C and Richardson, D and Chapman, "In Need Of Repair: The State Of K-12 Computer Science Education in California," 2012.

[26] H. H. Hu, C. Heiner, and J. McCarthy, "Deploying exploring computer science statewide," in *SIGCSE 2016 - Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 72–77. doi: 10.1145/2839509.2844622.

[27] M. A. Park and J. Lee, "Rural minorities in computing education: A study of rural schools with no CS/IT courses in Oklahoma," in *Proceedings - 2016 International Conference on Computational Science and Computational Intelligence, CSCI 2016*, IEEE, 2017, pp. 370–373. doi: 10.1109/CSCI.2016.0076.

[28] A. K. Northrup, PE, A. C. Burrows, and T. F. Slater, "Identifying Implementation Challenges for a New Computer Science Curriculum in Rural Western Regions of the United States," *Probl. Educ. 21st Century*, vol. 80, no. 2, pp. 353–370, 2022, doi: 10.33225/pec/22.80.353.

[29] I. Yuliana *et al.*, "Computational Thinking Lesson in Improving Digital Literacy for Rural Area Children via CS Unplugged," *J. Phys. Conf. Ser.*, vol. 1720, no. 1, 2021, doi: 10.1088/1742-6596/1720/1/012009.

[30] G. Siemens, "Learning Analytics: The Emergence of a Discipline," *Am. Behav. Sci.*, vol. 57, no. 10, pp. 1380–1400, 2013, doi: 10.1177/0002764213498851.

[31] L. Tateo, "The Journey of Learning Analytics," *Mind, Cult. Act.*, vol. 26, no. 4, pp. 371–382, 2019, doi: 10.1080/10749039.2019.1686028.

[32] 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 Learn.*, vol. 20, no. 2, pp. 13–29, 2016.

[33] R. K. Veluri *et al.*, "Learning analytics using deep learning techniques for efficiently managing educational institutes," *Mater. Today Proc.*, vol. 51, pp. 2317–2320, 2022, doi: 10.1016/j.matpr.2021.11.416.

[34] N. Nistor and Á. Hernández-Garcíac, "What types of data are used in learning analytics? An overview of six cases," *Comput. Human Behav.*, vol. 89, no. July, pp. 335–338, 2018, doi: 10.1016/j.chb.2018.07.038.

[35] P. Blikstein, "Multimodal learning analytics," *ACM Int. Conf. Proceeding Ser.*, pp. 102–106, 2013, doi: 10.1145/2460296.2460316.

[36] S. Dawson, S. Joksimovic, O. Poquet, and G. Siemens, “Increasing the impact of learning analytics,” *ACM Int. Conf. Proceeding Ser.*, no. March, pp. 446–455, 2019, doi: 10.1145/3303772.3303784.

[37] Y. S. Tsai and D. Gasevic, “Learning analytics in higher education - Challenges and policies: A review of eight learning analytics policies,” *ACM Int. Conf. Proceeding Ser.*, pp. 233–242, 2017, doi: 10.1145/3027385.3027400.

[38] R. Moskovitch *et al.*, “Identity theft, computers and behavioral biometrics,” *2009 IEEE Int. Conf. Intell. Secur. Informatics, ISI 2009*, pp. 155–160, 2009, doi: 10.1109/ISI.2009.5137288.

[39] H. Crawford, “Keystroke dynamics: Characteristics and opportunities,” *PST 2010 2010 8th Int. Conf. Privacy, Secur. Trust*, pp. 205–212, 2010, doi: 10.1109/PST.2010.5593258.

[40] T. Nakada and M. Miura, “Extracting typing game keystroke patterns as potential indicators of programming aptitude,” *Front. Comput. Sci.*, vol. 6, no. 2001, 2024, doi: 10.3389/fcomp.2024.1412458.

[41] J. Leinonen, “Keystroke Data in Programming Courses Juho Leinonen,” University of Helsink, 2019.

[42] R. Shrestha, “PROGRAMMING PROCESS, PATTERNS AND BEHAVIORS: INSIGHTS FROM KEYSTROKE ANALYSIS OF CS1 STUDENTS,” UTAH STATE UNIVERSITY, 2022.

[43] J. Edwards, K. Hart, and R. Shrestha, “Review of CSEDM Data and Introduction of Two Public CS1 Keystroke Datasets,” *J. Educ. Data Min.*, vol. 15, no. 1, pp. 1–31, 2023, doi: 10.5281/zenodo.7646659.

[44] R. Shrestha, J. Leinonen, A. Zavgorodniaia, A. Hellas, and J. Edwards, “Pausing While Programming: Insights From Keystroke Analysis,” in *IEEE/ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSESEET*, Pittsburgh, PA, USA. ACM, New York, NY, USA, 12, 2022, pp. 187–198. doi: 10.1145/3510456.3514146.

[45] Z. Pullar-Strecker, F. D. Pereira, P. Denny, A. Luxton-Reilly, and J. Leinonen, “G is for Generalisation: Predicting Student Success from Keystrokes,” *SIGCSE 2023 - Proc. 54th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, pp. 1028–1034, 2023, doi: 10.1145/3545945.3569824.

[46] B. U. Cowley, A. Hellas, P. Ihantola, J. Leinonen, and M. Spape, “Seeking flow from fine-grained log data,” in *ACM 44th International Conference on Software Engineering: Software Engineering Education and Training (ICSESEET)*, Pittsburgh, PA, USA., 2022, pp. 247–253. doi: 10.1145/3510456.3514138.

[47] J. Nakamura and M. Csikszentmihalyi, “The Concept of FLow,” in *Handbook of Positive Psychology*, New York, 2002, ch. The Concep, pp. 89–105. doi:

10.1002/9780470172698.ch19.

[48] A. Kolakowska, “A review of emotion recognition methods based on keystroke dynamics and mouse movements,” *2013 6th Int. Conf. Hum. Syst. Interact. HSI 2013*, pp. 548–555, 2013, doi: 10.1109/HSI.2013.6577879.

[49] C. Epp, M. Lippold, and R. L. Mandryk, “Identifying emotional states using keystroke dynamics,” in *Conference on Human Factors in Computing Systems - Proceedings*, Vancouver Canada, 2011, pp. 715–724. doi: 10.1145/1978942.1979046.

[50] R. Bixler and S. D’Mello, “Detecting boredom and engagement during writing with keystroke analysis, task appraisals, and stable traits,” in *International Conference on Intelligent User Interfaces, Proceedings IUI*, 2013, pp. 225–233. doi: 10.1145/2449396.2449426.

[51] J. Bennedsen and M. E. Caspersen, “Failure rates in introductory programming,” in *ACM SIGCSE Bulletin*, 2007, pp. 32–36. doi: 10.1145/1272848.1272879.

[52] S. Bergin and R. Reilly, “Programming,” 2005, pp. 411–415. doi: 10.1145/1047344.1047480.

[53] P. Kinnunen and B. Simon, “Experiencing programming assignments in CS1: The emotional toll,” *ICER’10 - Proc. Int. Comput. Educ. Res. Work.*, pp. 77–85, 2010, doi: 10.1145/1839594.1839609.

[54] N. Islam, G. Shafi Sheikh, R. Fatima, and F. Alvi, “A Study of Difficulties of Students in Learning Programming,” *J. Educ. Soc. Sci.*, vol. 7, no. 2, pp. 38–46, 2019, doi: 10.20547/jess0721907203.

[55] K. Arakawa, Q. Hao, T. Greer, L. Ding, C. D. Hundhausen, and A. Peterson, “In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments,” in *SIGCSE 2021 - Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, 2021, pp. 467–473. doi: 10.1145/3408877.3432357.

[56] J. Gorson, N. LaGrassa, C. H. Hu, E. Lee, A. M. Robinson, and E. O’Rourke, “An Approach for Detecting Student Perceptions of the Programming Experience from Interaction Log Data,” in *International Conference on Artificial Intelligence in Education*, 2021, pp. 150–164. doi: 10.1007/978-3-030-78292-4_13.

[57] A. Altadmri and N. C. C. Brown, “37 million compilations: Investigating novice programming mistakes in large-scale student data,” *SIGCSE 2015 - Proc. 46th ACM Tech. Symp. Comput. Sci. Educ.*, pp. 522–527, 2015, doi: 10.1145/2676723.2677258.

[58] L. Lundberg, “Classifying Struggling Students Through Error-Message Analysis in Programming Education A Swedish Case Study Classifying Struggling Students through Error-Messages Analysis in Programming Education : A Swedish Case Study,” A Swedish Case Study, 2024.

[59] C. Geng, W. Xu, Y. Xu, B. Pientka, and X. Si, “Identifying Different Student Clusters in Functional Programming Assignments: From Quick Learners to Struggling Students,” in *SIGCSE 2023 - Proceedings of the 54th ACM Technical Symposium on Computer Science Education*, 2023, pp. 750–756. doi: 10.1145/3545945.3569882.

[60] F. E. James, N. H. Bean, J. Weese, D. S. Allen, and M. Friend, “From Typing to Insights : An Interactive Code Visualization for Enhanced Student Support Using Keystroke Data,” in *SIGCSETS 2025: Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 2*, pp. 1493–1494. doi: 10.1145/3641555.3705188.

[61] G. Silva De Oliveira, Z. Gao, S. Heckman, and C. Lynch, “Exploring Novice Programmers’ Testing Behavior: A First Step to Define Coding Struggle,” *SIGCSE 2024 - Proc. 55th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 1, pp. 1251–1257, 2024, doi: 10.1145/3626252.3630851.

[62] T. T. Benyamin, A. Limke, H. Reichert, R. Qualls, T. Price, and ..., “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 …*, 2022. doi: 10.5281/zenodo.6983260.

[63] D. McCall and M. Kölling, “Meaningful categorisation of novice programmer errors,” *Proc. - Front. Educ. Conf. FIE*, vol. 2015-Febru, no. February, pp. 1–8, 2015, doi: 10.1109/FIE.2014.7044420.

[64] K. Castro-Wunsch, A. Ahadi, and A. Petersen, “Evaluating neural networks as a method for identifying students in need of assistance,” in *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, 2017, pp. 111–116. doi: 10.1145/3017680.3017792.

[65] D. Lokhande, “Deep neural network in prediction of student performance,” *GRADIVA Rev. J.*, no. February, 2023.

[66] L. Vives *et al.*, “Prediction of Students’ Academic Performance in the Programming Fundamentals Course Using Long Short-Term Memory Neural Networks,” *IEEE Access*, vol. 12, no. November 2023, pp. 5882–5898, 2024, doi: 10.1109/ACCESS.2024.3350169.

[67] M. Moresi, M. J. Gómez, and L. Benotti, “Predicting Students’ Difficulties from a Piece of Code,” *IEEE Trans. Learn. Technol.*, vol. 14, no. 3, pp. 386–399, 2021, doi: 10.1109/TLT.2021.3092998.

[68] A. Prasanth and H. Alqahtani, “Predictive Modeling of Student Behavior for Early Dropout Detection in Universities using Machine Learning Techniques,” in *International Conference on Engineering Technologies and Applied Sciences: Shaping the Future of Technology through Smart Computing and Engineering, ICETAS 2023*, 2023. doi: 10.1109/ICETAS59148.2023.10346531.

[69] Z. Ullah, A. Lajis, M. Jamjoom, A. H. Altalhi, J. Shah, and F. Saleem, “A rule-based method for cognitive competency assessment in computer programming using bloom’s

taxonomy," *IEEE Access*, vol. 7, pp. 64663–64675, 2019, doi: 10.1109/ACCESS.2019.2916979.

[70] and D. R. K. B. S. Bloom, M. D. Englehard, E. J. Furst, W. H. Hill, *Taxonomy of Educational Objectives: The Classi_cation of Educational Goals: Handbook I Cognitive Domain.* Harlow, U.K.: Longmans, Green, 1956.

[71] A. Shirafuji, T. Matsumoto, M. F. Ibne Amin, and Y. Watanobe, "Rule-Based Error Classification for Analyzing Differences in Frequent Errors," *2023 IEEE Int. Conf. Teaching, Assess. Learn. Eng. TALE 2023 - Conf. Proc.*, pp. 1–7, 2023, doi: 10.1109/TALE56641.2023.10398341.

[72] U. K. R. Md. Mostafizer Rahman, Yutaka Watanobe and and K. Nakamura, "A Novel Rule-Based Online Judge Recommender System to Promote Computer Programming Education," in *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*, 2021, pp. 15–27.

[73] L. Cagliero, L. Canale, L. Farinetti, E. Baralis, and E. Venuto, "Predicting student academic performance by means of associative classification," *Appl. Sci.*, vol. 11, no. 4, pp. 1–22, 2021, doi: 10.3390/app11041420.

[74] J. Sachs, "Teacher education and the development of professional identity: Learning to be a teacher," *Connect. Policy Pract. Challenges Teach. Learn. Sch. Univ.*, pp. 5–21, 2005, doi: 10.4324/9780203012529.

[75] L. Ni and M. Guzdial, "Who AM I?: Understanding high school computer science teachers' professional identity," in *SIGCSE'12 - Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 2012, pp. 499–504. doi: 10.1145/2157136.2157283.

[76] J. W. Reid *et al.*, "Perceived network bridging influences the career commitment decisions of early career teachers," *Int. J. STEM Educ.*, vol. 10, no. 1, 2023, doi: 10.1186/s40594-023-00408-9.

[77] C. R. Rodgers and K. H. Scott, "The development of the personal self and professional identity in learning to teach," *Handb. Res. Teach. Educ. Endur. Quest. Chang. Context. Third Ed.*, pp. 732–755, 2008, doi: 10.4324/9780203938690-85.

[78] L. Ni, T. Mcklin, H. Hao, J. Baskin, J. Bohrer, and Y. Tian, "Understanding Professional Identity of Computer Science Teachers: Design of the Computer Science Teacher Identity Survey," in *ICER 2021 - Proceedings of the 17th ACM Conference on International Computing Education Research*, 2021, pp. 281–293. doi: 10.1145/3446871.3469766.

[79] J. Everson and A. J. Ko, "'I would be afraid to be a bad CS teacher': Factors Influencing Participation in Pre-Service Secondary CS Teacher Education," in *ICER 2022 - Proceedings of the 2022 ACM Conference on International Computing Education Research*, 2022, pp. 237–246. doi: 10.1145/3501385.3543966.

[80] A. Joshi, A. Jain, E. Covelli, J. H. Yeh, and T. Andersen, “A Sustainable Model for High-School Teacher Preparation in Computer Science,” in *Proceedings - Frontiers in Education Conference, FIE*, IEEE, 2019, pp. 1–9. doi: 10.1109/FIE43999.2019.9028638.

[81] M. A. Huett, Kim C and Varga, “Building Pre-Service Teacher Interest in Computer Science Education through Mentoring Experiences(Abstract Only),” in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 690–690. doi: 10.1145/2839509.2850547.

[82] S. Morrissey *et al.*, “Designing a Program to Develop Computer Science Master Teachers for an Underserved Rural Area,” *J. STEM Teach. Educ.*, vol. 58, no. 1, 2023, doi: 10.61403/2158-6594.1488.

[83] R. Vivian *et al.*, “An International Pilot Study of K-12 Teachers’ Computer Science Self-Esteem,” in *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, 2020, pp. 117–123. doi: 10.1145/3341525.3387418.

[84] C. McInerney, C. Exton, and M. Hinchey, “A study of high school computer science teacher confidence levels,” in *Proceedings of the 15th Workshop on Primary and Secondary Computing Education, ser. WiPSCE ’20.*, 2020, pp. 2019–2020. doi: 10.1145/3421590.3421614.

[85] K. Hamlen, N. Sridhar, L. Bievenue, D. K. Jackson, and A. Lalwani, “Effects of teacher training in a computer science principles curriculum on teacher and student skills, confidence, and beliefs,” in *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 741–746. doi: 10.1145/3159450.3159496.

[86] S. Hennessy, K. Ruthven, and S. Brindley, *Teacher perspectives on integrating ICT into subject teaching: Commitment, constraints, caution, and change*, vol. 37, no. 2. 2005. doi: 10.1080/0022027032000276961.

[87] H. Ding, “Teachers ’ attitudes toward computer technology and factors influencing their attitudes and adoption of computer technology in classroom instruction : a literature review,” 1997.

[88] J. Goode, “Connecting K-16 curriculum & policy: Making computer science engaging, accessible, and hospitable for underrepresented students,” *SIGCSE’10 - Proc. 41st ACM Tech. Symp. Comput. Sci. Educ.*, pp. 22–26, 2010, doi: 10.1145/1734263.1734272.

[89] J. Margolis, R. Estrella, J. Goode, J. J. Holme, and K. Nao, “Stuck in the Shallow End | The MIT Press,” in *Stuck in The Shallow End*, MIT Press, 2008, p. 216.

[90] J. Goode, “If you build teachers, will students come? The role of teachers in broadening computer science learning for Urban youth,” *J. Educ. Comput. Res.*, vol. 36, no. 1, pp. 65–88, 2007, doi: 10.2190/2102-5G77-QL77-5506.

[91] L. Ni, G. Bausch, and R. Benjamin, “Computer science teacher professional development

and professional learning communities: a review of the research literature," *Comput. Sci. Educ.*, vol. 33, no. 1, pp. 29–60, 2023, doi: 10.1080/08993408.2021.1993666.

[92] K. M. Milbrath Ying-Chen, "Computer Technology Training for Prospective Teachers : Computer Attitudes and Perceived," *J. Inf. Techology Teach. Educ.*, vol. 8, no. February 2000, pp. 373–396, 2000.

[93] V. Braun and V. Clarke, "Using Thematic Anaysis in Psychology," *Qual. Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006.

[94] N. Erdemir and S. Yeşilçınar, "Reflective practices in micro teaching from the perspective of preservice teachers: teacher feedback, peer feedback and self-reflection," *Reflective Pract.*, vol. 22, no. 6, pp. 766–781, 2021, doi: 10.1080/14623943.2021.1968818.

[95] A. Ghosh and A. Verma, "Thematic analysis of reflective peer feedback in programming-heavy engineering courses," *Proc. - Front. Educ. Conf. FIE*, vol. 2021-Octob, pp. 1–9, 2021, doi: 10.1109/FIE49875.2021.9637374.

[96] M. Celepkolu and K. E. Boyer, "Thematic analysis of students' reflections on pair programming in CS1," in *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 771–776. doi: 10.1145/3159450.3159516.

[97] R. P. Gauthier and J. R. Wallace, "The Computational Thematic Analysis Toolkit," in *Proceedings of the ACM on Human-Computer Interaction*, 2022, pp. 1–15. doi: 10.1145/3492844.

[98] V. Lytvyn *et al.*, "Identifying Textual Content Based on Thematic Analysis of Similar Texts in Big Data," in *International Scientific and Technical Conference on Computer Sciences and Information Technologies*, IEEE, 2019, pp. 84–91. doi: 10.1109/STC-CSIT.2018.8929808.

[99] T. Rahman, J. Nwokeji, R. Matovu, S. Frezza, H. Sugnanam, and A. Pisolkar, "Analyzing Competences in Software Testing: Combining Thematic Analysis with Natural Language Processing (NLP)," in *Proceedings - Frontiers in Education Conference, FIE*, IEEE, 2021, pp. 1–9. doi: 10.1109/FIE49875.2021.9637220.

[100] F. E. James, R. Feldhausen, N. H. Bean, J. Weese, D. Allen, and M. Friend, "From Typing to Insights: An Interactive Code Visualization Tool for Enhanced Student Support Using Keystroke Data," in *132nd American Society for Engineering Education (ASEE)*, 2025.

[101] R. N. Robins Anthony, Rountree Janet, "Learning and Teaching Programming: A Review and Discussion," *Comput. Sci. Educ.*, vol. 13, no. 2, pp. 137–172, 2003.

[102] D. McCall and M. Kölling, "A new look at novice programmer errors," *ACM Trans. Comput. Educ.*, vol. 19, no. 4, 2019, doi: 10.1145/3335814.

[103] L. Murphy, G. Lewandowski, R. McCauley, B. Simon, L. Thomas, and C. Zander,

"Debugging: The Good, the Bad, and the Quirky – a Qualitative Analysis of Novices' Strategies," in *ACM SIGCSE Bulletin*, 2008, pp. 163–167. doi: 10.1145/1352322.1352191.

[104] J. C. Paiva, J. P. Leal, and Á. Figueira, "Automated Assessment in Computer Science Education: A State-of-the-Art Review," *ACM Trans. Comput. Educ.*, vol. 22, no. 3, 2022, doi: 10.1145/3513140.

[105] J. P. Munson and J. P. Zitovsky, "Models for early identification of struggling novice programmers," *SIGCSE 2018 - Proc. 49th ACM Tech. Symp. Comput. Sci. Educ.*, vol. 2018-January, pp. 699–704, 2018, doi: 10.1145/3159450.3159476.

[106] M. R. Shinn, "Identifying students at risk, monitoring performance, and determining eligibility within response to intervention: Research on educational need and benefit from academic intervention," *School Psych. Rev.*, vol. 36, no. 4, pp. 601–617, 2007, doi: 10.1080/02796015.2007.12087920.

[107] M. C. Jadud, "Methods and tools for exploring novice compilation behaviour," in *ICER 2006 - Proceedings of the 2nd International Computing Education Research Workshop*, 2006, pp. 73–84. doi: 10.1145/1151588.1151600.

[108] C. Inc, "Codio - The Hands-On Platform for Computing & Tech Skills Education." [Online]. Available: https://www.codio.com/

[109] U. DigitalCommons, U. All Graduate Theses, and R. Shrestha, "Programming Process, Patterns and Behaviors: Insights from Programming Process, Patterns and Behaviors: Insights from Keystroke Analysis of CS1 Students Keystroke Analysis of CS1 Students," Utah State University, 2022.

[110] L. Van Waes and P. J. Schellens, "Writing profiles: The effect of the writing mode on pausing and revision patterns of experienced writers," *J. Pragmat.*, vol. 35, no. 6, pp. 829–853, 2003, doi: 10.1016/S0378-2166(02)00121-2.

[111] T. Olive, R. A. Alves, and S. L. Castro, "Cognitive processes in writing during pause and execution periods," *Eur. J. Cogn. Psychol.*, vol. 21, no. 5, pp. 758–785, 2009, doi: 10.1080/09541440802079850.

[112] R. A. Alves, S. L. Castro, L. De Sousa, and S. Strömqvist, "Influence of typing skill on pause-execution cycles in written composition," *Writ. Cogn. Res. Appl.*, vol. 20, pp. 54–65, 2007, doi: 10.1163/9781849508223_005.

[113] I. E. Allen and C. A. Seaman, "Likert scales and data analyses," *Qual. Prog.*, vol. 40, no. 7, pp. 64–65, 2007.

[114] C. J.A., "A coefficient of agreement for nominal scales," *Educ. Psychol. Meas.*, vol. XX, no. 1, pp. 37–46, 1960.

[115] D. Marasini, P. Quatto, and E. Ripamonti, "Assessing the inter-rater agreement for ordinal

data through weighted indexes," *Stat. Methods Med. Res.*, vol. 25, no. 6, pp. 2611–2633, 2016, doi: 10.1177/0962280214529560.

[116] F. E. James, W. Joshua, F. Russel, B. Nathan, A. David, and F. Michelle, "Expanding Computer Science Education in Rural Areas: Impact of Teacher Training on Teachers' Identity, Commitment, Confidence and Competence," in *132nd American Society for Engineering Education (ASEE)*, 2025.

[117] T. Camp *et al.*, "Generation CS: The growth of computer science," *ACM Inroads*, vol. 8, no. 2, pp. 44–50, 2017, doi: 10.1145/3084362.

[118] J. R. Warner, J. Childs, C. L. Fletcher, N. D. Martin, and M. Kennedy, "Quantifying Disparities in Computing Education: Access, Participation, and Intersectionality," *SIGCSE 2021 - Proc. 52nd ACM Tech. Symp. Comput. Sci. Educ.*, pp. 619–625, 2021, doi: 10.1145/3408877.3432392.

[119] Code.org, "2023 State of Computer Science Education," 2023.

[120] K. Margolis, J. and Estrella, R. and Goode, J. and Holme, J.J. and Nao, *Stuck in the Shallow End, updated edition: Education, Race, and Computing*. MIT Press, 2017.

[121] K. Aguar, H. R. Arabnia, J. B. Gutierrez, W. D. Potter, and T. R. Taha, "Making CS inclusive: An overview of efforts to expand and diversify CS education," in *Proceedings - 2016 International Conference on Computational Science and Computational Intelligence, CSCI 2016*, IEEE, 2017, pp. 321–326. doi: 10.1109/CSCI.2016.0067.

[122] C. Huett, Kim C. and Westine, "Using Needs Assessment to Inform a Rural School District's Efforts to Expand Access to Computer Science Education: (Abstract Only)," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, New York NY USA: Association for Computing Machinery, 2018, p. 1097. doi: 10.1145/3159450.3162293.

[123] J. Yadav, Aman and DeLyser, Leigh Ann and Kafai, Yasmin and Guzdial, Mark and Goode, "Building and expanding the capacity of schools of education to prepare and support teachers to teach computer science," in *Preparing Pre-Service Teachers to Teach Computer Science: Models, Practices, and Policies. C. Mouza, A. Yadav, and A. Ottenbreit-Leftwich, eds. Information Age*, 2021, ch. Preparing.

[124] J. Goode, M. Skorodinsky, J. Hubbard, and J. Hook, "Computer Science for Equity: Teacher Education, Agency, and Statewide Reform," *Front. Educ.*, vol. 4, no. January, pp. 1–12, 2020, doi: 10.3389/feduc.2019.00162.

[125] W. Pewkam and S. Chamrat, "Pre-Service Teacher Training Program of STEM-based Activities in Computing Science to Develop Computational Thinking," *Informatics Educ.*, vol. 21, no. 2, pp. 311–329, 2022, doi: 10.15388/infedu.2022.09.

[126] J. Liu, E. P. Hasson, Z. D. Barnett, and P. Zhang, "A survey on computer science K-12 outreach: Teacher training programs," *Proc. - Front. Educ. Conf. FIE*, pp. T4F-1-T4F-6,

2011, doi: 10.1109/FIE.2011.6143111.

[127] K. P. S. Goodpaster, O. A. Adedokun, and G. C. Weaver, “Teachers’ Perceptions of Rural STEM Teaching: Implications for Rural Teacher Retention,” *Rural Educ.*, vol. 33, no. 3, pp. 9–22, 2018, doi: 10.35608/ruraled.v33i3.408.

[128] J. Simmonds, F. J. Gutierrez, C. Casanova, C. Sotomayor, and N. Hitschfeld, “A teacher workshop for introducing computational thinking in rural and vulnerable environments,” *SIGCSE 2019 - Proc. 50th ACM Tech. Symp. Comput. Sci. Educ.*, pp. 1143–1149, 2019, doi: 10.1145/3287324.3287456.

[129] I. A. Lee, M. P. Dombrowski, and E. Angel, “Preparing STEM teachers to offer New Mexico computer science for all,” *Proc. Conf. Integr. Technol. into Comput. Sci. Educ. ITiCSE*, pp. 363–368, 2017, doi: 10.1145/3017680.3017719.

[130] C. Leonard, Jacqueline and Mitchell, Monica and Barnes-Johnson, Joy and Unertl, Adrienne and Outka-Hill, Jill and Robinson, Roland and Hester-Croff, “Preparing teachers to engage rural students in computational thinking through robotics, game design, and culturally responsive teaching,” *J. Teach. Educ.*, vol. 69, no. 4, pp. 386–407, 2018, doi: 10.1177/0022487117732317.

[131] J. Mahadeo, Z. Hazari, and G. Potvin, “Developing a computing identity framework: Understanding computer science and information technology career choice,” *ACM Trans. Comput. Educ.*, vol. 20, no. 1, pp. 7–714, 2020, doi: 10.1145/3365571.

[132] C. B. Oser, J. Strickland, E. J. Batty, E. Pullen, and M. Staton, “The rural identity scale: Development and validation,” *J. Rural Heal.*, vol. 38, no. 1, pp. 303–310, 2022, doi: 10.1111/jrh.12563.

[133] C. S. Dweck, *Self-theories: Their role in motivation, personality and development*. Pychology Press, 2013. doi: 10.4324/978131578304.

[134] C. . Dweck, *Mindset: The New Psychology of Success*. Random House Publishing Group, 2006.

[135] Codio Inc., “Codio: The Hands-On Platform for Computing and Tech Skills Education.” [Online]. Available: https://www.codio.com

[136] D. K. Lee, “Alternatives to P value: Confidence interval and effect size,” *Korean J. Anesthesiol.*, vol. 69, no. 6, pp. 555–562, 2016, doi: 10.4097/kjae.2016.69.6.555.

[137] I. Hedges, Larry V and Olkin, *Statistical methods for meta-analysis*. Academic Press, 2014.

[138] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Routledge, 2013.

[139] B. Avalos, “Teacher professional development in Teaching and Teacher Education over ten years,” *Teach. Teach. Educ.*, vol. 27, no. 1, pp. 10–20, 2011, doi:

10.1016/j.tate.2010.08.007.

[140] D. Clarke and H. Hollingsworth, “Elaborating a model of teacher development,” *Teach. Teach. Educ.*, vol. 18, no. 2002, pp. 947–967, 2002.

[141] V. D. Opfer and D. Pedder, “Conceptualizing teacher professional learning,” *Rev. Educ. Res.*, vol. 81, no. 3, pp. 376–407, 2011, doi: 10.3102/0034654311413609.

[142] A. Yadav, S. Gretter, S. Hambrusch, and P. Sands, “Expanding computer science education in schools: understanding teacher experiences and challenges,” *Comput. Sci. Educ.*, vol. 26, no. 4, pp. 235–254, 2017, doi: 10.1080/08993408.2016.1257418.

[143] C. Mouza, D. Codding, and L. Pollock, “Investigating the impact of research-based professional development on teacher learning and classroom practice: Findings from computer science education,” *Comput. Educ.*, vol. 186, no. December 2021, p. 104530, 2022, doi: 10.1016/j.compedu.2022.104530.

[144] C. Brandt, “Integrating feedback and reflection in teacher preparation,” *ELT J.*, vol. 62, no. 1, pp. 37–46, 2008, doi: 10.1093/elt/ccm076.

[145] E. Ene and C. Riddlebarger, “Intensive Reflection in Teacher Training: What is it Good For?,” *J. Acad. Writ.*, vol. 5, no. Schön 1983, pp. 157–168, 2015, doi: 10.18552/joaw.v5i1.160.

[146] N. Yurtseven and S. Altun, “The Role of Self-Reflection and Peer Review in Curriculum-focused Professional Development for Teachers,” *H. U. J. Educ.*, vol. 33, no. 1, pp. 207–228, 2018, doi: 10.16986/HUJE.2017030461.

[147] R. M. Ryan and E. L. Deci, “Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions,” *Contemp. Educ. Psychol.*, vol. 25, no. 1, pp. 54–67, 2000, doi: 10.1006/ceps.1999.1020.

[148] H. H. Cohen David, *Learning Policy: When State Education Reform Works*. Yale University Press, 2008.

[149] M. Darling-Hammond, L, Hyler, M. E., Gardner, “Effective Teacher Professional Development.,” 2017.

[150] L. Chin-Yew, “Text Summarization branches out,” 2004, ch. ROUGE: A P, pp. 74–81.

Appendix A - Assignment Feedback Survey

1. On a scale of 1 to 5, how would you rate the difficulty level of this assignment?

   (1 = Very Easy, 5 = Very Difficult)

   - (•) 1
   - ( ) 2
   - ( ) 3
   - ( ) 4
   - ( ) 5

2. Approximately how much time in total did you spend working on this assignment?

   (Include all time spent thinking, coding, and revising)

3. Did you struggle with this assignment?

   - ( ) Yes
   - ( ) No

4. If you answered "Yes" to the previous question, please specify what part of the assignment you struggled with the most.

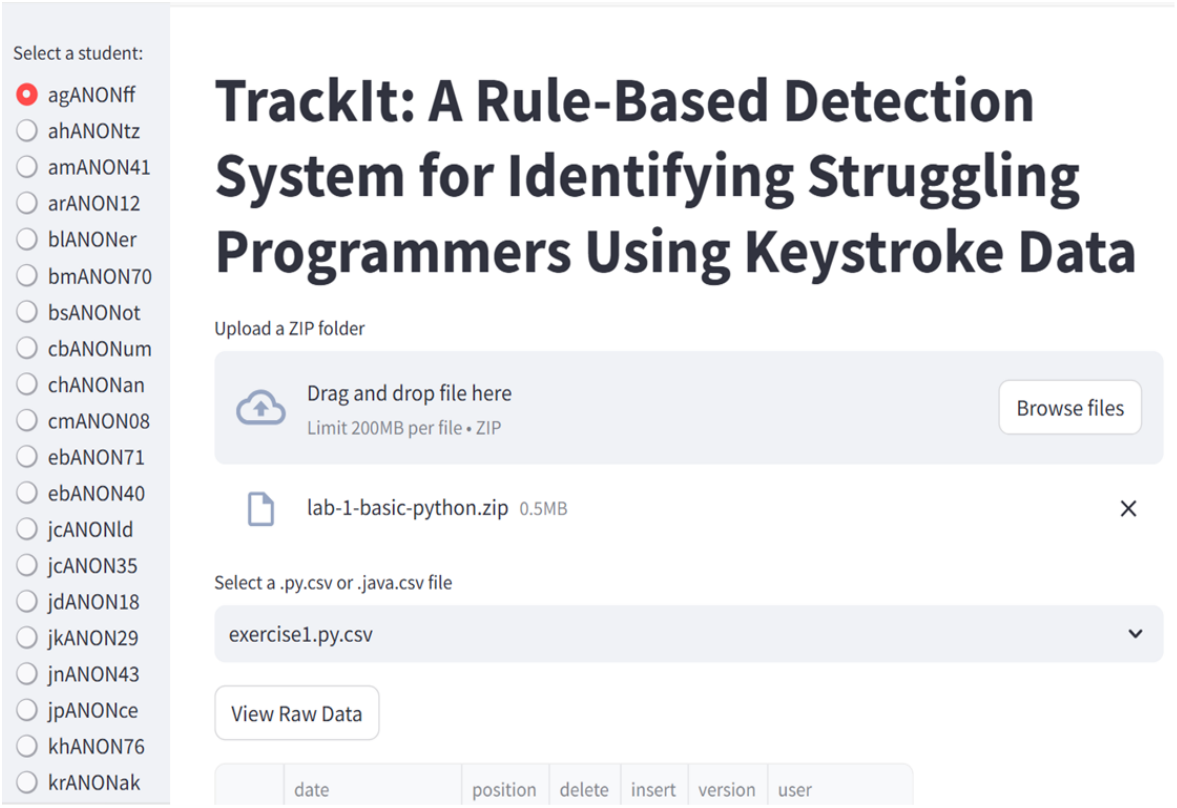
5. On a scale of 1 to 5, how confident did you feel while completing this assignment?

   (1 = Not Confident at All, 5 = Very Confident)

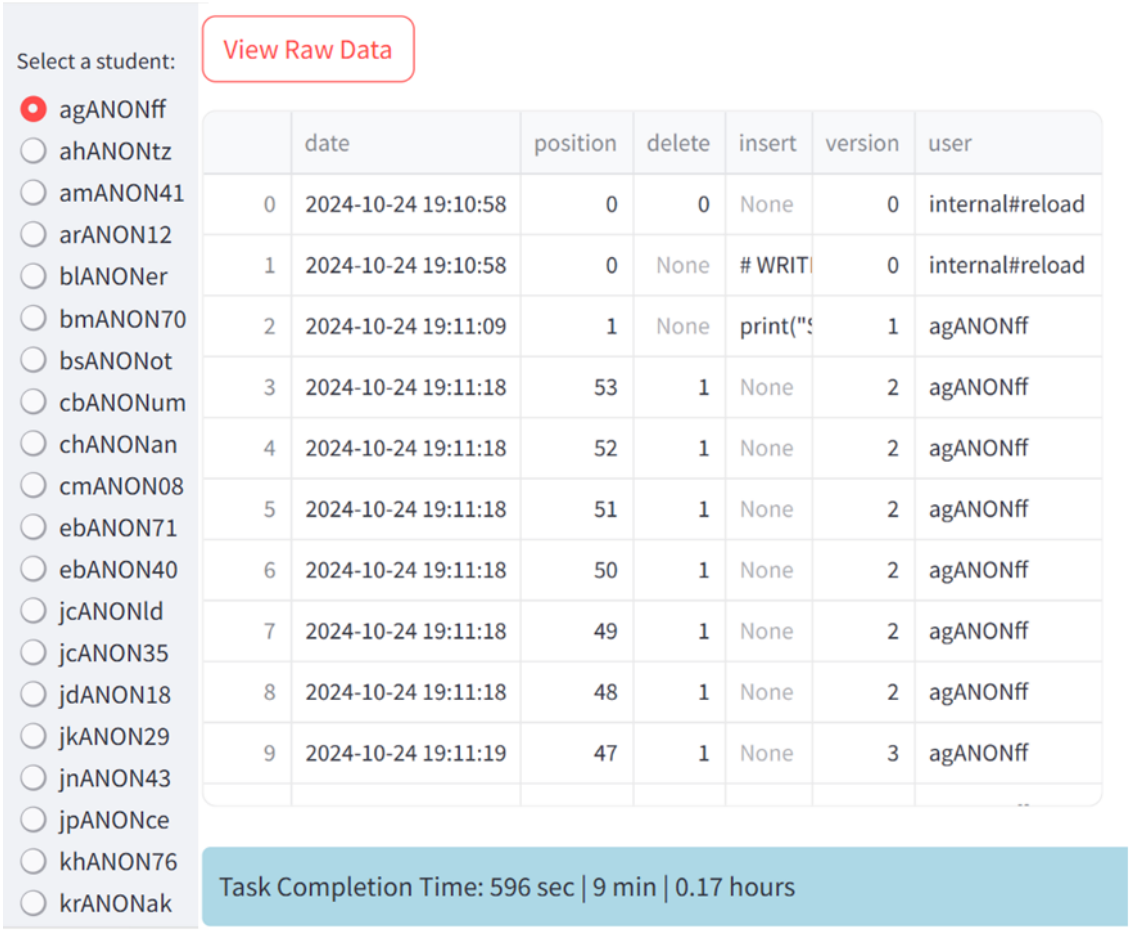
   - (•) 1
   - ( ) 2
   - ( ) 3
   - ( ) 4
   - ( ) 5

Appendix B - *TrackIt* Features/Functionalities

TrackIt: File Upload



TrackIt: Raw Keystroke Data Display and Total Coding Time



Select a student:

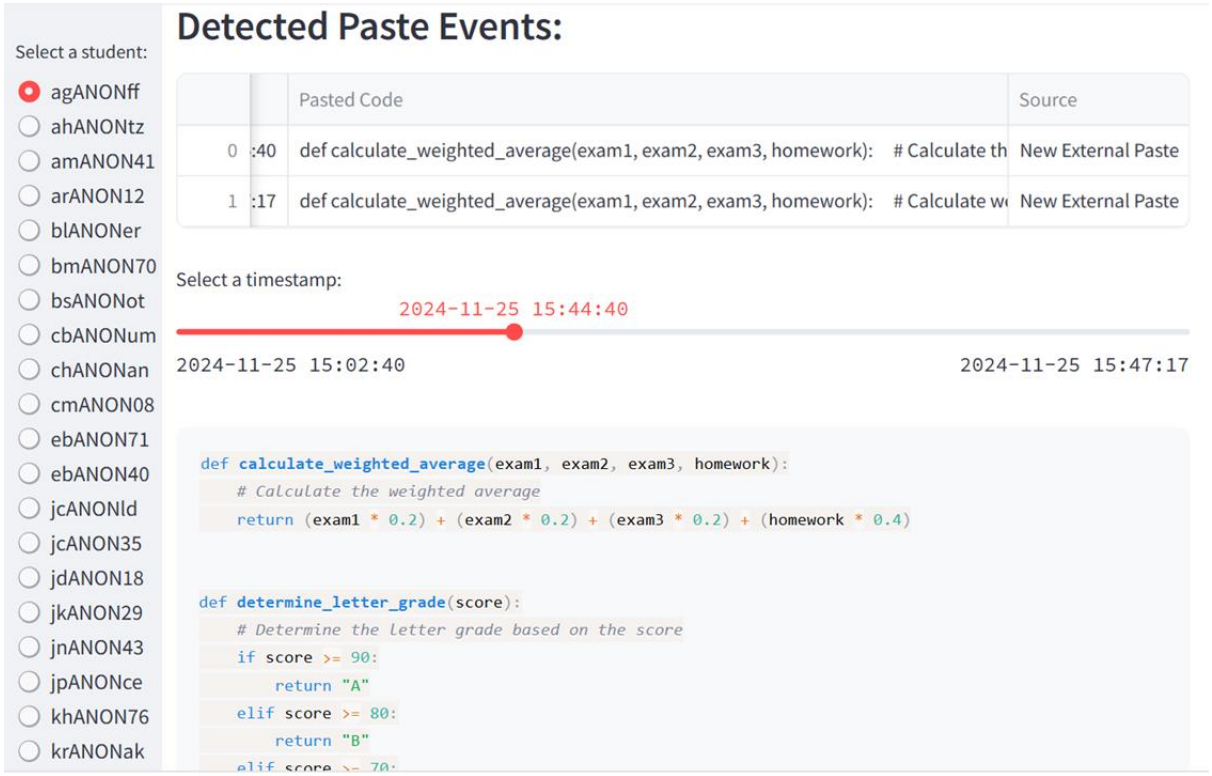
- agANONff
- ahANONtz
- amANON41
- arANON12
- blANONer
- bmANON70
- bsANONot
- cbANONum
- chANONan
- cmANON08
- ebANON71
- ebANON40
- jcANONld
- jcANON35
- jdANON18
- jkANON29
- jnANON43
- jpANONce
- khANON76
- krANONak

View Raw Data

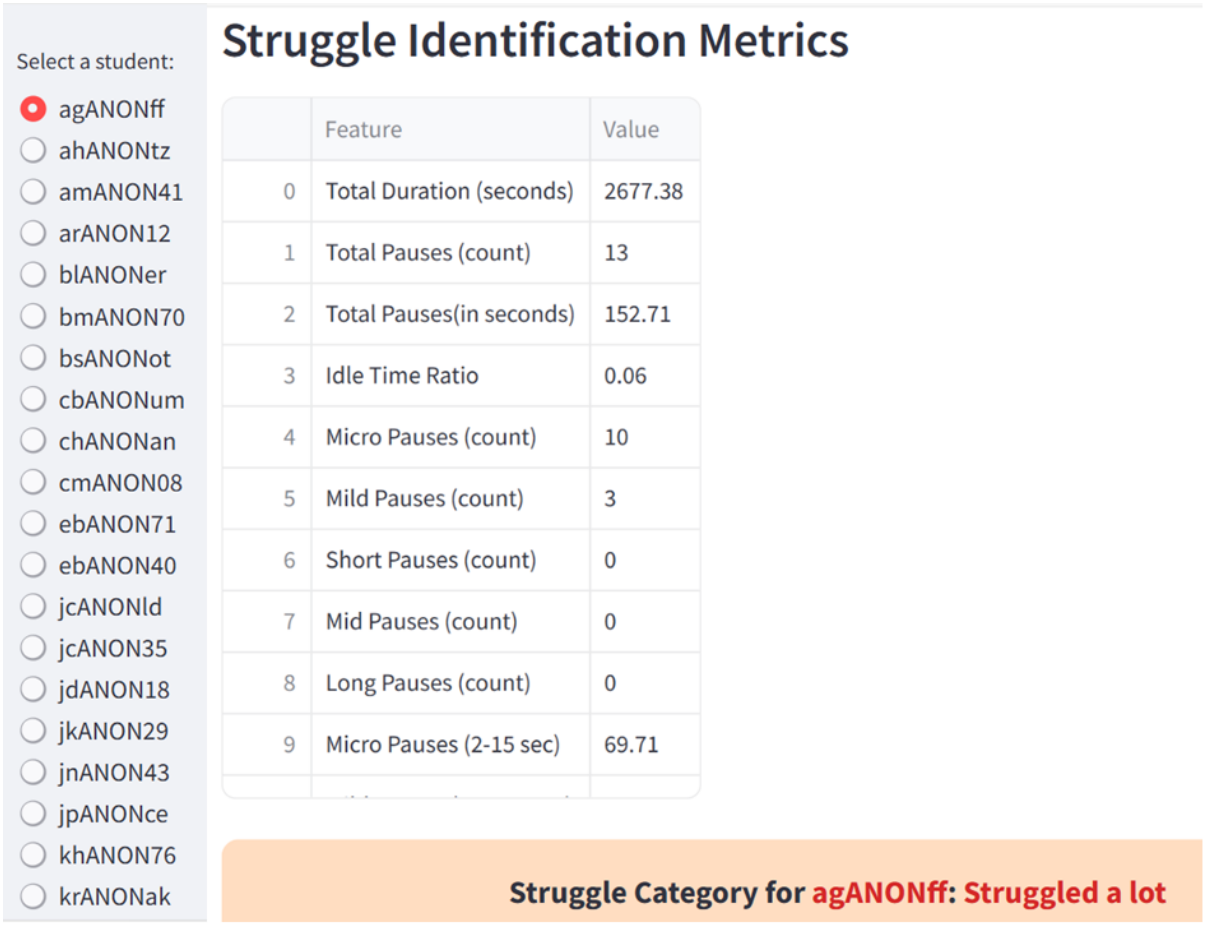
	date	position	delete	insert	version	user
0	2024-10-24 19:10:58	0	0	None	0	internal#reload
1	2024-10-24 19:10:58	0	None	# WRIT	0	internal#reload
2	2024-10-24 19:11:09	1	None	print("S	1	agANONff
3	2024-10-24 19:11:18	53	1	None	2	agANONff
4	2024-10-24 19:11:18	52	1	None	2	agANONff
5	2024-10-24 19:11:18	51	1	None	2	agANONff
6	2024-10-24 19:11:18	50	1	None	2	agANONff
7	2024-10-24 19:11:18	49	1	None	2	agANONff
8	2024-10-24 19:11:18	48	1	None	2	agANONff
9	2024-10-24 19:11:19	47	1	None	3	agANONff

Task Completion Time: 596 sec | 9 min | 0.17 hours

TrackIt: Copy-Paste Functionality and Code Progress Visualization



TrackIt: Struggle Identification Metrics



Complete Struggle Identification Metrics – Download

	Feature	Value
0	Total Duration (seconds)	2677.38
1	Total Pauses (count)	13
2	Total Pauses(in seconds)	152.71
3	Idle Time Ratio	0.06
4	Micro Pauses (count)	10
5	Mild Pauses (count)	3
6	Short Pauses (count)	0
7	Mid Pauses (count)	0
8	Long Pauses (count)	0
9	Micro Pauses (2-15 sec)	69.71
10	Mild Pauses (15-120 sec)	83
11	Short Pauses (120-180 sec)	0
12	Mid Pauses (180-600 sec)	0
13	Long Pauses (>10 min)	0
14	Total Insertions	17
15	Total Deletions	1946
16	Insert-Delete Ratio	0.01
17	Deletion Ratio	108.11
18	Correction Speed (deletions/min)	43.61
19	Typing Speed (characters/min)	0.38
20	Active Segment Length (Avg Keystrokes before Pause)	24
21	Large Pastes (50+ chars)	2
22	Very Large Pastes (100+ chars)	2
23	Struggle Score	31
24	Struggle Category	Struggled a lot
25	Struggle Rating	5

TrackIt: Struggle Features/Metrics for All Students

Show All Students' Struggle Features

📊 Struggle Features for All Students

	Total Duration (seconds)	Total Pauses (count)	Total Pauses(in seconds)	Idle Time Ratio	Micro Pau
agANONff	2,677.38	13	152.71	0.06	
ahANONtz	4,927.73	14	4,833.13	0.98	
amANON41	9,691.33	33	5,339.54	0.55	
blANONer	308.88	40	218.21	0.71	
bmANON70	10,783.23	1	6.97	0	
bsANONot	12,142.41	3	9,442.75	0.78	
cbANONum	295.63	24	222.71	0.75	
chANONan	475.41	24	325.76	0.69	