An Empirical Evaluation of Active Live Coding in CS1

ANSHUL SHAH, University of California, San Diego, USA
THOMAS REXIN, University of California, San Diego, USA
FATIMAH ALHUMRANI, University of California, San Diego, USA
WILLIAM G. GRISWOLD, University of California, San Diego, USA
LEO PORTER, University of California, San Diego, USA
GERALD SOOSAI RAJ, University of California, San Diego, USA

Objectives The traditional, instructor-led form of live coding has been extensively studied, with findings showing that this form of live coding imparts similar learning to static-code examples. However, a concern with Traditional Live Coding is that it can turn into a passive learning activity for students as they simply observe the instructor program. Therefore, this study compares Active Live Coding—a form of live coding that leverages in-class coding activities and peer discussion—to Traditional Live Coding on three outcomes: 1) students' adherence to effective programming processes, 2) students' performance on exams and in-lecture questions, and 3) students' lecture experience.

Participants Roughly 530 students were enrolled in an advanced, CS1 course taught in Java at a large, public university in North America. The students were primarily first- and second-year undergraduate students with some prior programming experience. The student population was spread across two lecture sections—348 students in the Active Live Coding (ALC) lecture and 185 students in the Traditional Live Coding (TLC) lecture.

Study Methods We used a mixed-methods approach to answer our' research questions. To compare students' programming processes, we applied process-oriented metrics related to incremental development and error frequencies. To measure students' learning outcomes, we compared students' performance on major course components and used pre- and post-lecture questionnaires to compare students' learning gain during lectures. Finally, to understand students' lecture experience, we used a classroom observation protocol to measure and compare students' behavioral engagement during the two lectures. We also inductively coded open-ended survey questions to understand students' perceptions of live coding.

Findings We did not find a statistically significant effect of ALC on students' programming processes or learning outcomes. It seems that both ALC and TLC impart similar programming processes and result in similar student learning. However, our findings related to students' lecture experience shows a persistent engagement effect of ALC, where students' behavioral engagement peaks and *remains elevated* after the in-class coding activity and peer discussion. Finally, we discuss the unique affordances and drawbacks of the lecture technique as well as students' perceptions of ALC.

Conclusions Despite being motivated by well-established learning theories, Active Live Coding did not result in improved student learning or programming processes. This study is preceded by several prior works that showed that Traditional Live Coding imparts similar student learning and programming skills as static-code examples. Though potential reasons for the

Authors' addresses: Anshul Shah, ayshah@ucsd.edu, University of California, San Diego, USA; Thomas Rexin, tjrexin@ucsd.edu, University of California, San Diego, USA; Fatimah Alhumrani, falhumrani@ucsd.edu, University of California, San Diego, USA; William G. Griswold, bgriswold@ucsd.edu, University of California, San Diego, USA; Leo Porter, leporter@ucsd.edu, University of California, San Diego, USA; Gerald Soosai Raj, asoosairaj@ucsd.edu, University of California, San Diego, USA.

ACM 1946-6226/2025/6-ART
https://doi.org/10.1145/3743686

lack of observed learning benefits are discussed in this work, multiple future analyses to further investigate Active Live Coding may help the community understand the impacts (or lack thereof) of the instructional technique.

CCS Concepts: • **Social and professional topics → Computing education**; **CS1**.

Additional Key Words and Phrases: live coding, active learning, student engagement, programming processes, learning gain

1 INTRODUCTION

Live coding is an instructional technique in which the instructor programs in front of students while verbalizing their thought process. This instructor-led live coding, which we will call Traditional Live Coding (TLC), has been the subject of extensive study in computing education research. Early works explored common student perceptions of the lecture technique [4, 5, 18, 21], subsequent works evaluated the impact of live coding on student grades and learning [25–27], and more recent works measured the effect of live coding on students' programming processes [30, 33]. The recent empirical work on live coding has compared the traditional, instructor-led form of live coding to the use of static-code examples, which is a common alternative to live coding [29]. However, the findings of these recent works have not shown any improvement in student learning as a result of live coding [25, 30, 33].

A common criticism of traditional live coding is that it can ultimately be a passive experience for students, in which they observe the instructor without any active engagement [15]. Given the lack of observed learning benefits from Traditional Live Coding, a form of live coding that includes an active learning component may offer the learning benefits that were not seen in recent empirical evaluations of live coding [30, 33]. In the form of live coding called *Active Live Coding (ALC)*, the instructor uses Traditional Live Coding with several active coding components in which students complete a small programming task, discuss with peers, and then see a demonstration of the correct solution by the instructor. From a theoretical perspective, Active Live Coding engages more Methods of Cognitive Apprenticeship [9]—a learning theory concerning the transfer of expertise from expert to learner—and involves a higher level of engagement according to the ICAP Framework [8]—a framework for classifying learning activities into a hierarchy based on student engagement.

In this study, we follow a similar experimental setup and data analysis to a recent empirical evaluation to compare Traditional Live Coding to static-code examples by Shah et al. [33]. Our study implements a course-long treatment of Active Live Coding in order to identify possible short-term *and* long-term impacts of the teaching technique. Our analysis aims to evaluate Active Live Coding across three key dimensions: 1) students' adherence to programming processes, 2) students' course outcomes and grades, and 3) students' lecture experience. Specifically, we ask the following research questions:

- RQ1: How do students' programming processes (in terms of incremental development and error frequency metrics) differ between students in the traditional and active live coding groups?
- RQ2: How do course outcomes (such as performance on exams, code comprehension questions, programming assignments, etc.) differ between students in the traditional and active live coding groups?
- RQ3: How does the student experience (in terms of engagement and perceptions of the live coding technique) differ between students in the traditional and active live coding groups?

2 THEORETICAL FRAMEWORK

There are two theories that we use to frame our study: Cognitive Apprenticeship and the ICAP Framework. We find it necessary to involve *both* theories given the difference in how the two theories impact student learning. Cognitive Apprenticeship is a learning theory that describes the instructor's choice of learning activities while the ICAP Framework describes how students *engage* with those learning activities.

Method	Description
Modeling	Instructor demonstrates a task to learners while the instructor verbalizes their thought process.
Scaffolding	Instructor provides and fades targeted learning activities for learners to practice a task with support.
Coaching	Instructor provides feedback and guidance to students as students complete tasks.
Articulation	Learner explains their reasoning and justifies the strategies they used.
Reflection	Learner reflects on their own processes and compares their strategies to the instructor's strategies.
Exploration	Learner completes tasks independently without scaffolds or support from the instructor.

Table 1. Methods in Cognitive Apprenticeship.

2.1 Cognitive Apprenticeship

The Cognitive Apprenticeship learning theory was outlined by Collins et al. and aims to bring the traditional apprenticeship model—which is one of the oldest models of knowledge transfer—into the classroom [9]. A key difference between traditional apprenticeship and Cognitive Apprenticeship is that traditional apprenticeship transfers knowledge of primarily physical tasks that can be learned through observation, such as blacksmithing or tailoring [9]. By contrast, Cognitive Apprenticeship outlines a model for instructors to *make their thinking visible* to facilitate the transfer of complex skills that require higher-order reasoning and thought processes. The initial work describing Cognitive Apprenticeship presents examples of teaching students about skills such as reading comprehension, mathematical problem-solving, and writing [9].

The Cognitive Apprenticeship learning theory broadly describes four dimensions of a learning environment: *Content*, *Sequence*, *Sociology*, and *Methods* [9]. *Content* refers to the types of knowledge that instructors should teach students, such as domain knowledge, learning strategies, and heuristic strategies [9]. *Sequence* refers to the ordering of learning activities to facilitate learning [9]. *Sociology* refers to the social characteristics of the learning environment, such as cooperation and situated learning [9]. Finally, *Methods*, which is the relevant dimension for the present study, refers to the instructional techniques to promote the development of expertise [9]. Table 1 outlines the six Methods of Cognitive Apprenticeship: modeling, scaffolding, coaching, reflection, articulation, and exploration.

Shah and Soosai Raj conducted a literature review of 143 papers that explicitly mentioned Cognitive Apprenticeship in computing education research venues [34]. The review aimed to understand which Cognitive Apprenticeship Methods have been used and evaluated in computing education and what benefits have generally been attributed to these Methods [34]. The authors found that the majority of work discussed teaching strategies that engaged the first three Methods of Cognitive Apprenticeship—modeling, scaffolding, and coaching—while significantly less work has mentioned the reflection, articulation, and exploration Methods of Cognitive Apprenticeship. One potential reason for this difference is that instructors felt that implementing the last three Methods of Cognitive Apprenticeship takes up too much lecture time [34]. Nonetheless, a key takeaway from the literature review is that deeper empirical analyses into the impact of articulation, reflection, and exploration Methods of Cognitive Apprenticeship are needed [34].

Selvaraj et al. found that the theoretical construct most commonly cited with live coding is the modeling Method of Cognitive Apprenticeship [29]. Although there are variations of live coding, the typical form of live coding involves an instructor programming in front of their students while verbalizing their thought process [29], just as prescribed in the modeling Method of Cognitive Apprenticeship [9]. In fact, the studies conducted by Shah et al. were motivated by a desire to empirically detect whether live coding—through the lens of the modeling Method—imparts implicit strategies such as incremental development and debugging techniques [33]. These works, which compared live coding to static-code examples, are empirical evaluations of the modeling Method, but they do not involve other Methods of Cognitive Apprenticeship. As Shah et al. point out in their study, the modeling Method exposes learners to only the implicit processes and strategies used by experts, but does not necessarily lead to learners being able to *gain control* over using these implicit processes. The remaining Methods, such as scaffolding, reflection, and articulation, are vital for learners to not just observe but also *apply* these implicit processes [9].

2.2 The ICAP Framework

Chi and Wylie developed the ICAP Framework, a theory related to active learning which outlines four "modes" of engagement: Interactive, Constructive, Active, and Passive (creating the acronym "ICAP") [7, 8]. A learning activity can lead to students' engagement behaviors being in one of these four modes [8]. The *Passive* mode occurs when learners simply observe and absorb information from the instructor without overtly engaging with the materials, such as taking notes. The *Passive* mode is characterized by a lack of student behavioral engagement with the instruction. The next mode of engagement is the *Active* mode, which is classified by "some form of overt motoric action or physical manipulation" being undertaken by students [8]. A specific example of *Active* engagement provided by Chi and Wylie is students copying down solution steps while listening to a lecture or taking verbatim notes [8]. Next, the *Constructive* mode of engagement is characterized by students *creating* products that go beyond what was provided in the learning materials, such as asking questions, generating predictions, or drawing diagrams [8]. Finally, the *Interactive* mode occurs when two students work together and *both* students are being constructive in their contributions [8]. Chi and Wylie note that when one partner is dominating the conversation and the other is primarily listening, then the behaviors are not *interactive*. Instead, the student dominating the conversation is in a *Constructive* mode while the student that is listening is in *Passive* mode (or *Active* if they are taking notes as they listen). Of course, an underlying assumption in the ICAP framework is that learners *enact* the behaviors that are *intended* by the instructor [8]. For example, in a *Constructive* activity where students have to write new code, they could engage with the activity only *Actively* by copy-pasting existing code from the example or *Passively* by simply not working on the task. This assumption represents a limitation of engagement-based interventions in which students fail to engage with learning activities.

The ICAP Framework has been extensively studied in various STEM Education disciplines [8, 41], such as undergraduate biology [41], high-school science [42], and undergraduate physics education [10]. The most similar study to our present study comes from Deslauriers et al., who conducted an experiment to compare the impact of traditional physics lecturing (which is *Passive* or *Active*) to a treatment condition where students are making predictions, problem solving, and discussing with each other (which is *Constructive* or *Interactive*) [10]. Although the study was only conducted in one week in the course, the results showed that students in the treatment condition scored significantly higher on the exam for that week, attended class more frequently during that week, and also shared in surveys that they enjoyed the new teaching style [10]. This study by Deslauriers et al. is highly relevant to our study because of the similarity in the treatment and control groups to our study. Our similar experimental setup, although spanning an entire term rather than one week, tests a similar set of conditions in the computer science domain.

CA Method	Trad. Live Coding	Active Live Coding
Modeling	✓	✓
Scaffolding		✓
Coaching		
Articulation		✓
Reflection		✓
Exploration		

Table 2. Theoretical Framing of Traditional and Active Live Coding according to the Cognitive Apprenticeship Methods.

ICAP Level	Trad. Live Coding	Active Live Coding
Passive	✓ (worst case)	
Active	✓ (best case)	
Constructive		✓ (worst case)
Interactive		✓ (best case)

Table 3. Theoretical Framing of Traditional and Active Live Coding according to the ICAP Framework.

The ICAP Framework has also been cited in computing education research. In their work on subgoal learning via self-explanation, Margulieux and Catrambone use the ICAP Framework to motivate the approach of using self-explanation to learn since the "higher" engagement modes are associated with more student learning. For example, a *Constructive* approach to self-explain lecture material offers greater learning benefits to students than a *Passive* approach where students simply listen to the lecture material. Indeed, Margulieux and Catrambone found that when students self-explain the subgoals of the problems they completed, they perform better on future tasks than if they did not self-explain at all [20].

The studies by Deslauriers et al. and Margulieux and Catrambone are only two of the many works that have established an empirical foundation for the ICAP Framework. Given the significant body of work that has found evidence in support of the ICAP Framework, we would expect that learning activities in the *Constructive* or *Interactive* engagement modes will result in more student learning than activities in the *Passive* or *Active* engagement modes. Therefore, the ICAP Framework would suggest that students in the Active Live Coding lectures should perform better on exams and assignments than those in the Traditional Live Coding lectures.

2.3 Theoretical Framing of Active vs Traditional Live Coding

Cognitive Apprenticeship is the primary learning theory cited with live coding [29]. We have not found any work that discusses live coding through the lens of the ICAP Framework, likely because the most common form of live coding in the literature is traditional, instructor-led live coding, which is mostly a passive learning activity. However, some clear theoretical differences arise between Traditional and Active Live Coding, which are summarized in Tables 2 and 3.

In terms of the Cognitive Apprenticeship Methods, ALC includes *modeling*, since the instructor is demonstrating the programming process while verbalizing their thoughts, *scaffolding*, since the instructor provides a small activity for students to complete, *articulation*, since students discuss their solutions with each other, and *reflection*, since students have the chance to compare their own approach to a peer's approach and the instructor's approach. We do not consider ALC to engage the coaching Method, since each student does not get direct feedback or guidance on their own approach, or the exploration Method, since students are not independently completing

open-ended tasks. In contrast, TLC only employs the *modeling* Method since students are only watching and listening to what the instructor is doing. There is no opportunity for students to complete scaffolded activities or discuss with peers. Therefore, we would expect the students in the ALC group to exhibit better programming processes than students in the TLC group since they have an opportunity to complete scaffolded activities, articulate their approach, and reflect on their approach compared to peers.

In terms of the ICAP Framework, we classify Traditional Live Coding (TLC) as either an *Active* or *Passive* learning activity. Traditional, instructor led live coding is a *Passive* activity in the sense that students can simply watch and listen as the instructor programs rather than coding along [15]. Of course, students may also be copying down the instructor's code or take notes during TLC sessions, which would constitute an *Active* engagement mode. Watkins et al. found that in live coding lectures, some students type the instructor's code, but students do not display any other engagement behaviors besides the *Active* engagement mode of typing along with the instructor. [40]. On the other hand, Active Live Coding (ALC) at the very least reaches the *Constructive* mode since students are required to write code and reaches the *Interactive* mode depending on the quality of discussion between students. In fact, in their original paper presenting the ICAP Framework, Chi and Wylie noted that a learning activity consisting of problem solving and peer discussion, which is very similar to ALC in our study, constituted an *Interactive* activity. The reason it is questionable for whether ALC is *Interactive*, however, is due to the quality of peer discussion: when a discussion is dominated by one student and the other is only listening, then neither student is experiencing *Interactive* engagement. Based on the higher engagement level associated with Active Live Coding, we would expect the learning gains to be greater in the ALC group.

3 RELATED WORK

The related work discussed in this section is organized by the three research questions we ask in this study: programming processes, student outcomes, and lecture experience. In general, the line of work related to live coding has existed since the early 2000s, with much of the early work reporting on instructor and student *perceptions* of live coding [29]. More recently, however, the studies related to live coding have also studied student behavior and outcomes, with specific empirical analyses dedicated to each of the three research questions in this study.

3.1 Impact of Live Coding on Programming Processes

Much of the early work on live coding uncovered students' and instructors' *perceptions* of live coding [4, 5, 18, 21]. For example, Bennedsen and Caspersen discussed how live coding can reveal implicit programming processes to students, such as how to use an IDE and how to use incremental development [4]. Further, Kölling and Barnes presented live coding through the lens of "apprentice-based learning," such as how the instructor first *models* the process to students, then students *apply* what they have observed, and finally *design* their own open-ended programming task [18], invoking the theory of Cognitive Apprenticeship [9]. Finally, Paxton went a step beyond discussing the goals and theory of live coding by collecting survey responses from students [21]. Direct statements from students in Paxton's study showed that students enjoyed seeing the debugging process and how an expert solves a programming task [21]. Although these papers showed that live coding aims to reveal the programming process and that students reported seeing aspects of the programming process, none of these papers empirically tested whether live coding actually *imparts* adherence to effective programming processes such as incremental development, debugging, and testing.

In order to fill this gap from prior work, Shah et al. conducted a series of experiments to compare a live coding pedagogy with a static-code one. In these experiments, half of the students in a large, CS1 course were taught via live coding during lectures and the other half of students were taught with static-code examples [33]. All other course components were identical for the two groups of students, such as assignments, lab sections, and exams.

One of the goals of these studies was to test whether students in the live coding group adhered to incremental development, debugging, and testing more than the students in the static-code group [33]. In two separate studies, the authors collected snapshots of students' code on programming assignments and coding assessments each time students ran their code. They applied a set of programming process metrics, such as the Measure of Incremental Development (MID) [32] to measure adherence to incremental development, the Repeated Error Density (RED) [3] to measure how quickly students debugged an error, and frequency of diagnostic print statements to measure how students tested and verified their code [30, 33]. However, in *both* studies from Shah et al., the authors found no significant differences across any of the programming process metrics.

Like earlier papers, the studies from Shah et al. frame their experiments on live coding through the lens of Cognitive Apprenticeship [9], specifically noting that instructor-led live coding only engages the modeling Method of Cognitive Apprenticeship [33]. This has been presented as a potential reason for the lack of significant findings related to students' programming processes. As a result, part of the motivation for the present study is to evaluate whether a learning technique that involves more Methods of Cognitive Apprenticeship—Active Live Coding—results in greater adherence to incremental development and debugging practices than a technique that only involves the modeling Method.

3.2 Impact of Live Coding on Course Outcomes

The first empirical studies to evaluate live coding primarily compared students' exam scores between a static-code group and a live-coding group in introductory CS courses. Rubin conducted the first comparative, empirical study between live coding and static-code examples. In a large introductory programming course with four lecture sections, Rubin selected two lecture sections to be live-coding groups and used the other two lecture sections as control groups that would learn via static-code examples. Rubin found that the groups scored similarly on the programming assignments and course exams, indicating a similar amount of student learning from the two lecture styles. One difference, however, between the experimental and control groups was that the live coding group scored higher on the final project at the end of the course, which was graded manually for correctness and clarity [27]. Importantly, in their interpretation of results, Rubin notes that students may have scored better on the final project because students' *debugging* skills would be better in the live coding group after seeing the instructor debug. However, Rubin did not conduct any empirical analyses on the students' debugging processes.

In a similar follow-on study conducted by Raj et al., the authors wanted to 1) measure the cognitive load associated with static code examples and live coding via surveys and 2) compare students' learning via a pre-test and post-test [25]. In terms of cognitive load, the authors found that live coding was associated with significantly less extraneous cognitive load compared to the static-code group. Extraneous load relates to the load on working memory that gets in the way of student learning, such as being distracted by other students during lecture or hearing disorganized lecture material [25]. The authors found no significant differences on learning gain between the pre-test and post-test, although the static-code group showed slightly higher, though not significant, learning gain than the live coding group [25].

The series of works by Shah et al. also investigated the impact of live coding compared to static code examples on students' performance on assignments and exams. In the two experiments conducted by Shah et al., both found similar outcomes between live coding and static code groups on exams and assignments, with no statistically significant differences between the groups [30, 33]. Even a deeper analysis into student performance on code tracing questions, code writing questions, and code explaining questions showed similar student performance across these different types of questions [33]. In general, these works from Shah et al. confirmed the prior findings that compared a static code pedagogy to a live coding pedagogy related to course outcomes—in general, there is little to no difference in student performance on exams and assignments between the two types of code examples [25, 27, 30, 33].

3.3 Impact of Live Coding on Lecture Experience

A significant amount of work has concerned the impact of live coding on students' lecture experience, revealing a variety of benefits and drawbacks of live coding.

The benefits of live coding on students' lecture experience includes, but is not limited to, revealing the programming process to students [18, 21, 26, 33], reducing cognitive load during lecture [25], and potentially engaging more students during lecture [6, 21, 26, 31]. Many early works related to live coding, as mentioned before, touted live coding as a way to expose the implicit programming process to students [18, 21]. In fact, Shah et al. conducted an open-ended survey to students in both the live coding and static code lecture groups in their study to understand the main perceived benefits from students' point of view [33]. The qualitative analysis revealed that students in the live coding group mentioned observing some part of the programming process at a higher rate than students in the static code group. The opposite was true for "Code Comprehension," however, as more students in the static code group reported that seeing the static code examples improved their understanding of the code's purpose than students in the live coding group [33]. Another key perceived benefit of live coding, which has not yet been empirically tested, is that live coding results in higher student engagement [6, 21, 26] (the *type* of engagement—cognitive, behavioral, or emotional [13]—is not typically specified in these works). For example, student feedback in Paxton's work showed that students found it fun to see the output of running code [21]. Similarly, students' feedback in Raj et al.'s study showed that they tend to type along with the instructor as they live code [26]. It seems intuitive that students may be more engaged watching an instructor program dynamically during lecture, but this claim has not been empirically tested. In fact, this lack of empirical evaluation motivates part of our third research question, which is to measure students' behavioral engagement in the Traditional Live Coding and Active Live Coding lecture sections.

The drawbacks of live coding have also been extensively identified, such as the difficulty for students to follow along with the instructor [26, 33] and the limited time in a lecture that results in a rushed live coding example [6, 33, 39]. Shah et al. also conducted an open-ended survey to ask students about the drawbacks of the code examples in their lecture for both the static-code and live coding groups [33]. Nearly 20% of the students in the live-coding group suggested that the instructor should *slow down*, whereas only 2% of students in the static-code group suggested the same thing [33]. This feeling of a rushed lecture pace is likely because live coding simply takes more time than static-code examples [6]. Indeed, Watkins et al. conducted a comparative study of live coding and static-code examples in a single lab session. They found that the live coding session, which covered the same material as the static-code session, took more than twice as long to complete [39]. Although the study by Watkins et al. was in a lab section with flexible timing, when an instructor is bound to a fixed-time lecture, there certainly exists a time constraint to complete all the material. The impact of this rushed lecture pace is that students are unable to follow along as easily. Shah et al. included an analysis on a set of anonymous, end-of-course feedback items that asked students whether the instructors' lecture style *facilitated note-taking* and *held students' attention*. In *both* questions, there was a statistically significant difference showing that students in the live coding lectures had a *harder time* note-taking and paying attention [33]. Given these downsides, instructors must be careful to keep their live coding sessions to a reasonable pace and to ensure that the class is able to follow along with the example. Indeed, there are many factors that determine the effectiveness of a live coding lecture [29], revealing the difficulty of using live coding.

4 STUDY CONTEXT

4.1 Course Setup

The study was conducted in the Fall 2023 term at UC San Diego—a large, public, research-focused university in North America. The course was an advanced CS1 course taught in Java. The course content included basic data types, basic data structures, and object-oriented programming, such as classes, inheritance, and generics. The

course enrollment was 600 total students, who were split into a 400-person Active Live Coding lecture section taught at 9:30AM on Tuesdays and Thursdays and a 200-person Traditional Live Coding lecture section taught at 11AM on the same days. Both lecture sections were taught by the same instructor. When students registered for the course, they only knew the instructor of the course and did not know about the difference between the lecture sections (i.e., that one would be Traditional Live Coding and one would be Active Live Coding).

In a typical week of the course, students attended two lecture sections for 80 minutes each and a mandatory discussion section for 50 minutes. Students also completed weekly programming assignments (PAs), worksheets, and textbook activities in an online textbook hosted on Stepik [35]. The frequency and description of the different course components is provided in Table 4.

Table 4. Key course components of the CS1 course.

Component	Frequency	Description
Lectures	Twice per week	Each lecture is 80 minutes and covers the main course material. The treatment condition of Active Live Coding only applies to the lectures. An examination of the lecture structure is in Table 5.
Programming Assignments (PAs)	Once per week	Students apply the material they learned in lecture in a weekly programming assignment graded for correctness. Assignments are hosted on the Edstem online IDE [12].
Worksheets	Once per week	Students independently complete a paper-based worksheet with code tracing, writing, and explaining questions.
Reading Quizzes	Once per week	Students independently complete interactive programming activities in an online textbook and can submit responses unlimited times without penalty.
Midterm Exam	Once per term	Students independently complete an in-person, proctored exam for 2 hours, covering the concepts taught in the first half of the course.
Midterm Coding Challenge	Once per term	Students complete a proctored coding task on Edstem [12]. Students have 45 minutes to complete the task, which is graded based on correctness.
Final Exam	Once per term	Students independently complete an in-person, proctored exam for 3 hours, covering all concepts taught in the course.
Final Coding Challenge	Once per term	Just like the Midterm Coding Challenge, students independently complete a proctored coding task in 45 minutes and are graded based on correctness.
Discussion	Once per week	Students may attend an in-person, 50-minute session to review the material covered in that week's lectures and preview the next programming assignment.
Office Hours	Everyday (optional)	Students may attend office hours held by course staff (TAs, tutors, etc.) to receive help on course materials. Office hours were hosted M-F from roughly 9 A.M. to 7 P.M.

4.2 Lecture Structure

Both the TLC and ALC lectures covered the same material and featured similar instructional activities as seen in Table 5. Before the lecture started, the course staff handed out an in-class worksheet to each student. This worksheet included small code examples and simple questions related to the upcoming lecture material for additional practice. Each lecture, the instructor either asked students to complete the worksheet during the lecture or assigned the worksheet questions to be completed before next lecture. The lectures always began with roughly 5 minutes of course announcements, followed by a pre-lecture questionnaire that students submitted via Gradescope [16] within a 10-minute window for attendance. These pre-lecture questions were meant to capture students' understanding of the lecture material before the lecture began and provided a basis for measuring learning gain by asking the same questions after the lecture. Following the announcements and pre-lecture questions, the instructor reviewed the in-class worksheet from the previous lecture, typically taking up to 10 to 15 minutes.

In both ALC and TLC lectures, the instructor then began teaching new material through Traditional Live Coding, complemented by handwritten notes. During these live coding sessions, the instructor occasionally posed questions to the students, asking them to predict missing code segments or the output of the code. In the ALC lecture only, the professor then initiated an Active Live Coding segment, marking the point where the two lectures diverge. In the active live coding segment, the instructor provided a scaffolded Java file for students to complete, which was a result of the previous Traditional Live Coding segment. Students usually had to write a simple method or implement the key logic of a method. During this phase, students forked the instructor's workspace on Edstem and spent 5 to 7 minutes completing missing code. Following this, they spent 3 to 5 minutes discussing their approach with peers, a process similar to peer instruction [22]. Finally, the professor explained the solution using Traditional Live Coding and continued to cover the remainder of the new content. Typically, the instructor used between 1 to 3 ALC components in each lecture. The only difference between the TLC and ALC lectures was the active coding done by students and the peer discussion following the active coding. Importantly, *both lectures had traditional live coding components*, but only the ALC lecture had the active live coding component.

During the TLC section, the instructor used this time to live code the same material that students were asked to write during the ALC lecture. However, given that the ALC components take significantly longer than simply using TLC to show the same material, the instructor could slow down during parts of the TLC lecture or spend more time explaining and debugging an error. The lectures all ended with post-lecture questions that took approximately 5 minutes. Students were free to leave once they finished answering the questions.

4.3 Coding Challenges

An important part of the course setup and data collection for RQ1 were the Midterm and Final Coding Challenges. In both coding challenges, students were given 45 minutes to 1) create new methods and classes and 2) write accurate tests for their implementations. These summative assessments were developed by the research team to evaluate how students create programs on their own in a controlled environment. In the Midterm Coding Challenge (MCC), students were given a partially-complete class with fields and a valid constructor and were tasked with adding a method to the existing class, writing a new class and defining some fields, a constructor, and methods for that class, and creating a tester class that tests both the given class and newly created class. The concepts in the MCC included string concatenation, conditionals, incrementing of variables, and testing. In the Final Coding Challenge, students were tasked with writing two functions, one of which computed the average of a given column in a 2-D array and the other returned a cropped version of the 2-D array. The concepts on the FCC included nested for-loops, array indexing, modifying the elements in an array, and testing edge cases. For both coding challenges, students were graded on correctness based on the amount of test cases that their code passed.

Table 5. The different instructional components used during the two lectures.

Activity	Description
Course Announcements	Instructor provides due date reminders and discusses course logistics
Pre-Lecture Questions	Students answer two multiple-choice questions on Gradescope for attendance credit
Worksheet Review	Instructor solves a problem from an in-class worksheet on a projector
Handwritten Notes	Instructor handwrites notes on a blank piece of paper in front of the class using a projector.
Traditional Live Coding	Instructor codes in front of students and prompts questions to the class
Active Live Coding	Students extend the code from the instructor's workspace for attendance credit
Post-Lecture Questions	Students answer two more questions on Gradescope for attendance credit

4.4 Participants

In accordance with our human subjects protocol, students consented to release their data for research purposes at the start of the course. At the start of the term, the members of the research team sent out a consent form to all students in the class that described the general goals of the research project and asked students whether 1) they were at least 18 years old and 2) consented to release their data for research purposes. The instructor of the course was not allowed to see the list of consenting students at any point in the research project. A total of 521 students across both lecture sections agreed to release their data. This human subjects protocol and consent process was only used to collect student data related to their programming processes.

At the start of the quarter, we distributed a survey to gather students' demographics, confidence level, and reason for taking the course. The students' information in each group is summarized in Table 6. The students had very similar distributions for school year and we also did not see any glaring differences between the racial or gender makeup of the two lecture sections. There was a slight difference in the metrics related to confidence, as more students in the ALC lecture responded with a "4" (confident) and "5" (very confident) on their confidence to do well in the course. Similarly, more students in the ALC lecture would be satisfied with an "A" grade than TLC students.

In terms of demographics among the consenting students, 61% of students in the ALC group identified with he/him/his pronouns and 36% identified with she/her/hers pronouns compared to 62% and 34% of students in the TLC group, respectively. Furthermore, the self-identified racial makeups for both groups consisted of about 70% Asian or Asian American students, 15% Chicanx or Latinx students, and 15% White or Caucasian students. The remaining students self-identified into racial groups with less than 10 students and we do not disclose these groups in accordance with our human subject protocol.

5 METHODS

5.1 RQ1 Methods: Programming Processes

To collect data regarding students' programming processes, we conducted two coding challenges, as discussed in Section 4.1. Before the experiment began, we reached out to Edstem [12] about obtaining snapshots of students'

Table 6. Comparison between Active Live Coding and Traditional Live Coding lecture sections.

	School Year				
	First	Second	Third	Fourth	Fifth
ALC	51.9%	24.4%	17.1%	5.1%	0.6%
TLC	48.4%	24.2%	18.5%	8.3%	0.6%
	Confidence rating in ability to do well in the course				
	1 (low)	2	3	4	5 (high)
ALC	1.0%	4.8%	25.6%	47.5%	21.2%
TLC	1.9%	7.0%	31.2%	40.1%	19.8%
	Minimum final grade that students would be satisfied with				
	A-range		B-range	C-range	
ALC	70.1%		25.6%	3.2%	
TLC	64.3%		32.5%	3.2%	

code every time they compiled their code by clicking the "Run" button during a programming task. The team at Edstem agreed, provided that we show the Edstem staff the consent form students agreed to at the start of the term and the responses to the form showing which students consented. After sending the list of consenting students to Edstem, we obtained snapshots of students' coding workspace each time they compiled and ran their code in the coding challenges. Using this data, we analyzed students' programming processes across three dimensions: 1) adherence to incremental development, 2) error frequencies, and 3) programmer productivity.

5.1.1 Comparing Adherence to Incremental Development. Incremental development has been specifically cited as a perceived learning benefit of live coding [29]. Shah et al. already conducted an analysis using the Measure of Incremental Development (MID) to compare students in a static-code pedagogy to a live-coding one [33]. Therefore, we aimed to replicate this analysis for our experiment. We applied the language-agnostic metric to our data since the MID has been tested on simple programming tasks between one to three functions long, which precisely describes our coding challenges [32]. The MID is publicly-available as a Python package[1] and the source code is freely available, allowing us to access the code and modify its functionality to fit the Java programming language.

We computed MID values for all Midterm and Final Coding Challenge submissions. The sample size for both analyses was 345 students for the Active Live Coding group and 185 students for the Traditional Live Coding group. Given this sufficiently large sample size, we conducted two-sample t-tests [24] to detect any differences, if any, between the two groups.

5.1.2 Comparing Error Frequencies and Debugging. For debugging and error-frequency measures, we applied the Repeated Error Density (RED) developed by Becker [3]. The RED uses the output of student code to track the frequency of error messages and penalize students for consecutive errors of the same type [3]. For example, three consecutive syntax errors in a row will result in a higher RED score than three syntax errors with at least one compilation between each error [3]. A higher RED value indicates a greater frequency of a particular error.

To calculate the RED, we wrote a script to iterate through each students' snapshots, compile the Java file for each snapshot, and capture the output of the program execution. We parsed the outputs to identify which snapshots results in errors and which errors occurred. As with the MID analysis, we conducted two-sample t-tests to compare the two lecture sections across the various error types on the Midterm and Final Coding Challenges.

[1] https://pypi.org/project/measure-incremental-development

The most common errors that students encountered were Symbol Not Found, Missing Identifier, Syntax Error, Type Mismatch, Non-Static Access, and Redefinition Conflict errors, described in Table 7.

Table 7. Description of the errors tracked for RED Analysis

Error Name	Error Description
Symbol Not Found	A reference to a variable or method cannot be located by the compiler
Missing Identifier	Compiler encounters a statement or expression that requires an identifier but finds something else instead
Syntax Error	Compiler encounters a statement or expression that requires an syntactical token (such as a semicolon, parenthesis, etc.) but does not find the token
Type Mismatch	An attempt to assign or operate on a value of one data type with another incompatible data type
Non-Static Access	An attempt to access a non-static variable or method from within a static context
Redefinition Conflict	An attempt to define a variable, method, or class with the same name as one that already exists in the current scope

5.1.3 Comparing Programmer Productivity. Finally, we collected data about programmer productivity on the coding challenges in terms of the number of compilations that students conducted before their final submission and the number of requirements correctly implemented. The number of compilations is easily derived based on the number of snapshots for each student in our dataset. To determine the number of requirements satisfied, we used student grades on the MCC and FCC since these challenges were graded on whether or not they passed a set of hidden test cases that were run only after students made their final submission. As with the analyses related to incremental development and error frequencies, We conducted two-sample t-tests across the comparisons for programmer productivity.

Though we aimed to fully replicate the analysis from Shah et al., we were unable to collect data related to the time to completion. Though we had timestamps in the snapshot data from Edstem, we did not know the start time for each student. As a result, we did not analyze time to completion in this study.

5.2 RQ2 Methods: Course Performance

Our second research question compares 1) course grades, including assignments, attendance, and exam scores, and 2) performance across code tracing and code writing questions on the exams.

5.2.1 Comparing Student Grades. As discussed in Section 4.1, in a typical week, students completed a reading quiz, attended two lectures, attended a discussion section, completed a programming assignment, and completed a homework worksheet. Students also completed two exams during the term. Overall, students accumulated grades the following categories: lecture attendance, programming assignments, exams, and worksheets. To compare student grades, we compared final grades across these various course components using two-sample t-tests.

5.2.2 Comparing Student Performance on Code Tracing and Code Writing Questions. Although overall course grades is an important measure of students' learning outcomes, it is a coarse measurement of students' skills. As a result, we aimed to compare student performance across various types of exam questions, as done by Shah et al.

Table 8. Counts of each question type on the midterm and final exams.

Question Type	Midterm	Final
Basic Questions	2	1
Code Writing	4	4
Code Tracing (Loops)	0	4
Code Tracing (No Loops)	4	5
Code Explaining	1	0

[33]. For this, two members of the research team classified each question from the midterm and final exam into the following question types, which were described by Venables et al. [38]:

- **Code explaining** questions involve students describing the purpose of a piece of code in plain English.
- **Code writing** questions involve students generating blocks or lines of code. These questions may ask students to fill in the blank of a nearly-completed program or even write an entire function.
- **Code tracing (without loops)** questions involve students predicting the output of a piece of code from a given input, provided that the code does not include any loops. Venables et al. distinguish code tracing questions based on whether or not there is a loop (`while` or `for`) in the provided code because of the added complexity of the loop.
- **Code tracing (with loops)** are the same as the item above, except applies to code tracing questions where the code includes a `while` or `for` loop.
- **Basic questions** involve more students' answering conceptual or theoretical questions that do not fall into the categories above.

The counts of each question type for each exam are outlined in Table 8.

5.2.3 Comparing Student Learning Gain During Lecture. A deeper description of the methods related to learning gain during lecture can be found in previously-published work from this experiment [2]. To calculate learning gain, we analyzed the pre- and post-lecture question results. At the start of the experiment, we aimed to have the pre- and post-lecture questions be isomorphic [43], but after 3 weeks, we made the pre- and post-lecture questions to be the *exact same* due to concerns about whether the questions were truly isomorphic.

Our method for calculating learning gain is the same as Porter et al. in their study related to learning gain in Peer Instruction [23]. In order to make the analysis agnostic to different correctness levels between the two groups in the pre-lecture questions, the calculation of learning gain focuses only on the Potential Learner Group (PLG)—the group who answered the pre-lecture questions *incorrectly*. The learning gain metric for each question is the percentage of PLG students that correctly answered the post-lecture question.

5.3 RQ3 Methods: Lecture Experience

Our third research question consists of 1) identifying student perceptions of Active Live Coding, 2) comparing perceptions of the Traditional Live Coding components, which occurred in both lectures, and 3) comparing student behavioral engagement during lectures.

5.3.1 Identifying Student Perceptions of Active Live Coding. In Week 4 of the course, we required students to complete a reflection survey that counted for 1% of weekly programming assignment. The survey was sent to all students, but one section of the survey was specifically for students who were part of the Active Live Coding (ALC) lecture section. In total, 406 students responded to the survey out of the 531 total students that finished the course, resulting in a response rate of 76.5%. Of the 406 total responses, 271 were in the ALC group. One of the

questions that we asked specifically to ALC students was: "On a scale of 1 to 5, how helpful is the active coding component of lectures in which you write a small part of the live coding example?" We also asked an open-ended, follow-up question that read: "Please give a brief explanation of your rating on the active coding component."

We conducted a bottom-up, thematic analysis (also called "open-coding") of the 271 student responses [17]. In this process, two members of the research team conducted several rounds of independently coding the data and then deliberating together to create a final code book that includes the variety of themes seen in the student responses. The two coders could apply multiple labels to a single student response if appropriate. In the first round, the two coders independently analyzed the first 30 responses from students and wrote down one or multiple *themes* for each answer. For each theme, the coders wrote down a general description of the meaning of that theme. After the independent coding, the coders met to go over the themes they identified and come to a consensus on the first iteration of the code book, which was only based on the first 30 student responses at the time. During this deliberation phase, the coders not only decided on the themes and descriptions for the code book but also came to a consensus on the theme(s) for each of the first 30 responses. In the second round of coding, the coders analyzed the next 30 student responses. In the independent coding phase of the round, the coders could apply an existing theme from the code book or could propose a new theme if an existing theme did not fit. Further, the coders could modify or add to the existing description of the themes to more accurately reflect the meaning of the theme. The two coders then met again to create the second iteration of the code book and to agree on codes for the 30 responses they individually reviewed. This process continued for four more rounds, with the coders completing 30 responses in the third round, 60 responses in the fourth round, 60 responses in the fifth round, and the remaining 61 responses in the sixth round.

The final code book can be found in Table 19 in the Appendix.

5.3.2 Comparing Perceived Benefits of Traditional Live Coding. Though the analysis above uncovers the perceptions of Active Live Coding (ALC), we wanted to compare student perceptions of the Traditional Live Coding (TLC) components of the lectures, since both groups of students were exposed to this lecture technique. The goal of this analysis was not only to replicate the analysis conducted by Shah et al. [33] but also to determine whether there is a difference in student perceptions of TLC between students who have been exposed to ALC and who have not. To make this comparison while also replicating the analysis from Shah et al., we included *the same survey question as Shah et al.* in the Week 4 survey mentioned above. The survey question reads: *"What are some specific things about the live-coding examples that have been helpful for your learning?"*. As mentioned above, 406 students responded to this survey.

Our open-coding process for this survey question relied on the same code book developed by Shah et al. for the same question [33]. Therefore, we did not undergo the same open-coding process as the analysis for the perceptions of ALC since the code book was already developed. Instead, two coders (the same two coders who conducted the analysis of perceptions of ALC) studied the code book from Shah et al. and began the process of independently coding the responses and then deliberating to resolve disagreements. The coders could apply multiple labels to a student response. Though the code book was already created, we were concerned that there may be responses in our data that did not fit the existing code book. Therefore, we instructed the coders to modify or expand the code book if needed. Ultimately, however, our coders did not need to make any changes to the original code book. The coders analyzed the 406 responses across five rounds of analysis. The coders analyzed 50 responses in the first round, 75 in the second round, 75 in the third round, 100 in the fourth round, and 106 in the fifth round. The final code book is the same as the code book presented in the appendix of the original work by Shah et al. [33].

One unique situation arose with our analysis of this open-ended question, which was intended to ask students about *only the Traditional Live Coding* portion of the lectures. Several students in the ALC group responded to this question with a response that discussed their perception of *Active Live Coding* rather than *Traditional*

Table 9. Condensed rubric used by classroom observers, developed by Lane and Harris [19].

	Engaged Criteria	**Disengaged Criteria**
Computer Use	Student is taking notes or has instructor's code open	Student is using the computer for non-class purposes or is doing homework for the class
Looking/ Listening	Student is looking at the instructor and nodding along to demonstrate attention	Student is not looking at the projector and instead is distracted by phone, laptop, etc.
Student Interaction	Student is not talking to neighbors during instruction	Student is talking or laughing with neighbors during instruction

Live Coding. This likely occurred since students in the ALC group assumed that the active coding component is simply part of a "live coding" pedagogy. Therefore, in our qualitative analysis, we instructed the two coders to indicate when they felt that an answer was specifically about Active Live Coding, such as mentioning the process of coding themselves or forking the instructor's workspace. In total, the coders identified 52 of the 406 responses that were specific to Active Live Coding. We excluded these answers from the analysis to compare perceptions of live coding. However, we still conducted an analysis of these 52 responses using the code book developed for the Active Live Coding analysis (Figure 2).

Since students answered the question about their perceptions of Active Live Coding only in a later part of the same survey, we did not include these 52 responses in the other analysis to avoid duplicate perspectives from students. Instead, we conducted a separate qualitative analysis using the same code book from the Active Live Coding perceptions analysis. The goal of this extra analysis was to ensure that we did not miss additional, unique perspectives of Active Live Coding.

The final code book that we created can be found in Table 20 in the Appendix.

5.3.3 Comparing Behavioral Engagement During Lecture. Our analysis on student behavioral engagement is part of a previous publication [31].

To measure the behavioral engagement level of students during lectures, we used the Behavioral Engagement Related to Instruction (BERI) protocol [19]. The BERI protocol is an observation method created by Lane and Harris that was specifically designed to measure student engagement in large lecture halls that exceed one hundred students [19]. The method has been tested for validity and reliability and has been shown to accurately capture student engagement levels of an entire lecture hall with just one classroom observer [19]. In the BERI protocol, an observer with a clear understanding of the course material and knowledge of the BERI protocol positions themselves in a seat among the students during the lecture. At the beginning of the lecture, the observer selects 10 students to observe throughout the lecture. Then, several times throughout the lecture, the observer spends roughly 15 seconds observing each of the 10 students and applies a rubric, displayed in Table 9, to determine whether each student is engaged or disengaged.

To apply the BERI protocol, we used two observers, even though Lane and Harris showed that one observer is sufficient for reliable data collection [19]. Among the two observers, we had one primary observer who attended every lecture throughout the term and a secondary observer attended roughly half of the lectures (one lecture per week rather than both lectures per week). When the two observers were at the same lecture, they coordinated their data collection times so that the observations happened at the same time. The observers made sure to sit in different parts of the classroom in the same lecture to cover a greater variety of seating locations. Furthermore, to ensure consistency *between lecture sections*, the observers made sure to sit in the same relative area of the classroom for the two lectures so that the data collected on a specific day would be comparable between the

Table 10. Comparison of Measure of Incremental Development (MID) on Midterm Coding Challenge (MCC) and Final Coding Challenge (FCC). A lower MID value indicates more adherence to incremental development.

Item	Group	N	Mean	Std dev	t-stat	p value	Cohen's D
All MCC Submissions	Active LC	345	1.71	2.43	0.82	0.41	0.08
	Traditional LC	185	1.54	2.01			
Perfect Scores on MCC	Active LC	223	1.21	1.74	0.76	0.45	0.08
	Traditional LC	123	1.08	1.29			
All FCC Submissions	Active LC	338	1.73	2.30	-1.42	0.16	-0.13
	Traditional LC	184	2.04	2.39			
Perfect Scores on FCC	Active LC	219	1.40	1.72	-2.87	<0.01*	-0.32
	Traditional LC	123	2.08	2.67			

two lectures. The observers aimed to be as discrete as possible during lectures as to not cause students to act differently. Our observations happened roughly every 10 to 15 minutes during the lecture. The observers ensured that they collected data at least once per instructional activity (see Table 5). For each data collection moment, the observers recorded how many of their 10 students were engaged. With this frequency of data collection, we were able to create a representation of how student engagement changes *throughout a lecture* as the instructor shifts between instructional activities.

6 RESULTS

6.1 RQ1 Results

6.1.1 Comparing Adherence to Incremental Development. Table 10 compares student adherence to incremental development, measured via the Measure of Incremental Development [32], on the Midterm Coding Challenge (MCC) and Final Coding Challenge (FCC). The table compares MID scores, where a lower value indicates more adherence to incremental development, across all submissions for the MCC and FCC. However, we also wanted to see whether there was a difference in MID scores between the students who correctly solved the coding challenges, since students who struggle to get the correct answer may exhibit different programming processes. Across all submissions, we found no statistically significant differences in terms of adherence to incremental development. However, when we isolated our analysis to only the perfect scores so that measurement of incremental development did not include students who struggled to complete the task, we see that the Active Live Coding group had a higher adherence to incremental development on the FCC.

6.1.2 Comparing Error Frequencies and Debugging. Tables 11 and 12 shows the comparison of students' error frequencies, measured via the Repeated Error Density [3], on the Midterm Coding Challenge and Final Coding Challenge, respectively. To interpret the RED, a *lower* value indicates a lower error frequency. We ran the RED for the errors described in Table 7. However, for the FCC, there was no Non-Static Access error because there were no static methods involved in the FCC. Therefore, we exclude this error for our analysis on the FCC. As seen in Tables 11 and 12, we found no statistically significant differences in any of the error frequencies on the MCC or FCC. Despite the lack of significance, the effect sizes tended to favor the TLC group, who had lower average RED values. However, the non-significance of the differences ultimately points to a similar error frequency between the groups, despite the trend in the small effect sizes.

Table 11. Comparison of Repeated Error Density (RED) on Midterm Coding Challenge (MCC). A lower RED value indicates less frequent errors.

Item	Group	N	Mean	Std dev	t-stat	p value	Cohen's D
Symbol Not Found	Active LC	345	1.91	3.58	1.84	0.07	0.17
	Traditional LC	185	1.34	3.15			
Missing Identifier	Active LC	345	0.10	0.6	-0.88	0.38	-0.08
	Traditional LC	185	0.15	0.76			
Type Mismatch	Active LC	345	0.04	0.18	0.95	0.34	0.09
	Traditional LC	185	0.02	0.11			
Syntax Error	Active LC	345	0.55	2.10	0.18	0.86	0.02
	Traditional LC	185	0.51	1.91			
Non-Static Access	Active LC	345	0.05	0.42	0.07	0.95	0.01
	Traditional LC	185	0.05	0.42			
Redefinition Conflict	Active LC	345	0.16	1.15	0.71	0.48	0.06
	Traditional LC	185	0.09	0.34			

Table 12. Comparison of Repeated Error Density (RED) on Final Coding Challenge (FCC).

Item	Group	N	Mean	Std dev	t-stat	p value	Cohen's D
Symbol Not Found	Active LC	338	0.60	1.59	-0.3	0.77	-0.03
	Traditional LC	184	0.65	2.04			
Missing Identifier	Active LC	338	0.02	0.19	-0.57	0.57	-0.05
	Traditional LC	184	0.03	0.34			
Type Mismatch	Active LC	338	0.28	1.19	1.40	0.16	0.13
	Traditional LC	184	0.15	0.56			
Syntax Error	Active LC	338	0.59	1.83	1.41	0.16	0.13
	Traditional LC	184	0.38	1.06			
Redefinition Conflict	Active LC	338	0.03	0.17	-0.48	0.63	-0.04
	Traditional LC	184	0.04	0.25			

6.1.3 Comparing Programmer Productivity. Tables 13 and 14 shows the two dimensions of programmer productivity that we collected: correctness on the coding challenges (Table 13) and the number of compilations until submission (Table 14). We did not find any significant differences in terms of p-values, though the small effect sizes favor the TLC group. Notably, as seen in Table 13, the TLC group scored 4 to 5 percentage points higher than the ALC group in both coding challenges. Similarly, across both coding challenges and even among perfect scores on coding challenges, there was no statistically significant different between the groups.

6.2 RQ2 Results

6.2.1 Comparing Student Grades. Table 15 compares scores on textbook activities, lecture attendance, discussion attendance, programming assignment (PA) grades, worksheet grades, final exam score, and overall course grades between the Active Live Coding and Traditional Live Coding groups. We did not detect any statistically significant

Table 13. Comparison of correctness on Midterm and Final Coding Challenges.

Item	Group	N	Mean	Std dev	t-stat	p value	Cohen's D
MCC Correctness	Active LC	348	84.2	25.7	-1.79	0.07	-0.16
	Traditional LC	183	88.2	21.9			
FCC Correctness	Active LC	348	78.4	34.1	-1.57	0.12	-0.14
	Traditional LC	183	83.2	32.4			

Table 14. Comparison of number of compilations until final submission on Midterm Coding Challenge (MCC) and Final Coding Challenge (FCC)

Item	Group	N	Mean	Std dev	t-stat	p value	Cohen's D
All MCC Submissions	Active LC	346	20.8	16.5	1.63	0.10	0.15
	Traditional LC	185	18.5	13.8			
All FCC Submissions	Active LC	334	19.2	14.6	0.83	0.41	0.08
	Traditional LC	182	18.2	12.2			
Perfect Scores on MCC	Active LC	223	15.6	12.3	1.03	0.30	0.12
	Traditional LC	123	14.3	10.3			
Perfect Scores on FCC	Active LC	218	17.6	14.3	0.85	0.40	0.10
	Traditional LC	122	16.3	11.6			

differences except for the final exam scores, which showed the Traditional Live Coding group scoring higher than the Active Live Coding group by 4 percentage points.

However, in order to more fully examine the factors that led to this difference in final exam grades, we conducted a Multiple Linear Regression (MLR) analysis [1] in order to determine the relative impact of the different independent variables we collected—prior programming experience, university major, and year-in-university. The results of our MLR analysis is shown in Table 16. In Table 16, there is a baseline value in each category: the baseline for Year is first-year, the baseline for Major is computer science, the baseline for Prior Experience is no, and the baseline for Treatment is the Traditional Live Coding group. Each coefficient value in the table shows the relative effect of that predictor *compared to its baseline value*. For example, to interpret the Treatment grouping, we would say that all else being equal (such as Year, Major, and Prior Experience), a student in the Active Live Coding condition is expected to score 0.53 percentage points *lower* on the final exam than if they were a student in the Traditional Live Coding condition. However, the effect of the ALC treatment is not statistically significant, as seen from the right-most column with the p values. Each of the groupings besides the Treatment condition had a predictor with a significant association with final exam grades: second-year, math major, other major, and prior programming experience. The results of this MLR analysis reveal that the significant difference shown in Table 15 is not due to the Treatment condition but rather can be explained through differences in Year, Major, and Prior Experience.

Finally, we conducted a Leave-One-Out analysis [37] to understand the relative impact of each grouping (Year, Major, Prior Experience, and Treatment) on our model's performance, which is shown in Table 17. The results of the Leave-One-Out analysis are shown in order of significance, where the Treatment condition has the lowest impact on model performance, then Year, then Major, and finally Prior Experience, which has the largest impact

Table 15. Comparison of overall course performance between live-coding and static-code groups

Item	Group	N	Grade (out of 100) Mean	Std dev	t-stat	p value	Cohen's D
Textbook Activities	Active LC	348	97.6	12.8	-0.69	0.49	-0.06
	Traditional LC	183	98.3	8.4			
Lecture Attendance	Active LC	348	95.4	14.5	0.12	0.9	0.01
	Traditional LC	183	95.2	15.4			
Discussion Attendance	Active LC	348	97.1	13.8	1.18	0.24	0.11
	Traditional LC	183	95.5	17.4			
PA Grades	Active LC	348	95.9	9.2	1.66	0.1	0.15
	Traditional LC	183	94.3	12.2			
Worksheet Grades	Active LC	348	92.2	8.9	-0.65	0.52	-0.06
	Traditional LC	183	92.8	9.6			
Final Exam	Active LC	348	85.1	16.5	-2.8	0.01*	-0.25
	Traditional LC	183	89.0	12.4			
Overall Grade	Active LC	348	91.8	9.5	-1.71	0.09	-0.16
	Traditional LC	183	93.2	8.4			

Table 16. Least squares regression model fitted with students' *Year*, *Major*, *Prior Experience*, and *Treatment Condition* as independent variables and *Final Exam Score* as dependent variable.

Grouping		coef	N	std err	t	p value
Y-intercept	**const**	86.14	N/A	1.46	59.15	0.00
Year	**Second-Year**	2.75	199	0.97	2.84	<0.01*
	Third-Year	2.21	146	1.13	1.96	0.05
	Fourth-Year	-1.13	57	1.51	-0.75	0.45
	Transfer	4.46	1	8.59	0.52	0.60
Major	**Elec. Engineer.**	-2.11	53	1.36	-1.56	0.12
	Math	-2.55	164	1.07	-2.38	0.02*
	Other	-3.68	160	1.17	-3.13	<0.01*
Prior Experience	**Yes**	7.60	459	1.11	6.83	<0.01*
Treatment	**ALC**	-0.53	348	0.83	-0.64	0.52

Table 17. Impact of each predictor on adjusted r-squared value for final exam grades.

Model	Adj. R^2	Adj. R^2 Diff from Full Model
Full Model	0.159	N/A
Full Model, NO Treatment	0.159	0.0
Full Model, NO Year	0.151	-0.008
Full Model, NO Major	0.144	-0.015
Full Model, NO Prior Experience	0.073	-0.086

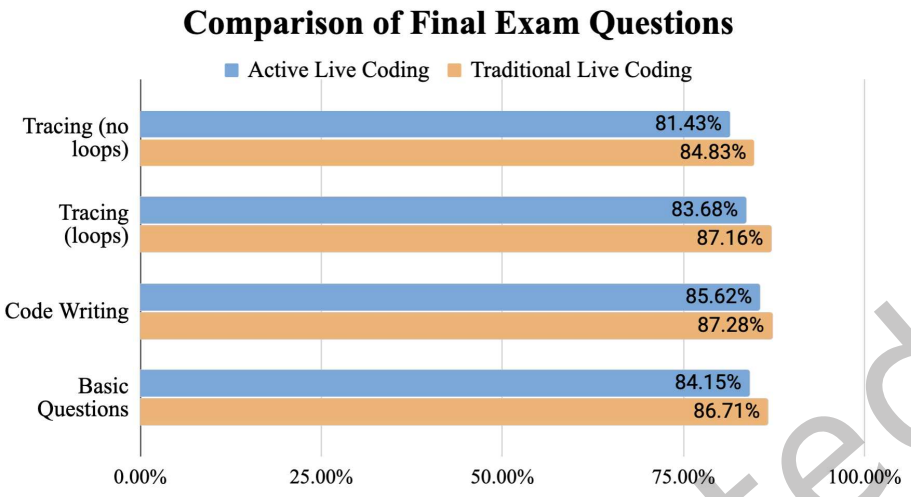

Fig. 1. A comparison of student grades on various types of questions on the Final Exam

on the model's predictive power. Interestingly, there is no difference in the Adjusted R^2 value when we remove Treatment from the model, showing the minimal impact of our treatment compared to the other groupings.

6.2.2 Comparing Student Performance on Code Tracing and Code Writing Questions. Finally, just as Shah et al. had done in their study, we compared student performance across code tracing, code writing, and basic, conceptual questions between the two lecture styles. Figure 1 shows the comparison of student performance, which revealed no statistically significant differences between the two groups.

6.2.3 Comparing Student Learning Gain During Lecture. Our learning gain metric is the proportion of students who incorrectly answered the pre-lecture question incorrectly but correctly answered the corresponding post-lecture question. Since this metric is sensitive to the number of students in the Potential Learning Group (PLG), we compared the rate of correctness of students on the pre-lecture questions for both lecture groups. The rates of correctness were similar throughout the term, with the ALC lecture having an average of 43.1% of students in the PLG and the TLC lecture having 40.8% of students in the PLG.

Table 18. Comparison of students' learning gain.

Lecture Condition	Num Questions	Learning Gain	z stat	p val	Cohen's H
ALC	2768	50.7%	-1.85	0.06	0.06
TLC	1323	53.7%			

Table 18 shows that there is a 3 percentage point difference in the aggregate learning gain throughout the term. The "Num Questions" column represents the total number of *pairs of pre- and post-lecture questions* we analyzed during the quarter from the PLG. According to a z-test of proportions [28], this difference is not statistically significant. The low Cohen's H [11] effect size implies a relatively small magnitude of the difference in the learning gain.

Category	Label	Frequency
Helps with course	helps with PAs	5.5%
	helps with exam review	1.1%
	helps with discussion sections	0.4%
Programming Process	applies lecture material	25.1%
	shows instructor's solution	7.4%
	helps with debugging	2.6%
	improves coding skills	3.7%
Understanding	reinforces understanding	33.6%
	provides immediate feedback	10.0%
	provides incremental learning	0.7%
Active Learning	provides hands on experience	16.6%
	promotes engagement	9.6%
	allows for group learning	4.1%
Negatives	rushes lecture	3.7%
	student gets stuck	3.0%
	repetitive examples	1.5%
Unclear	vague	11.6%

Fig. 2. A comparison of student responses to the question "Please give a brief explanation of your rating on the active coding component" for the Active Live Coding group ($n = 271$).

6.3 RQ3 Results

Our results for RQ3 can be divided into three parts. First, we conducted a thematic analysis on students' perceptions of Active Live Coding. Second, we compared students' perceived benefits of their respective lecture style to detect differences in student perceptions. Third, we conducted an observational study to measure student behavioral engagement during the different lecture styles throughout the term.

6.3.1 Identifying Student Perceptions of Active Live Coding. Figure 2 shows open-ended student responses to the question: "*Please give a brief explanation of your rating on the active coding component.*" The column titled **Label** represents the codes that our research team generated during the coding process and the **Category** column represents a post-analysis grouping of the labels. The percentages do not sum to 100% because each student response could have multiple labels. The categories of perspectives include benefits, such as helping with performance on course components, helping students' programming processes, improving student understanding of concepts, and keeping students engaged with active learning, and drawbacks, such as the directions for the active coding component being too vague and lectures being rushed due to Active Live Coding. The most common response students mentioned was that Active Live Coding "reinforces understanding" (33.6%) of key concepts, which students typically mentioned was due to the "hands on experience" (16.6%) of coding themselves. Students also appreciated that active live coding "applies lecture material" (23.7%) they just saw in class. We also saw that students appreciated how Active Live Coding "provides immediate feedback" (10.0%) on their programming solution since they see other students' approaches and the instructor's solution.

Category	Label	TLC	ALC
Code Comprehension	part-by-part breakdown	7.5%	5.0%
	reference of correct code	25.4%	26.4%
Programming Process	thought process while coding	9.0%	11.4%
	debugging/avoiding errors	38.8%	20.9%
	code writing	8.2%	14.1%
	testing code	2.2%	1.8%
Features of Code Examples	instructor's explanation	14.2%	9.5%
	variations of code	0.7%	1.8%
	predicting output	6.0%	0.5%
Lecture Experience	following along with instructor	7.5%	3.6%
	taking notes	1.5%	0.9%
	group learning	0.7%	0.5%
Application	application of abstract concepts	9.0%	13.6%

Fig. 3. A comparison of student responses to the question "What are some specific things about the live-coding examples that have been helpful for your learning?" for the TLC (n = 134) and ALC (n = 220) groups.

6.3.2 Comparing Perceived Benefits of Active and Traditional Live Coding. Figure 3 shows the comparison of student responses to the question "*What are some specific things about the live-coding examples that have been helpful for your learning?*" A darker green color represents a higher frequency of students that mentioned that label. We generally saw similar frequencies between the two groups, though one prominent difference we noticed was nearly 40% of students in the TLC lecture mentioned debugging as a benefit of the live coding examples, whereas only 20% of students in the ALC lecture noted this. Both of these values are in stark contrast to the work from Shah et al., who found that only 13% of students mentioned debugging as a benefit of live coding.

As mentioned before, one issue with this analysis was that 51 students in the ALC lecture gave a response about active coding rather than Traditional Live Coding. This explains the difference in the responses for Figure 2 (n = 271) and Figure 3 (n = 220). Not only is an interesting finding that 51 of the 271 students mentioned a quality of Active Live Coding when answering a question about Traditional Live Coding, but we saw a new label emerge from these responses. Specifically, students mentioned that they appreciated the level of detail and clarity in the comments and directions that the instructor gave before the active coding component. One student wrote: "[The instructor] writing the comments of what we're supposed to do before doing the code has also been helpful."

6.3.3 Comparing Student Behavioral Engagement During Lecture. Figure 4 shows the comparison of student behavioral engagement across all lecture activities throughout the term. Each percentage in one of the horizontal bars represents the average percent of students that were engaged during that lecture activity during the term. There is no engagement rate for Active Live Coding for the TLC group since there was no active live coding portion in the TLC lecture. In general, the engagement levels were relatively similar for the Pre-Lecture Questions and Worksheet Review. Interestingly, students in the ALC lecture exhibited slightly *higher* engagement during traditional live coding (which happens in both lecture groups) but *lower* engagement during the written notes section.

Figure 5 represents the change in student engagement throughout a lecture based on specific lecture activities. For each ten minute increment into the lecture in Figure 5, we found the most common lecture activity from our

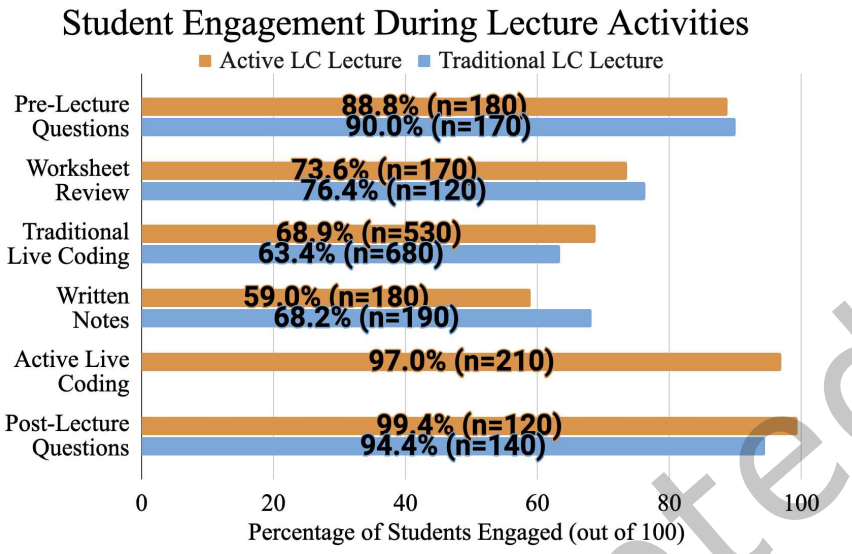

Fig. 4. Comparison of behavioral engagement between the two lecture groups.

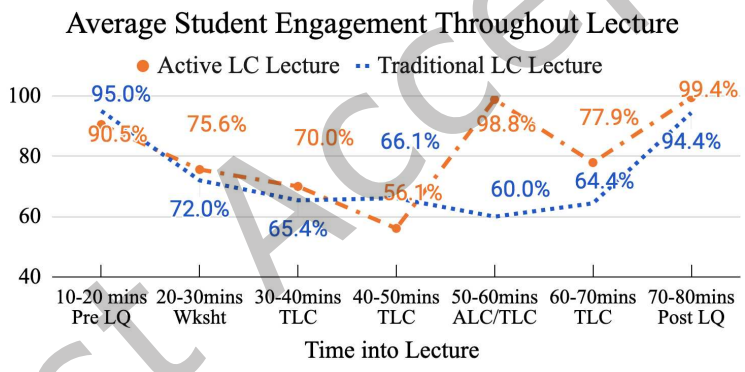

Fig. 5. Average engagement throughout lectures.

observations (i.e., for minutes 10 to 20 into the lecture, the most common lecture activity were the pre-lecture questions; for minutes 20 to 30 into the lecture, the most common lecture activity was worksheet review, etc.). We then calculated the average engagement for that lecture activity *specifically within that ten minute increment.* Traditional live coding, which occurred in both lectures, was the most common lecture activity for both lectures 30 to 50 minutes into the lecture. For the increment between 50 and 60 minutes the most common activity within the ALC lecture was active live coding whereas the most common activity for the TLC lecture continued to be traditional live coding. We see a strong peak for the ALC lecture during the active live coding component, which is unsurprising given that the active coding portion was required for students to complete. Interestingly, the ten minute increment following the active live coding component shows a *higher* engagement level for the ALC group than the TLC group, potentially demonstrating a *persistent engagement effect* of active live coding.

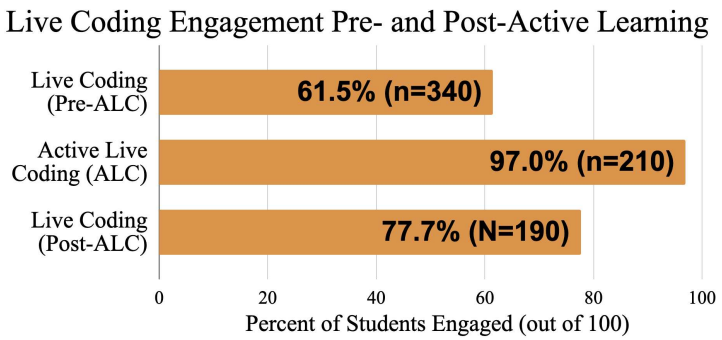

Fig. 6. Student engagement during live coding before and after active live coding in the ALC lecture.

To further explore this persistent engagement effect, we compared student engagement before and after the Active Live Coding component in Figure 6. We specifically compared the traditional live coding components before and after an active live coding component. Figure 6 shows that the engagement rate for traditional live coding *before* ALC was only 61.5%, but this value increased to 77.7% *after* ALC. A two-sample t-test [24] revealed that the difference between these values is significant, with $p < 0.001$ and Cohen's d of 1.16—a large effect size [14].

7 DISCUSSION

7.1 Interpretation of Results

Paragraphs within this subsection were re-ordered to provide better structure for the interpretation of results.

7.1.1 Similar adherence to programming processes (RQ1) and student learning (RQ2). Our quantitative findings related to students' programming processes and learning outcomes generally showed no significant difference between the ALC and TLC groups. Students in both groups showed similar adherence to incremental development (Table 10) and error frequencies (Tables 11 and 12), while being able to produce correct code at similar rates (Table 13). Further, students performed similarly across all major grading components in the course (Tables 15 and 16) and exhibited similar learning gain during lectures (Table 18). Though there was one statistically significant difference from our analysis that showed the TLC students scoring higher on the final exam, our regression analysis of the factors that explain the difference between the two groups showed that the ALC treatment condition had an insignificant association with final exam scores. Though we only showed the regression analysis for the final exam scores, we repeated this analysis on all other comparison items in RQ1 and RQ2. A prevailing theme in all of these analyses is that these other factors—students' year in program, major, and prior programming experience—contributed more to the model's accuracy than our treatment condition. Our main takeaway for RQ1 and RQ2 is that Active Live Coding resulted in similar programming processes and learning outcomes as Traditional Live Coding. Several potential explanations exist for the lack of impact of Active Live Coding. One hypothesis is that the effect of roughly 30 to 40 minutes of active coding and peer discussion per week (i.e., one ALC session in each of the two lectures per week) is marginal compared to the significant amount of course components per week. As mentioned in Section 4.1, students' attend 160 minutes of lecture and 50 minutes of a discussion section per week and complete one programming assignment and worksheet per week. All these other course components represent significant impacts to students' learning beyond the moments of Active Live Coding. Combined with demographic factors such as major, year-in-university, and prior programming experience, these other activities represent a significant amount of learning experiences that may outweigh the

effect of Active Live Coding. A second hypothesis is that Active Live Coding takes up more lecture time than Traditional Live Coding, resulting in less time for an instructor to explain the material. In the TLC lectures, the instructor covered the same material as the Active Live Coding lectures, but had more time to go into more detail in their lessons. Throughout the term, this may result in more time for instructors to answer student questions, explain program errors, and share their thought process. Though the ICAP Framework shows that students learn the material better by engaging with it more actively, the *amount of time* that a learner engages with material may also impact learning gains. Indeed, one of the drawbacks of live coding mentioned by Bruhn and Burton is the greater time commitment of live coding compared to static code examples [6]. A third potential reason for the lack of significant findings is that the Traditional Live Coding lectures engaged students beyond a *passive* engagement level. Since the instructor occasionally prompted the class with a verbal question (i.e., *"what would be printed if I ran the code now?"*, students' engagement level may have become active or constructive. Therefore, the traditional live coding components may already be sufficiently engaging and informative for students.

7.1.2 Differences in students' lecture experience (RQ3). Our findings related to students' lecture experience in Active Live Coding revealed important differences between ALC and TLC. In fact, student responses demonstrated the multiple Cognitive Apprenticeship Methods being applied. Of course, the modeling Method is engaged by Traditional Live Coding, and the responses in Figure 3 show how students' in both lecture groups saw the instructor's programming process and heard the instructor's thought process. Further, the open-ended question related to students' perceptions of ALC, shown in Figure 2, uncovered aspects *specific to Active Live Coding* that are not present in TLC. For example, students discussed how the active coding component "provides a hands on experience" to students to code on their own. This demonstrates the scaffolding Method of Cognitive Apprenticeship as students' are able to complete a simple activity on their own in a low-stakes environment that uses the concepts just taught in lecture. Then, students mentioned being able to discuss their solution with peers, which is captured by the label "allows for group learning" in Figure 2. This aspect of ALC leverages the articulation Method of Cognitive Apprenticeship, as students discuss their approach with peers. Finally, students' mentioned "seeing the instructor's solution" to the coding activity, which specifically engages the reflection Method of Cognitive Apprenticeship, as students' compare their own approach to the instructor's approach. An interesting set of responses from 9.96% of respondents mentioned that ALC "provides immediate feedback" on their solution to the programming task—a mechanism that is simply not present in Traditional Live Coding. The tenet of providing immediate, individual feedback relates to the coaching Method of Cognitive Apprenticeship. In Section 2.1, we did not mention coaching as a Method that is leveraged by Active Live Coding since there is no one-to-one interaction between instructor and student. Instead, there is only a one-to-many feedback channel as the instructor provides the solution to the coding activity. However, despite this limitation of Active Live Coding, it seems students still felt that ALC achieved some of the benefits of the coaching Method.

Another key finding from RQ3 is that ALC improved student engagement but did not impact student learning. A notable category of responses in Figure 2 discussed the engaging nature of Active Live Coding. Our analysis using the BERI protocol revealed an interesting effect of Active Live Coding. Specifically, Figure 5 shows that engagement starts and ends at a high rate, but we saw a lower engagement rate about 30 minutes into the lecture. From about 30 minutes to 70 minutes in the Traditional Live Coding lecture, the engagement rate hovers between 60 to 70 percent as students observe the instructor live coding. However, the large spike due to Active Live Coding resulted in a *persistent engagement effect* where students had a heightened engagement rate even 20 minutes *after* Active Live Coding. A key reason for this heightened engagement effect, which can be partially explained by the findings in Figure 2, is that students can see the instructor's solution and get immediate feedback about whether their approach was correct. In other words, the 20 minutes following Active Live Coding are important for students' to identify whether they were correct and to compare their solution to the instructor's solution.

This *persistent engagement effect* is highlighted by Figure 6, which showed a statistically significant difference in engagement levels before and after Active Live Coding.

7.1.3 Theoretical implications of our results. Overall, our results are unexpected. Based on Cognitive Apprenticeship and the ICAP Framework, one may predict that students' in the Active Live Coding group would exhibit greater adherence to programming processes and learn more than their Traditional Live Coding counterparts. Since Active Live Coding engages more Methods of Cognitive Apprenticeship, we would have expected students in the ALC group to adhere to incremental development and debug errors more efficiently than students in the TLC group. However, the results of RQ1 and RQ2 showed that students exhibited similar programming processes and course performance in the two groups despite their lectures engaging the articulation and reflection Methods of Cognitive Apprenticeship. Previous work on active learning, such as Peer Instruction, in computing education has shown empirical improvements to retention and learning gain [23] and failure rates [22]. These works by Porter et al. showed that students benefited from the peer discussion [23], which is also a critical component of Active Live Coding. However, our results did not show similar results to the findings from Porter et al., motivating a deeper investigation into whether Active Live Coding improves student learning.

Similarly, the ICAP Framework predicts that activities with a higher level of engagement will result in more student learning. However, despite finding a higher engagement rate in ALC, our results from RQ2 did not show an increase in student learning as a result of this higher engagement. In fact, the most granular metric related to student learning during lecture—the learning gain analysis shown in Table 18—showed no statistically significant difference in the learning gain during lecture between the two groups, although the TLC had a higher average learning gain throughout the term. The comparison of learning gain via the pre- and post-lecture questions, which are multiple-choice questions that are code tracing or conceptual questions, may not be the best way to assess student learning for ALC, which asks students to *write code*. However, even the metrics related to correctness on programming tasks in Table 13 do not show the ALC students outperforming the TLC students. Overall, we saw almost no impact of ALC on student learning, raising questions about why the higher level of engagement did not translate to higher student learning.

7.2 Threats and Limitations

The main factor that threatens the internal validity of our work is the experience of the instructor who taught both groups. The instructor has been an instructor for six years and has regularly used Traditional Live Coding all six years. In contrast, this was the instructor's first time ever using Active Live Coding. Upon reflection, the instructor mentioned that the timing of the Active Live Coding lecture was difficult to manage the active coding and peer discussion portion takes roughly 10 of the 80 total minutes. As a result, the Traditional Live Coding students may have benefited from the instructor's experience while the Active Live Coding students may have suffered from the instructor's lack of experience with the lecture style.

A second threat to internal validity is the difference between the two lectures in terms of the time of day and the student makeup. For example, one issue we noticed is that many more Math majors were enrolled in the Traditional Live Coding lecture because a required Math course was offered at 9:30am on Tuesdays and Thursdays—the same times as the Active Live Coding lecture. Though our analysis took this specific factor into account, there could easily be other selection biases we did not detect. Although students did not know that there would be any difference between lecture sections or that the 9:30am lecture would be ALC while 11am would be TLC, we were unable to randomly assign students to the lecture sections, as Raj et al. had done [25].

A third threat to internal validity is that we did not track whether students completed the active live coding activity. While the classroom observer consistently saw high engagement during this part of lecture, we are unable to find out how many students actually attempted the activity. Our results might be different if students earned a grade for the active live coding activity, or if the activity was graded on correctness.

The main factor that threatens the external validity of this work is the instructor effect. Different instructors may see varying levels of student success. In fact, one of the takeaways from Porter et al.'s work on Peer Instruction is that different instructors saw varying levels of benefits of using Peer Instruction [22], which may also be the case for Active Live Coding or even Traditional Live Coding. Therefore, replication studies with different instructors can help create a solid basis of empirical findings related to the impact of Active Live Coding.

7.3 Future Work

This work has occurred after a long line of prior work to evaluate traditional live coding [25, 27, 30, 33, 36, 39]. The overwhelming finding from these works is that there has been no significant difference between traditional live coding and static-code examples. Similarly, our findings in this study indicate a similar effect of Active and Traditional Live Coding compared to Traditional Live Coding. A useful avenue of future work could investigate *why* Traditional Live Coding and Active Live Coding do not result in improved student learning. It may be the case that some factors during lecture, such as student engagement, distractions, cognitive load, or other factors, may mitigate any potential learning gain from live coding. A qualitative approach to understanding how students process a live coding example may help us understand why we have not seen benefits from the activity. Similarly, our experimental design considers a course-long intervention with programming process data and course outcome data collected from summative assessments. Future work may consider alternative data to analyze that may shed light on the differences between the two pedagogical techniques.

We have not found any existing empirical evaluations of Active Live Coding. Our experiment only investigates the impact of a single instructor using Active Live Coding throughout the term. As a result, future works may use methods such as lab studies or term-long interventions with a different instructor to further investigate the impact of Active Live Coding. Additional analyses on Active Live Coding may also explore other outcomes that we did not analyze in our study, such as students' sense of belonging in the course and in the computer science major. A potential benefit of Active Live Coding, along with other active learning techniques that encourage peer discussion, is that students have a stronger sense of community in the course, resulting in greater feelings of belonging.

8 CONCLUSION

While our study is just a single data point in the broader literature related to live coding and active learning, the findings of this study can help inform an instructor of the potential effects of using Active Live Coding. Specifically, Active Live Coding seems to impart similar learning and adherence to programming processes as Traditional Live Coding while also promoting student engagement and peer discussions. Students also mentioned some unique affordances of Active Live Coding, such as providing immediate feedback on the correctness of their programming solution, which is not present in a Traditional Live Coding study. Therefore, although we did not detect empirical benefits of Active Live Coding compared to Traditional Live Coding, instructors may expect students to have similar perceptions of and engagement with Active Live Coding should they choose to adopt this pedagogy.

REFERENCES

[1] Leona S. Aiken, Stephen G. West, and Steven C. Pitts. 2003. *Multiple Linear Regression.* John Wiley & Sons, Ltd, Chapter 19, 481–507. https://doi.org/10.1002/0471264385.wei0219 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471264385.wei0219

[2] Anon Authors. [n. d.]. A Comparison of Student Behavioral Engagement in Traditional Live Coding and Active Live Coding Lectures. In *Proceedings (Anon).* Anon, Location. https://bit.ly/anon-engagement-paper

[3] Brett A. Becker. 2016. A New Metric to Quantify Repeated Compiler Errors for Novice Programmers. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16).* Association for Computing Machinery, New York, NY, USA, 296–301. https://doi.org/10.1145/2899415.2899463

[4] Jens Bennedsen and Michael E. Caspersen. 2005. Revealing the Programming Process. *SIGCSE Bull.* 37, 1 (feb 2005), 186–190. https://doi.org/10.1145/1047124.1047413

[5] Naomi R. Boyer, Sara Langevin, and Alessio Gaspar. 2008. Self Direction & Constructivism in Programming Education. In *Proceedings of the 9th ACM SIGITE Conference on Information Technology Education (SIGITE '08)*. Association for Computing Machinery, New York, NY, USA, 89–94. https://doi.org/10.1145/1414558.1414585

[6] Russel E. Bruhn and Philip J. Burton. 2003. An Approach to Teaching Java Using Computers. *SIGCSE Bull.* 35, 4 (dec 2003), 94–99. https://doi.org/10.1145/960492.960537

[7] Michelene T. H. Chi. 2009. Active-Constructive-Interactive: A Conceptual Framework for Differentiating Learning Activities. *Topics in Cognitive Science* 1, 1 (2009), 73–105. https://doi.org/10.1111/j.1756-8765.2008.01005.x arXiv:https://onlinelibrary.wiley.com/-doi/pdf/10.1111/j.1756-8765.2008.01005.x

[8] Michelene T. H. Chi and Ruth Wylie. 2014. The ICAP Framework: Linking Cognitive Engagement to Active Learning Outcomes. *Educational Psychologist* 49, 4 (2014), 219–243. https://doi.org/10.1080/00461520.2014.965823 arXiv:https://-doi.org/10.1080/00461520.2014.965823

[9] Allan M. Collins, John Seely Brown, and Susan E. Newman. 1988. Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing, and Mathematics. *Knowing, Learning, and Instruction* 8, 1 (1988), 2–10. https://doi.org/10.5840/thinking19888129

[10] Louis Deslauriers, Ellen Schelew, and Carl Wieman. 2011. Improved Learning in a Large-Enrollment Physics Class. *Science* 332, 6031 (2011), 862–864. https://doi.org/10.1126/science.1201783 arXiv:https://www.science.org/doi/pdf/10.1126/science.1201783

[11] Rajarshi Dey and Madhuri S. Mulekar. 2018. *Effect Size as a Measure of Difference Between Two Populations.* Springer New York, New York, NY, 715–726. https://doi.org/10.1007/978-1-4939-7131-2_110195

[12] Edstem. 2023. Edstem. https://edstem.org/

[13] Jennifer A Fredricks, Phyllis C Blumenfeld, and Alison H Paris. 2004. School engagement: Potential of the concept, state of the evidence. *Review of educational research* 74, 1 (2004), 59–109.

[14] David C. Funder and Daniel J. Ozer. 2019. Evaluating Effect Size in Psychological Research: Sense and Nonsense. *Advances in Methods and Practices in Psychological Science* 2, 2 (2019), 156–168. https://doi.org/10.1177/2515245919847202

[15] Alessio Gaspar and Sarah Langevin. 2007. Active learning in introductory programming courses through Student-led " live coding " and test-driven pair programming. https://api.semanticscholar.org/CorpusID:11945824

[16] Gradescope. 2024. Gradescope. https://www.gradescope.com/

[17] Gunnar Harboe, Jonas Minke, Ioana Ilea, and Elaine M. Huang. 2012. Computer support for collaborative data analysis: augmenting paper affinity diagrams. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. Association for Computing Machinery, New York, NY, USA, 1179–1182. https://doi.org/10.1145/2145204.2145379

[18] Michael Kölling and David J. Barnes. 2004. Enhancing Apprentice-Based Learning of Java. *SIGCSE Bull.* 36, 1 (mar 2004), 286–290. https://doi.org/10.1145/1028174.971403

[19] Erin S. Lane and Sara E. Harris. 2015. A New Tool for Measuring Student Behavioral Engagement in Large University Classes. *Journal of College Science Teaching* 44, 6 (2015), 83–91. http://www.jstor.org/stable/43632000

[20] Lauren E. Margulieux and Richard Catrambone. 2016. Using Subgoal Learning and Self-Explanation to Improve Programming Education. *Cognitive Science* (2016). https://api.semanticscholar.org/CorpusID:9558170

[21] John Paxton. 2002. Live Programming as a Lecture Technique. *J. Comput. Sci. Coll.* 18, 2 (dec 2002), 51–56.

[22] Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 177–182. https://doi.org/10.1145/2445196.2445250

[23] Leo Porter, Cynthia Bailey Lee, Beth Simon, and Daniel Zingaro. 2011. Peer instruction: do students really learn from peer discussion in computing?. In *Proceedings of the Seventh International Workshop on Computing Education Research (ICER '11)*. Association for Computing Machinery, New York, NY, USA, 45–52. https://doi.org/10.1145/2016911.2016923

[24] Harry O. Posten. 1984. Robustness of the Two-Sample T-Test. In *Robustness of Statistical Methods and Nonparametric Statistics*, Dieter Rasch and Moti Lal Tiku (Eds.). Springer Netherlands, Dordrecht, 92–99. https://doi.org/10.1007/978-94-009-6528-7_23

[25] Adalbert Gerald Soosai Raj, Pan Gu, Eda Zhang, Arokia Xavier Annie R, Jim Williams, Richard Halverson, and Jignesh M. Patel. 2020. Live-Coding vs Static Code Examples: Which is Better with Respect to Student Learning and Cognitive Load?. In *Proceedings of the Twenty-Second Australasian Computing Education Conference (ACE'20)*. Association for Computing Machinery, New York, NY, USA, 152–159. https://doi.org/10.1145/3373165.3373182

[26] Adalbert Gerald Soosai Raj, Jignesh M. Patel, Richard Halverson, and Erica Rosenfeld Halverson. 2018. Role of Live-Coding in Learning Introductory Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research (Koli Calling '18)*. Association for Computing Machinery, New York, NY, USA, Article 13, 8 pages. https://doi.org/10.1145/3279720.3279725

[27] Marc J. Rubin. 2013. The Effectiveness of Live-Coding to Teach Introductory Programming. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 651–656. https://doi.org/10.1145/2445196.2445388

[28] Randall E. Schumacker. 2023. *Learning Statistics Using R.* SAGE Publications, Inc., 55 City Road, London, Chapter 12. https://doi.org/10.4135/9781506300160

[29] Ana Selvaraj, Eda Zhang, Leo Porter, and Adalbert Gerald Soosai Raj. 2021. Live Coding: A Review of the Literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 164–170. https://doi.org/10.1145/3430665.3456382

[30] Anshul Shah, Vardhan Agarwal, Michael Granado, John Driscoll, Emma Hogan, Leo Porter, William Griswold, and Adalbert Gerald Soosai Raj. 2023. The Impact of a Remote Live-Coding Pedagogy on Student Programming Processes, Grades, and Lecture Questions Asked. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V.1 (ITiCSE '23)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3587102.3588846

[31] Anshul Shah, Fatimah Alhumrani, William G. Griswold, Leo Porter, and Adalbert Gerald Soosai Raj. 2024. A Comparison of Student Behavioral Engagement in Traditional Live Coding and Active Live Coding Lectures. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 513–519. https://doi.org/10.1145/3649217.3653537

[32] Anshul Shah, Michael Granado, Mrinal Sharma, John Driscoll, Leo Porter, William Griswold, and Adalbert Gerald Soosai Raj. 2023. Understanding and Measuring Incremental Development in CS1. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education (SIGCSE '23)*. Association for Computing Machinery, New York, NY, USA, 7. https://doi.org/10.1145/3545945.3569880

[33] Anshul Shah, Emma Hogan, Vardhan Agarwal, John Driscoll, Leo Porter, William G. Griswold, and Adalbert Gerald Soosai Raj. 2023. An Empirical Evaluation of Live Coding in CS1. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1 (ICER '23)*. Association for Computing Machinery, New York, NY, USA, 476–494. https://doi.org/10.1145/3568813.3600122

[34] Anshul Shah and Adalbert Gerald Soosai Raj. 2024. A Review of Cognitive Apprenticeship Methods in Computing Education Research. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1202–1208. https://doi.org/10.1145/3626252.3630769

[35] Stepik. 2024. Stepik. https://stepik.org/

[36] Sheng-Rong Tan, Yu-Tzu Lin, and Jia-Sin Liou. 2016. Teaching by demonstration: programming instruction by using live-coding videos. In *Proceedings of EdMedia + Innovate Learning 2016*. Association for the Advancement of Computing in Education (AACE), Vancouver, BC, Canada, 1294–1298. https://www.learntechlib.org/p/173121

[37] Aki Vehtari, Andrew Gelman, and Jonah Gabry. 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing* 27 (2017), 1413–1432. Issue 5. https://doi.org/10.1007/s11222-016-9696-4

[38] Anne Venables, Grace Tan, and Raymond Lister. 2009. A Closer Look at Tracing, Explaining and Code Writing Skills in the Novice Programmer. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop (ICER '09)*. Association for Computing Machinery, New York, NY, USA, 117–128. https://doi.org/10.1145/1584322.1584336

[39] Andrea Watkins, Craig S. Miller, and Amber Settle. 2024. Comparing the Experiences of Live Coding versus Static Code Examples for Students and Instructors. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 506–512. https://doi.org/10.1145/3649217.3653562

[40] Andrea Watkins, Amber Settle, Craig S. Miller, and Eric J. Schwabe. 2025. Live But Not Active: Minimal Effect with Passive Live Coding. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1190–1196. https://doi.org/10.1145/3641554.3701786

[41] Benjamin L. Wiggins, Sarah L. Eddy, Daniel Z. Grunspan, and Alison J. Crowe. 2017. The ICAP Active Learning Framework Predicts the Learning Gains Observed in Intensely Active Classroom Experiences. *AERA Open* 3, 2 (2017), 2332858417708567. https://doi.org/10.1177/2332858417708567 arXiv:https://doi.org/10.1177/2332858417708567

[42] Marvin Willerman and Richard A. Mac Harg. 1991. The concept map as an advance organizer. *Journal of Research in Science Teaching* 28, 8 (1991), 705–711. https://doi.org/10.1002/tea.3660280807 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/tea.3660280807

[43] Daniel Zingaro and Leo Porter. 2015. Tracking Student Learning from Class to Exam using Isomorphic Questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 356–361. https://doi.org/10.1145/2676723.2677239

Table 19. Final code book for perceptions of Active Live Coding.

Label	Description
debugging	1. ALC helps with catching and fixing errors
hands on experience	1. The act of writing the code while still learning the concepts 2. How writing code helps them being engaged with the class material
reinforce understanding	1. ALC helps with understanding or learning course material 2. How ALC helps reinforce the material through practice 3. check overall understanding
prior experience	1. When the student has previous knowledge of course material
engagement	1. feeling engaged in the class 2. Being more focused during lecture
thought process	1. Seeing/hearing instructor's thought process + general breakdown of the code 2. Student mentioning the instructor
application	1. Applying course material while writing code 2. When students get to see their execution of the code using the material they just learned
community	1. feeling included, part of the community of the class
improves coding skills	1. General statements on how it improves the ability to code for future endeavors 2. When learning syntax 3. When it helps with overall coding skills
repetitive	1. When the student is ahead in course material (PA is already done) 2. When it's too similar to other general coding they've done
helps with PA	1. ALC helps with preparing for the programming assignments
helps with discussion	1. ALC helps with preparing for the discussion sections of the course
rushed	1. Feeling too overwhelmed with course material 2. The allocated time for ALC feels too short 3. Feeling that the ALC section took time away from conceptual explanation time
disrupts lecture	1. Breaks the flow of the lecture 2. very similar to rushed (only one response)
gets stuck	1. Feeling furstrated on not being able to code 2. Not getting support when struggling due to feeling inadequate 3. When the student is behind in material so they aren't able to code properly
immediate feedback	1. Getting immediate help of course staff when confused 2. Students getting feedback on their code after the ALC section is over
incremental learning	1. Learning one step at a time 2. breakdown the challenge to more manageable pieces
helps with exams	1. ALC helps with preparing for the exams

Table 20. Final code book for comparison of perceived benefits of Traditional Live Coding.

Label	Description
part-by-part breakdown	1. Explaining code line by line 2. Student mentions seeing individual parts of the code 3. Labeling or color coding separate components of a program 4. Breaking down a program 5. Making the program more simple to understand
reference of correct code	1. Student mentions using it to guide other similar activities 2. See what code is supposed to look like 3. Thinking about how this code could be modified to do something similar 4. Examples of the correct code for a concept
general code understanding	1. Understanding how the code works in general 2. Useful to review for understanding
thought process while coding	1. Learning the problem solving process 2. Understanding why the professor writes certain lines of code
debugging/avoiding errors	1. Identifying and understanding common errors 2. Process of fixing errors 3. Seeing errors/unexpected output 4. Showing where code can go wrong
code writing	1. Modeling the process of writing code 2. How to approach writing code from scratch 3. Seeing the step-by-step process
testing code	1. Learning how to test the correctness of code 2. Understanding why a test passed or failed 3. Seeing examples of test cases
instructor's explanation	1. Live commentary on code 2. Explanation of code 3. Thorough answers to questions from students
variations of code	1. Showing different variations/changes in a code example 2. Showing trial and error process
predicting output	1. Student mentions enjoying trying to guess the output 2. The instructor asking students to guess what the code will output
seeing output	1. Seeing output of code examples along with them
following along with instructor	1. The ability to follow along with code as it is written live
taking notes	1. Taking notes to reinforce understanding
group learning	1. Suggestions from classmates who have better understanding 2. Students in class giving suggestions for next coding steps
application of concepts	1. Seeing concepts immediately applied during lecture