Understanding and Improving Student Note-Taking in Live Coding Lectures

Daniel Manesh
Virginia Tech
Blacksburg, Virginia, USA
danielmanesh@vt.edu

Tong Wu
Virginia Tech
Blacksburg, Virginia, USA
tongw@vt.edu

Yan Chen
Virginia Tech
Blacksburg, Virginia, USA
ych@vt.edu

Sang Won Lee
Virginia Tech
Blacksburg, Virginia, USA
sangwonlee@vt.edu

Abstract

Background and Motivation. Live coding is a common pedagogical technique where instructors write code in real time during lectures. For students, the main drawbacks of live coding are that it can feel too fast and it can be difficult to take notes.

Objectives. Our work seeks to improve the student experience in live coding lectures by: (1) understanding how instructors expect students to take notes and what challenges students face in doing so; and (2) investigating whether a specialized note-taking tool can help students keep up with the pace of the lecture and take better notes.

Methods. Based on interviews with instructors who use live coding (n=10), we designed a simple note-taking interface consisting of a rich text editor which allows students to take snapshots of the instructor's code. We conducted a within-subjects lab experiment (n=57) comparing our interface with a traditional code editor during two 15-minute live coding lectures. We used quizzes and surveys to assess learning, mental workload, and student perceptions, and analyzed students' notes to determine how much information was captured from the lecture.

Findings. In the experimental condition, NASA-TLX surveys indicated a significantly lower mental workload and students reported that they could more easily keep up with the lecture. Additionally, students perceived their notes to be more useful and our analysis revealed that the notes had significantly more information from the lecture and provided more context for copied code. Despite these benefits, we did not see a significant difference in learning between the two conditions.

Implications. Our results show that during live coding lectures, we can decrease student mental workload and increase the quality of notes by providing an interface which (1) allows capturing the instructor's code without having to type it out; and (2) maintains a clear visual distinction between code snippets and other text. Future work may examine if such an interface can lead to learning gains over long-term use in the classroom.

CCS Concepts

• **Social and professional topics → Computing education**; • **Human-centered computing** → *User interface design*; *Empirical studies in HCI*.

Keywords

live coding, note-taking

ACM Reference Format:
Daniel Manesh, Tong Wu, Yan Chen, and Sang Won Lee. 2025. Understanding and Improving Student Note-Taking in Live Coding Lectures. In *ACM Conference on International Computing Education Research V.1 (ICER 2025 Vol. 1), August 03–06, 2025, Charlottesville, VA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3702652.3744224

1 Introduction

Live coding in programming classes is a pedagogical technique where instructors teach programming concepts by writing code live in front of learners [50, 58, 61]. Instructors typically share their code editor through a projection screen in a classroom and explain their thought process by speaking aloud while writing code [57]. By revealing the instructor's programming process, live coding is seen as a valuable way to demonstrate tacit programming knowledge, such as debugging, incremental coding, and iterative testing [4, 58]. Live coding is typically well received by students, who often report increased engagement and a preference for live coding over other types of lecture delivery [17, 22, 68]. For these reasons, many consider live coding an effective lecturing technique and advocate for its use in the classroom [6, 9, 17, 20, 61, 75]. Live coding is also widely used in recorded videos, from streamers who broadcast their development process [11, 18], to recorded lectures on YouTube[1], to some of the most popular online programming courses, such as EdX's CS50 series[2] [1].

Live coding is not without drawbacks. For students, the main drawbacks of live coding are that it can be difficult to keep up with the pace of coding [17, 63, 66] and that it can be difficult to take notes [66, 69]. Researchers have called for future work to investigate how to maintain the benefits of live coding while minimizing these costs [17, 66].

[1]For example: https://youtu.be/EHi0RDZ31VA
[2]See: https://youtu.be/JP7ITIXGpHk

In this work, we investigate if the drawbacks of live coding can be mitigated by changing how students take notes during a live coding lecture. Some suggest that during live coding lectures, it will be beneficial for students to type along on their own computer to copy the instructor's code [23, 57]. Although a recent study found that the degree to which students type along is associated with different quiz outcomes [74], empirical studies of live coding instruction typically do not control for student note-taking behavior [57, 64, 66, 74]. There remains a need to better understand how students can most effectively engage with live coding lectures based on their individual needs and the challenges inherent to the medium.

To address this need, we set out to answer two research questions:

- **RQ1:** What do instructors expect students to do during live coding lectures and what types of challenges do they observe students facing?
- **RQ2:** Does a note-taking interface designed for live coding lead to better outcomes in terms of student learning, mental workload, and quality of notes?

To address RQ1, we interviewed ten instructors who use live coding in the classroom to understand their experiences, how they incorporate live coding into their lectures, and the perceived challenges faced by students. One key finding was that instructors were ambivalent about whether students should code along with them. Instructors felt copying down their code might aid students' memory and allow them to experiment with the code, but also acknowledged that typing along could be distracting for students and could lead to falling behind.

To address RQ2, we designed a simple note-taking interface for live coding based on the findings from our interviews. Our interface provides a rich text editor for taking notes, which allows interleaving text with embedded code snapshots taken from a mirrored version of the instructor's code editor. In a within-subjects lab study (n=57), we compared this mode of interaction with a baseline, where participants took notes in a typical code editor where they had to type out the instructor's code to copy it down.

Our results suggest that our experimental system helped students overcome two main challenges of live coding: it helped students take more comprehensive notes during live coding lectures and helped students keep up with the pace of lecture by reducing the mental workload required to copy down the instructor's code. Despite addressing these drawbacks, we did not find a significant difference in learning between the two conditions. Still, our results indicate that instructors who use live coding may potentially improve their students' experience by providing an interface which allows students to capture the instructor's code without typing it out and which has a clear visual distinction between code and other text. Future work can further explore the effectiveness of our approach in a classroom setting.

2 Related Works

2.1 Live Coding in CS Classrooms

Though the term "live coding" has alternate meanings in other domains [60], in the realm of CS education, it refers to a lecture technique where the instructor writes code live in front of students [63]. Live coding can be used to deliver the entire lecture [50] or can be reserved for example problems and interleaved with other modes of delivery [8]. While live coding, the instructor typically projects their screen and thinks aloud to share their thought process [57]. Most often, live coding is instructor-led [63], which is the mode we focus on in this paper. However, other approaches more directly involve students by having them actively guide the instructor's coding process [67] or even by asking the students themselves to code in front of the class [19, 20].

Many papers report the benefits of live coding, typically from the student's perspective. Students find live coding effective in many ways and often prefer it over other lecture techniques [22, 68]. Most notably, students perceive it as beneficial to watch instructors live coding, accessing the step-by-step procedure of problem solving [3, 17, 25]. Instructors, as practitioners, consider live coding a way to reveal their thought processes to students incrementally [76]. Brown and Wilson claim that live coding allows instructors to be more responsive to "what-if" questions [6]. In addition, the dynamic nature of live coding is thought to make it easier for students to pay attention and stay engaged [61, 63]. Relatedly, students' perceived cognitive load was reported to be lower in lectures using live coding compared to lectures using static code examples [57].

2.1.1 *Elusive Learning Gains.*

Despite the ample literature that suggests the potential benefits of live-coding lectures, researchers have repeatedly failed to find a learning gain from live-coding lectures compared to using static code examples [17, 57, 61, 64, 66, 74]. Given the myriad perceived benefits of live coding, why do we see inconclusive learning gains?

One possible explanation is that the benefits of live coding may not be easily captured by learning measurements. Researchers and practitioners have suggested that live coding may be well-suited for transferring procedural knowledge, such as debugging [6, 50, 58], and less effective for transferring conceptual knowledge [71]. While one study observed live coding led to better performance on programming-oriented projects [61], a more recent study found that live coding led to no significant difference in the programming process seen on proctored coding challenges [66].

Another explanation is that the drawbacks of live coding dampen its benefits. For instructors, a main drawback is that it can be time-consuming to prepare and deliver live coding lectures [50, 73]. For students, the main drawbacks of live coding are that it can be difficult to take notes and difficult to keep up with the lecture [63, 66]. The main aim of this work is to ameliorate these two drawbacks for students, which we further contextualize within existing theories of note-taking in Section 2.2.

2.1.2 *Improving Live Coding for Learners.*

Relatively little research has examined tools to aid learners in live coding lectures. Most work focuses on video recordings of live coding lectures, providing features such as slower playback speeds [2], video annotations [36], or embedded coding exercises [49]. One system, Storyteller, augments a live coding recording by synchronizing the video with a textual tutorial authored by the instructor [39]. Another system, Scrimba[3], allows learners to view recorded live coding lectures and create experimental forks of the instructor's code at any time. Although Scrimba primarily focuses on asynchronous lessons, it

[3]https://v2.scrimba.com/

can be used synchronously in a classroom setting, where its code exploration features are perceived as useful by students [23, 24]. We note that none of the above work supports note-taking in a live coding context.

2.2 Note-Taking and Live Coding

Note-taking is ubiquitous in higher education. Instructors deliver lectures and expect students to take notes, capturing the fleeting information that may not be readily available elsewhere [54]. Most students perceive note-taking as useful for their learning and studies suggest most students take notes during lectures [45, 54]. Research supports that note-taking is beneficial for learning [31, 35], and many studies aim to understand how those benefits might be mediated by learner characteristics [16], lecture delivery [34], note-taking procedures [5], and note-taking tools [44, 48].

2.2.1 The Benefits of Note-taking. One of the benefits of note-taking is its *storage* function [15]. That is, notes provide a record of the lecture information for later review. There is strong evidence that reviewing notes benefits learning [31, 35] and the storage function of note-taking is widely accepted as beneficial [54].

Another benefit of note-taking is its *encoding* function. That is, the note-taking process itself is thought to promote learning. The evidence for note-taking's encoding benefits is mixed [54], though a meta-review showed a small positive effect [34]. Note-taking can be cognitively demanding [7, 55], and consequently, benefits of encoding appear to be sensitive to a variety of factors, such as the speed and information density of the lecture [31, 54] and the note-taking procedure used [31, 34].

Finally, note-taking may help students stay focused. One study found that 62% of students surveyed took notes to help them pay attention in class [45]. Several other studies suggest that note-taking may play a role in reducing mind-wandering during lectures [30, 37, 78].

2.2.2 Taking Notes in Live Coding Lectures. In a typical live coding lecture, an instructor might deliver information by speaking, writing code, and writing text comments [57]. Effective note-taking is associated with generative activities like summarization and paraphrasing, as opposed to verbatim copying [5, 38, 44, 48]. While students can summarize or paraphrase the spoken words and text comments from a live coding lecture, there are practical reasons to have an exact copy of the instructor's code (e.g., to be able to reproduce and run the code without introducing errors). Thus, a compromise note-taking strategy for live coding might involve taking a verbatim copy of the instructor's code while selectively rewording and summarizing the other verbal information from the lecture.

This work considers two different approaches for capturing the instructor's code in notes: typing it out or using copy-and-paste operations. On one hand, typing out code might act as a beneficial syntax drill [21] and can potentially aid with recall [7]. On the other hand, note-taking is already a cognitively demanding activity, and the additional effort required to type out code could lead to cognitive overload and interfere with learning [29, 47, 55]. Our study helps clarify whether typing out or copy-and-pasting the instructor's code makes a difference.

There are other note-taking possibilities for live coding we do not address in this study. For example, students could be given a copy of the instructor's code to annotate, similar to how students might draw on PowerPoint slides. While annotation has shown promise for sensemaking about code [28], this approach may be less appropriate for live coding, where annotations can quickly become outdated as the code evolves. Another option is that students could write down the instructor's code using pen and paper. While one study showed longhand notes were superior to typed notes [48], others show that this may not always be the case [38, 44]. For a live coding lecture, writing out code with pen and paper may be too time-consuming to be practical, and students cannot run their code if it is on paper.

3 Study 1: Instructor Interviews

In order to deepen our understanding of live coding practices, we conducted a preliminary study consisting of 10 semi-structured interviews with CS instructors who regularly incorporate live coding into their lectures. The main goals of the interviews were: 1) to understand what instructors expected students to do during live coding lectures; and 2) to understand instructors' opinions of the difficulties students faced in live coding lectures. The study procedure, discussed below, was approved by our university's Institutional Review Board. Our findings from these interviews highlighted nuanced difficulties in note-taking and informed the design and implementation of our note-taking tool and lab study.

3.1 Method

We interviewed 10 instructors who had experience with live coding in CS lectures. Participants included professors with research responsibilities (n=2), graduate students (n=2), a high school teacher (n=1), and several other university-level teaching faculty, lecturers, and adjuncts (n=5). Their years of teaching experience ranged from a single semester to 34 years, with an average of 13.9 years. More information on the participants is found in Table 1.

The first round of participants was recruited through advertising to our university's mailing lists, and a second round was recruited through open calls to social media. Participants filled out a screening survey which defined live coding and asked them about their teaching experience. Based on their responses, we reached out to eligible participants (i.e., those with experience live coding) to schedule interviews. Interviews were conducted via Zoom and lasted between 70-90 minutes. The first author conducted the interviews, occasionally accompanied by another author.

We began the interviews by asking participants how they used live coding in their lectures. We also discussed what they believed students should be doing during their lectures, what challenges they observed students facing, and how they, themselves, prepared for the lecture.

We transcribed recordings of the interviews and conducted a thematic analysis [12]. We began with an initial round of open coding on the first six interviews. Through constant comparisons of our codes, we developed concepts which we recorded in analysis memos [13], which provided structure for our initial themes. Subsequently, we conducted a round of focused coding on all 10

Table 1: The background information of instructors from the interview study. The courses listed refer only to the courses instructors have recently taught using live coding and which we talked about in our interviews.

	Years Teaching	Course
T01	34	Intro Python; Intro Java
T02	7	Intro Python
T03	18	Databases; Data Structures and Algorithms
T04	16	Intro Data Science
T05	2	Java; Computer Organization
T06	24	Intro Data Science
T07	4	Intro Java
T08	25	Intro Java
T09	0.5	Intro Python
T10	8	Intro MATLAB

transcripts to further develop and iterate on our themes, which are presented in 3.2.

3.2 Results

From our analysis, we identified three themes which inform how we can support student note-taking in live coding lectures. Throughout this section, we will refer to our 10 interview participants as T01, T02, etc.

3.2.1 The Costs and Benefits of Typing Along. There was no consensus among instructors as to whether or not students should type along to copy down the instructor's code. As for the benefits, many instructors felt typing along would help students engage with and remember the lecture material (T01, T02, T03, T04, T06, T08, T10). As T06 said, "*when you write it down yourself, you remember it better than when you just watch somebody else.*" At the same time, some instructors qualified that they were not sure to what extent it was helpful, with T02 going so far as to say "*I don't think that they're learning very much if they're just copying,*" though clarifying "*I'm not saying it's like, a total loss or I wouldn't be teaching this way.*"

As for the drawbacks of typing along, instructors felt it could sometimes be distracting to students who may fall behind spending too much time and effort typing. For example, T01 and T04 both described having to repeat themselves while students tried to catch up with typing the instructor's code. T05 and T06 described students asking them to scroll back to previous code they did not copy in time, which T05 found particularly disruptive. In fact, T05 felt typing along was distracting enough that they recommended against it for their students:

> *(T05) I'd rather you be paying attention to what I'm saying and understanding what I'm doing than trying to frantically copy down every single ampersand, comma, double quote, and whatever.*

Even instructors who did value typing along would, at times, steer their students away from it, asking them to temporarily stop typing and instead focus on the instructor (T08, T10).

Instructors found that students often introduce errors in the code when they copy from the instructor. While some instructors viewed this as a good learning opportunity, others felt it was just another source of distraction.

> *(T02) It's an act of learning, like, oh, I forgot my parentheses or I, you know, need to make sure that I have spacing correct.*

> *(T05) They're tracking down their error for the rest of the lecture instead of paying attention.*

Many instructors (T01, T02, T04, T05, T06, T10) attempt to mitigate this issue by releasing their completed code after lectures, reasoning that students will worry less about correcting errors during lectures if they know they will have access to error-free code afterward.

Key takeaway: While typing along is perceived as an active, engaging way to follow a lecture, instructors also find it can be an unwelcome distraction for students. Instructors are uncertain whether the benefits outweigh the costs.

3.2.2 Exploration through Writing Code. Several instructors (T01, T02, T03, T06) felt that the incremental process of live coding was well-suited for student exploration, and considered this a strength of live coding.

> *(T06) [live coding] allows for a lot more interactive exploration. So, they have their own copy. [...] it gives them the ability to try the code and try it a little differently than what I did.*

Thus, live coding might encourage spontaneous *tinkering*, where students try out changes to the instructor's code without being prompted by the instructor to do so. Several instructors welcomed this type of tinkering, considering it an acceptable or even desirable behavior for students during their lectures (T01, T02, T05, T06, T08, T10). T05 and T06 thought having access to the instructor's code live during lectures might make students feel safe to experiment with the code, knowing they could easily revert to the instructor's code if necessary.

Most participants also mentioned giving students explicit coding tasks to complete as part of a lecture. Instructors described a smooth segue from walking through examples using live coding and then asking students to try a similar problem on their own (T01, T03, T06). These code exercises not only provided value for students to practice writing code, but instructors also found pedagogical value in sharing various student solutions with the class (T01, T02, T03, T04, T06, T10). They appreciated that students' code provided an opportunity to compare different solutions and also to debug authentically occurring student errors.

Key takeaway: Many instructors value exploration. Students should be able to write and run code during a lecture, either of their own accord or when the instructor asks them to do so.

3.2.3 Not All Information is in the Code. Most instructors did not dedicate the entirety of each lecture to live coding, but instead presented information in a variety of formats. Many instructors made use of lecture slides (T03, T05, T07, T08, T09), which were sometimes used as the main vehicle for conveying information (T07), but often used more sparingly, for example, only for a few lectures that benefited from visualization (T05). Many instructors also made use of screen annotation tools to draw directly on their slides or code (T04, T05, T06, T08, T10).

Four instructors primarily presented their lectures within an IDE (T01, T02, T05, T09). Instructors could highlight important points by adding comments (T01), though adding sufficient comments was seen as challenging due to time constraints (T02). These instructors provided their finished lecture code after class, but T01 and T05 had their own personal lecture notes, which they did not release to the students. T05 felt their personal notes would be difficult for others to understand: "*(T05) it's almost more like a mind map sort of style of notes [...] with just like, arrows going everywhere.*" Notably, T05's own lecture notes encoded the order in which they would build up the code during the lecture, but the code released to students only showed the final version. Thus, while one of the strengths of live coding is that it demonstrates the incremental process of writing code, that process may be challenging for students to capture in their notes if they simply copy the instructor's code.

While instructors generally could only speculate how their students took notes during their lectures, many hoped that students would take notes beyond just copying the code. For example, T06 said "*I hope they're writing down some markdown and stuff of things that we're talking about.*" T08, the high school teacher, encouraged more detailed notes by asking their students to adopt a specific note-taking strategy. They requested that their students keep notes in a Google Doc, interleaving textual explanations with screenshots of the instructor's code (which students accessed via a live video stream). T08 felt there was extra value in keeping notes separate from the code editor, forcing students to "*step away*" and think about how the code connects with other things they have learned.

Key takeaway: In live coding lectures, instructors present information in a variety of formats, including code, verbal instruction, code comments, lecture slides, markdown cells, drawings, and annotations. Students' notes taken during live coding lectures should capture more information than just the code itself.

4 Note-taking System Design

While our interviews provided insights into how to support students in live coding lectures, we still lacked an answer to the question: should students type along to copy the instructor's code? The instructors we interviewed had no consensus on an alternative approach, but the key takeaways from our interviews provided a guide for what might be effective. As we were not aware of an existing system that fit our needs, we decided to design our own based on the following three design goals:

- **(D1)** Students should be able to add the instructor's code to their notes without typing out a copy of the code.
- **(D2)** Students should be able to write, edit, and run code so that they can tinker and explore the code on their own.
- **(D3)** Students should be encouraged to take notes other than just copying code examples.

4.1 Experimental Note-taking System

Our note-taking system is a browser-based interface divided into two halves: a code editor on the left (Figure 1a) and a notes editor on the right (Figure 1d). In the code editor, the instructor's code is streamed in real time to a read-only tab. In the notes editor, the student can take notes using simple text-formatting options such as headers, bold text, and bulleted lists. If a student wants to capture code in their notes, they can simply highlight code from the code editor on the left-hand side and click "Add to Notes" (Figure 1f) to add a *code snapshot* (Figure 1e) to their notes **(D1)**. Code snapshots show up as an embedded widget in the notes editor containing a timestamp and the selected code with typical syntax highlighting, identical to what is found in the code editor. Because code snapshots are visually distinct from other notes, students can easily tell how much of their notes were authored by themselves versus the instructor, potentially encouraging students to take more of their own notes **(D3)**. For convenience, code snapshots in the notes editor support copy, paste, undo, and redo operations.

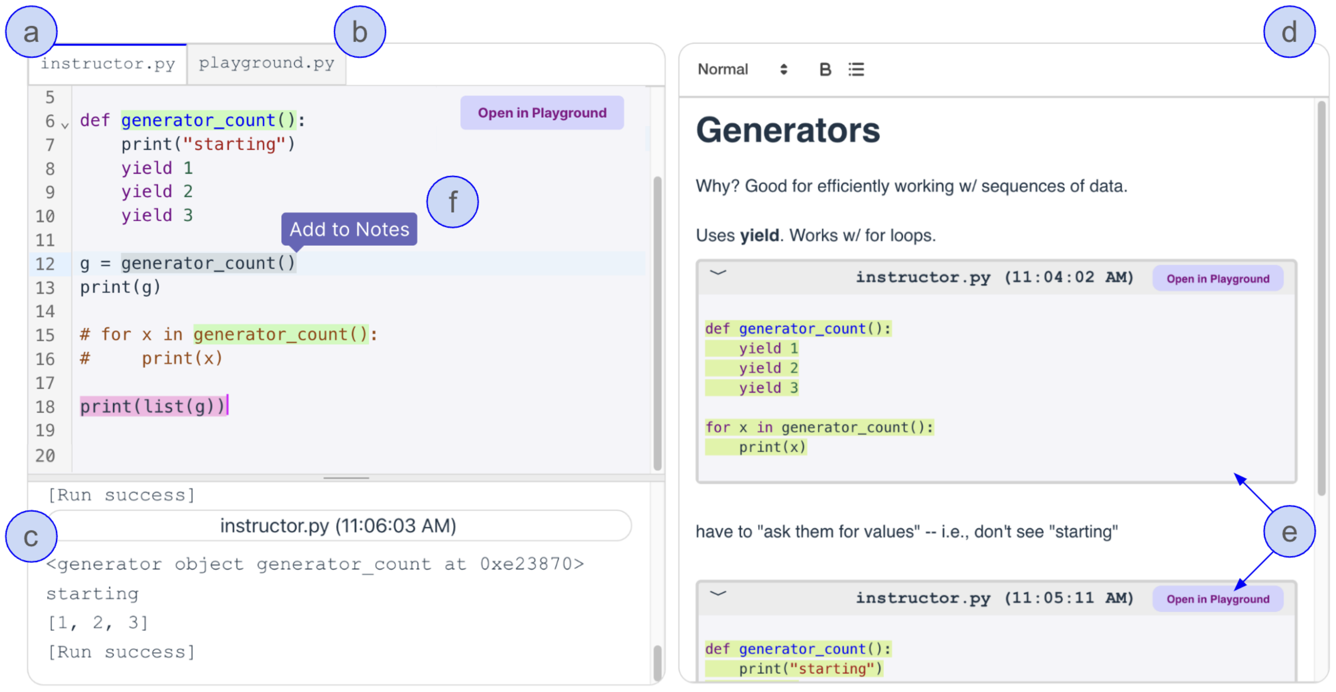
If students want to write their own code, they can click on the playground tab in the code editor (Figure 1b). Unlike the instructor tab, the playground tab is editable, providing a place for students to write and run Python code **(D2)**. If a student wants to explore writing their own code from scratch, they can type in code directly into the playground tab. On the other hand, if a student wants to tinker with the instructor's code (e.g., testing a small change), they can click the "Open in Playground" button in the instructor's code tab to immediately copy that code into the playground. Each code snapshot in the notes tab also has an "Open in Playground" button, which can be used even if that code no longer exists in the instructor tab. The code snapshot feature also works in the playground tab, so students can add snapshots of their own code to their notes.

4.2 Instructor Interface

We created a simple browser-based interface for instructors. The main purpose of the instructor interface is to synchronize with students using the experimental note-taking system (see 4.1). The interface consists of a code editor, run button, and console which displays the code output (similar to the left-hand side of Figure 1). All code changes and console output are broadcast to the experimental note-taking system and displayed on students' computers.

4.3 Baseline Code Editor

We developed a baseline interface which matched the look and feel of the instructor's code editor. The only difference is that a student can create separate tabs to write code. We reasoned that if students were typing along in a typical code editor, they would be free to



```

def generator_count():
    print("starting")
    yield 1
    yield 2
    yield 3

g = generator_count()
print(g)

# for x in generator_count():
#     print(x)

print(list(g))
```

```
[Run success]
instructor.py (11:06:03 AM)
<generator object generator_count at 0xe23870>
starting
[1, 2, 3]
[Run success]
```

Generators

Why? Good for efficiently working w/ sequences of data.

Uses **yield**. Works w/ for loops.

```
instructor.py (11:04:02 AM)

def generator_count():
    yield 1
    yield 2
    yield 3

for x in generator_count():
    print(x)
```

have to "ask them for values" -- i.e., don't see "starting"

```
instructor.py (11:05:11 AM)

def generator_count():
    print("starting")
```

Figure 1: A screenshot of our experimental note-taking system. (a) The instructor tab streams a live copy of the instructor's code to the student's browser. The tab also includes the instructor's cursor and current highlight, shown in pink. (b) The playground tab (not shown) is an editable code area where students can try their own code. They can either write a snippet from scratch, use normal copy and paste operations to copy the instructor's code, or use the "Open in Playground" button present in the instructor tab or on any code snapshot. (c) The console shows the output whenever the instructor runs code or whenever the student runs code from the playground. (d) The notes editor, where students have a simple text editor to write their notes. (e) Code snapshots are part of the notes editor and are created by highlighting code in the instructor or playground tabs and then clicking "add to notes" (f).

create new files (e.g., to organize their notes), so we also gave them that option here.

We highlight that the baseline interface has many similar affordances to the experimental system. Specifically, the baseline system allows students to take notes as code comments **(D3)**, and it allows students to change the instructor's code or write their own **(D2)**. The baseline differs from the experimental interface in two key respects: (1) students have to type out the instructor's code to capture it in their notes or experiment with their own changes, and (2) notes in code comments are not as easily distinguished from code as the code snapshots are in the experimental interface.

4.4 Implementation

Our system is a web application written in JavaScript which uses the CodeMirror 6 library to implement the code editors. We support running Python code in the client's browser using Pyodide in a background web worker thread. To prevent infinite loops, there is an eight-second time out for all code runs. On the backend, our system uses Node.js with Express, as well as Socket.IO, for broadcasting the instructor's changes to the experimental note-taking system. We use a relational database to store the instructor and student code and note-taking history. Code for the system can be found at https://github.com/echo-lab/live-coding-lecture.

5 Study 2: In-lab Experimental Study

We conducted a lab study to understand if our experimental note-taking system would have an advantage over the baseline code editor. We hypothesized the following:

- **H1:** Students find it easier to keep up with a live coding lecture using the experimental system.
- **H2:** Students take better notes using the experimental system.
- **H3:** Students have learning gains using the experimental system.

The study was approved by our university's Institutional Review Board and the study materials are available online.[4]

5.1 Participants

We recruited 57 participants using our university's mailing lists. The number of participants was based on a power analysis; using an

[4]Study materials: https://doi.org/10.5281/zenodo.15627773

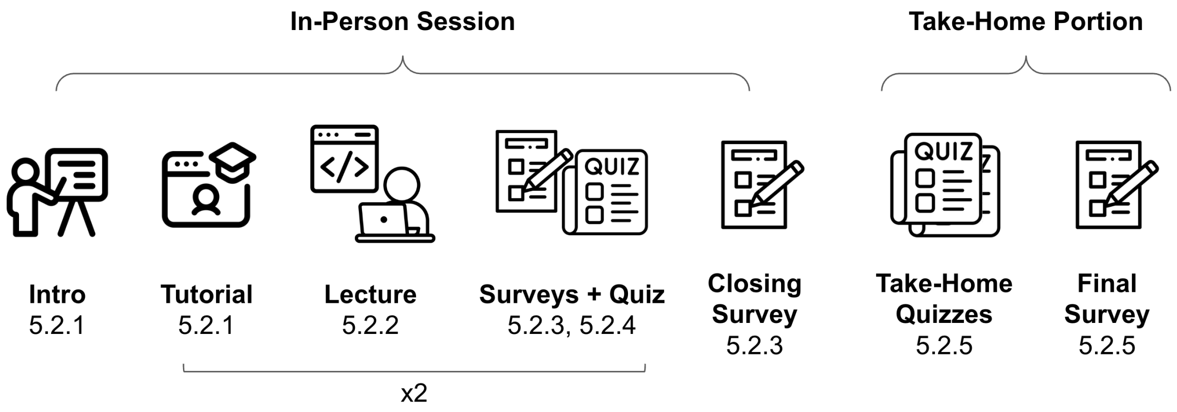


Figure 2: An overview of the study procedure with corresponding section numbers. Groups of 2-8 participants joined in-person sessions which lasted approximately 90 minutes. Each session had two different lectures, providing participants the opportunity to follow one lecture with the experimental note-taking system and the other with the baseline code editor. A week after the in-person session, participants completed two take-home quizzes using their lecture notes and filled out one final survey.

ANOVA for repeated measures within factors, the result indicated that a total sample size of 52 participants is required to detect a small effect size (f = 0.2) with a power of 0.80, $\alpha = 0.05$, two groups, two measurements. To be eligible, participants needed experience programming in Python at least equivalent to an introductory class. Interested participants filled out a short survey that asked them to rate their familiarity with a variety of topics in Python, including the topics of our two lectures: generators and decorators. We chose these topics as they are typically not covered in a Python class, but are simple enough to teach in a short lecture. We excluded those who answered "very familiar" for either lecture topic, but included those who answered "familiar," "somewhat familiar," or "I do not know." We reasoned that, in a typical CS classroom, students have a range of prior knowledge. We wanted to capture some of that variety while still being able to derive useful signals from the post-lecture quizzes.

5.2 Procedure

As summarized in Figure 2, our study consisted of a 90-minute in-person session followed a week later by take-home quizzes and a final survey. We hosted the in-person sessions with groups of 2-8 participants at a time. The sessions were led by the first author, while the second author facilitated. During these sessions, participants watched two 15-minute lectures, following one with our experimental note-taking system (Section 4.1) and the other using a baseline code editor (Section 4.3). At the start of each session, participants were randomly divided into two groups, determining the order they would use each interface. Thus, for each lecture, we had at least one student using the baseline code editor and one using the experimental system. We further outline the details of the study procedure below.

5.2.1 Introduction and Tutorial. We began the session with an outline of the study procedure. To incentivize active note-taking, we informed participants they would be quizzed on the material twice: once following the lecture (with no notes allowed) and once again the following week (with notes allowed).

We then gave a brief tutorial for our two interfaces, in which we projected our screen and asked participants to follow our actions. For the baseline interface, we instructed participants to type along with the lecturer as best they could and told them that they were free to write additional comments or try out code if they desired. For the experimental condition, we instructed participants to take any notes they wanted to record in the notes editor (Figure 1d) and to take snapshots of any code they wanted to save (Figure 1e). We explained how they could try out code in the playground tab and import code using the "open in playground" button. We repeated an abridged version of the tutorial before the second lecture.

5.2.2 Lectures. After the tutorial, we proceeded with the two lectures, which were designed and administered by the first author. One lecture was on generators in Python, and the other was on decorators. To limit ordering effects, we alternated the order of the two lectures in each session. To maintain consistency across lectures, the first author closely followed detailed notes, and we asked participants to refrain from asking questions about the lecture material. The lecture was designed based on the authors' experience and observations of live coding. The lecturer's code editor was projected in the room for participants to see, and the lecture proceeded with explanations interleaved with coding. At one point during each lecture, the instructor gave the students one minute to complete an exercise modeled after an example they had just explained, so in both conditions, students had the opportunity to write their own code.

5.2.3 In-person Surveys. After each lecture, we distributed a paper copy of the Raw NASA-TLX survey [26]. We chose the NASA-TLX survey because it focuses on a task (i.e., student note-taking), as opposed to an instrument like the CS Cognitive Load Component Survey [46], which focuses on instructional methods.

In addition to the NASA-TLX survey, we administered a digital survey about how participants perceived using the interface. This survey contained several seven-point Likert-scale questions asking participants how much they agreed with statements such as "I was able to keep up with the lecture without falling behind" and "I felt engaged with the lecture material while using this interface."

At the very end of the in-person session, we gave participants one final short survey, which prompted participants to give open-ended feedback comparing their experience using both interfaces.

5.2.4 In-person Quizzes. After the surveys following each lecture, we administered a closed-note quiz to participants to gauge learning. The quizzes consisted of one coding question followed by six multiple-choice and short-response questions. Participants had eight minutes in total to complete each quiz.

5.2.5 Take-home Quiz and Final Survey. Finally, one week after the in-person session, we sent an email to participants with links to their notes and to a second set of quizzes. Their notes were presented in the same interface they had used for each lecture, though the experimental system only showed the notes editor and not the instructor code tab (see Figure 1a). We emphasized that they were allowed to use these notes, but that they should not use any other resources or tools, such as search engines or large language models. Each take-home quiz had five multiple-choice and short-answer questions, followed by two coding questions. At the end of the quiz were two Likert-scale questions about how satisfied they were with their notes and if they used them or not. We suggested they take no more than 15 minutes on each quiz and reminded them there was no penalty for wrong answers. After the quizzes, participants filled out one last brief survey where they could provide open-ended feedback about their experience.

5.3 Qualitative Analysis of Notes Quality

We collected participants' notes and analyzed them to assess their quality, as the quality of notes has often been shown to correlate with better learning outcomes [33, 44, 51, 53]. One common measurement of quality involves breaking down a lecture into a set of *idea units*, which correspond to single phrases within a sentence [59], and then scoring each set of notes based on how many of those idea units are present [7, 38, 44]. As described below, we adapted this approach for live coding lectures, which consist of ideas spoken aloud accompanied by several code snippets which act as examples.

We began our analysis by summarizing the contents of each lecture into two different types of information: idea units and code units. We defined **idea units** as concepts or ideas that were spoken aloud during the live coding lecture. In line with previous work [38], idea units had to provide meaningful context. An example of an idea unit is "*once a generator is exhausted you cannot iterate through it again.*" A statement like "*here is an example of a generator*" would not count as an idea unit, because it does not describe what the example is.

We defined **code units** as conceptually related blocks of code written during the lecture. Typically, whenever the instructor ran the code in the editor, the relevant code would make up a single code unit. Code units can be thought of as examples—while they may illustrate an idea, we did not consider them to be idea units.

The first and second authors independently produced a set of idea units and code units for each lecture and then met to reconcile the differences. The final list of idea units and code units acted as a rubric: notes could be scored based on how many of the idea and code units from the lecture were present. We further categorized the code units present in each set of notes as follows:

- **Labeled or Unlabeled**: Labeled code units were those with any accompanying description which could later be used to contextualize the code. The description could be written in plain text or as code comments, and could be as simple as the phrase "generator example."
- **Correct or Erroneous**: Correct code units faithfully captured the meaning of the instructor's code, while erroneous code units had unintended errors (i.e., introduced by the student).

Thus, each set of lecture notes can be summarized by four metrics: (1) the number of idea units present; (2) the number of code units present; (3) the number of labeled code units present; and (4) the number of erroneous code units present. A summary is presented in Table 2.

The first two authors analyzed five sets of notes independently and then met to discuss and reconcile differences. After that, the first two authors repeated the process twice more, each time on a new set of twelve notes (10% of the data set). In the end, the authors achieved 96% agreement with a pooled Cohen's Kappa [14] value of 0.82, indicating a strong level of agreement [42]. Following this process, the first author analyzed the remaining set of notes.

6 Study 2: Results

Overall, when using our experimental note-taking system, participants reported a lower mental workload and generated more comprehensive notes. While participants perceived the experimental system as helpful for learning, those perceived benefits did not lead to an observable learning gain. We present our quantitative results broken down by each hypothesis, enriched with qualitative insights from our open-ended survey responses. We label responses from our participants as P01, P02, etc. For the remainder of this section, when we report that participants *agreed* with a statement, we present the aggregate number for the survey responses "*slightly agree*", "*agree*", and "*strongly agree*" (and analogously for disagree). For a more fine-grained breakdown of select survey responses, see Figure 3.

6.1 H1: Mental Workload and Keeping Up

The NASA Task Load Index (NASA-TLX) revealed significant differences, with the code editor consistently showing a higher workload across all measured dimensions (Figure 4). Wilcoxon signed-rank tests indicated significant differences ($p < 0.05$) for all subscales and for the raw total score ($W = 223.5, p < 0.0001, |r| = 0.626$)[5]. The most pronounced differences were observed in Temporal Demand, Effort, and Frustration (all $p < 0.0001$), with large effect sizes ($|r| = 0.695, 0.618, 0.680$, respectively). Significant differences were also observed for Mental Demand ($W = 297.0, p = 0.0001, |r| = 0.513$), Physical Demand ($W = 349.5, p = 0.0230, |r| = 0.303$), and

[5]For the NASA-TLX results, we excluded one participant because they left an item blank.

Table 2: The four metrics used to assess the quality of a set of notes.

Metric	Description
Idea Units	The total number of idea units from the lecture which were present in a set of notes.
Code Units	The total number of code units from the lecture which were present in a set of notes.
Labeled Code Units	Of the present code units, the number which were also contextualized with a label.
Erroneous Code Units	Of the present code units, the number which contained errors introduced by the student.

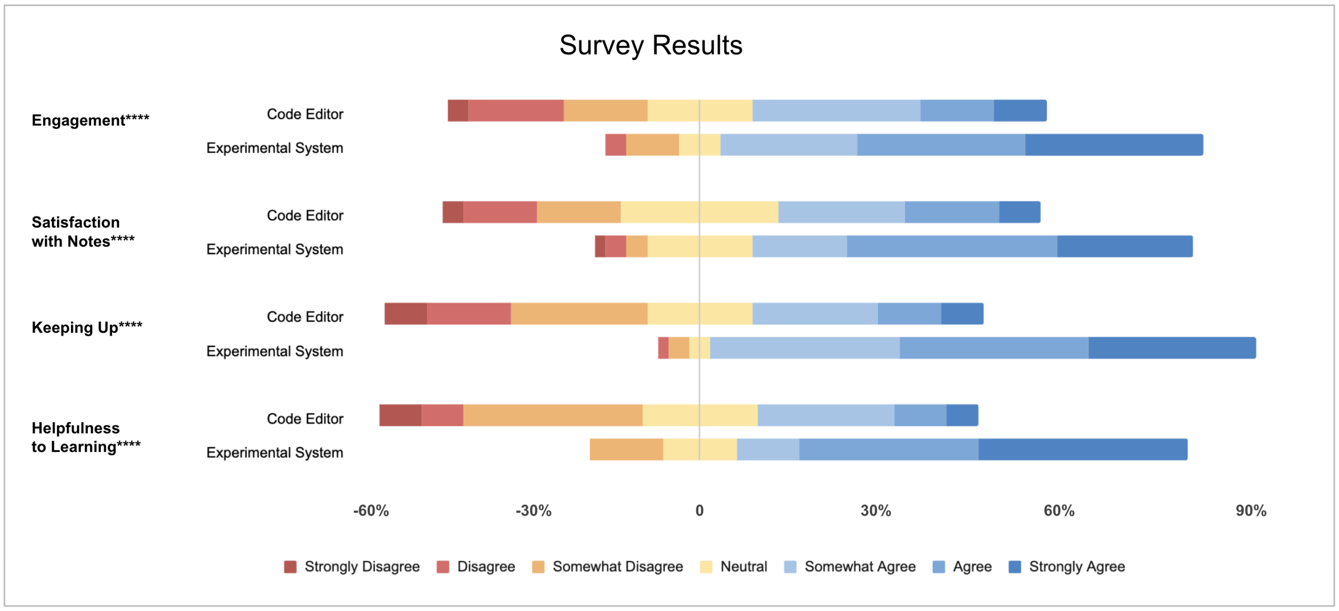


Figure 3: Comparison of participant responses on select questions from a 7-point Likert-scale survey broken down by experimental condition (code editor vs. experimental system). A vertical line at 0% represents the scale midpoint, and asterisks indicate significance levels from Wilcoxon signed-rank tests (* p < 0.05, ** p < 0.01, *** p < 0.001, **** p < 0.0001).

Performance ($W = 287.5, p = 0.0004, |r| = 0.470$). These results suggest that typing along in a code editor during lectures significantly increases students' mental workload.

From our survey, 91.2% of participants felt they kept up with the lecture when using our note-taking interface, while only 38.6% felt they kept up using a code editor ("Keeping up" in Figure 3). This difference was significant, according to a Wilcoxon signed-rank test ($W = 98.5, p < 0.0001, |r| = 0.766$). In their open-ended feedback, thirteen participants (23%) specifically mentioned that the experimental interface made it easier to follow the lecture, with one participant explaining "*(P28) the pace is maintained with the instructor*" and another expressing that "*(P31) [It] allowed me to pay more attention to the lecture instead of focusing on typing.*"

6.2 H2: Notes Quality

Our analysis revealed significant differences in our notes quality metrics (defined in 5.3) between the two experimental conditions. After confirming the non-normality of our four notes quality metrics with the Shapiro-Wilk test, we conducted a two-way mixed-effects ANOVA for aligned rank transformed [77] (ART) values of each metric with two factors: lecture topic (decorators vs. generators) and interface (code editor vs. experimental system). We summarize key results below.

In terms of idea units, our results showed a significant effect based on the interface ($F(1, 55) = 69.10, p < 0.0001$). As shown in Figure 5, more idea units were present in notes generated using the experimental system ($\mu = 12.6, \sigma = 6.11$) compared to the code editor ($\mu = 6.9, \sigma = 4.45$). Thus, when using the experimental system, students captured more of the conceptual ideas spoken aloud in the lecture.

In terms of code units, our results showed no significant effect from the interface on the total number of code units present in student notes ($F(1, 55) = 0.43747, p = 0.51$). On the other hand, there was a significant effect of the interface on the number of *labeled* code units ($F(1, 55) = 54.01785, p < 0.0001$): code units were more often labeled in notes generated using the experimental system ($\mu = 9.2, \sigma = 4.13$) compared to the code editor ($\mu = 5.8, \sigma = 3.45$). There was also a significant effect of the interface on the number of erroneous code units ($F(1, 55) = 54.102, p < 0.0001$), with errors being less common in the experimental system ($\mu = 0.19, \sigma = 0.40$) compared to the code editor ($\mu = 1.37, 1.60$). Altogether, although the notes in both conditions contained about the same amount of

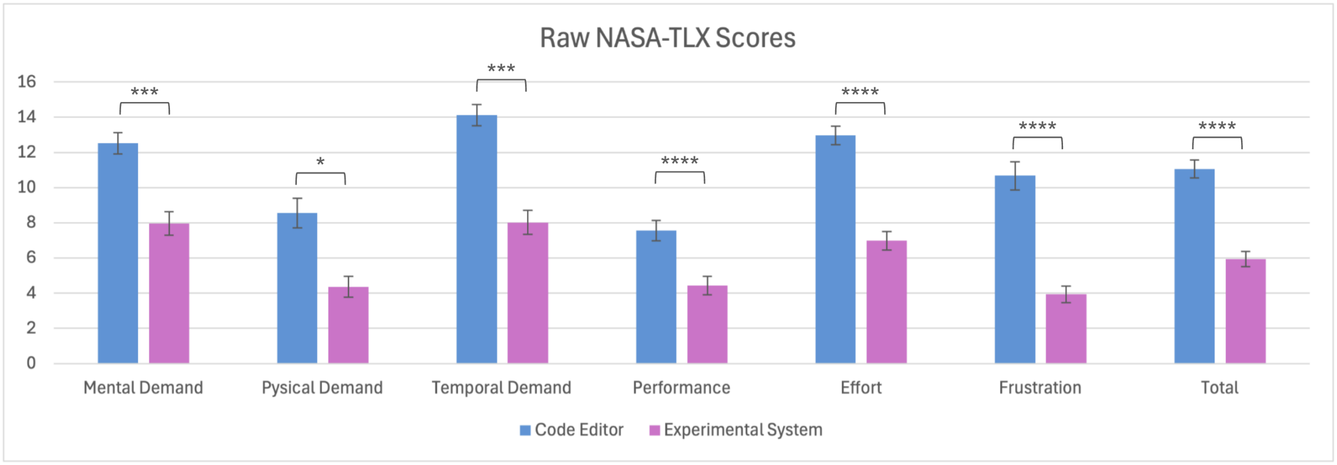


Figure 4: A comparison of Raw NASA-TLX Scores across the two experimental conditions. The total score is normalized to a scale from 0 to 20, matching the scales for each dimension. Error bars indicate the standard error for each measurement. Asterisks indicate significance levels from Wilcoxon signed-rank tests (* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$, **** $p < 0.0001$).

code, in the experimental condition, a higher portion of that code contained labels to provide context, and a lower portion of the code contained unintended errors. We note that errors and a lack of labels may make reviewing notes more difficult, and code errors may also occupy a student's attention if they plan to debug them during a lecture.

Participants' perceptions of their notes aligned with our analysis. When we asked participants to rate the quality of their notes after the open-note take-home quiz, 73.7% were satisfied with the quality of their notes in the experimental condition, while only 43.9% felt the same when using the standard code editor ("Satisfaction with Notes" in Figure 3). A Wilcoxon signed-rank test revealed this was a significant difference ($W = 211.5, p < 0.001, |r| = 0.647$). In the open-ended feedback, many participants expressed concern that their notes from the code editor condition were either missing content or contained incorrectly copied code.

Feedback from participants highlighted the ease of copying the instructor's code in the experimental system as a key benefit, but many participants also appreciated the extra formatting capabilities of the rich text editor.

> *(P24) [The experimental system] was much easier to structure information; I don't have to comment out notes, and I can have headers and bold and bullet points. I can also just copy code snippets from the instructor rather than having to make sure I type it all out exactly w/o typos.*

> *(P19) [The experimental system] was preferable because it was much easier to distinguish which parts of the notes were code snippets and which ones were not.*

6.3 H3: Learning Outcomes and Experience

Figure 6 shows both in-person and take-home quiz scores for both lecture topics: decorators and generators. Similar to our notes quality analysis, we first confirmed non-normality with the Shapiro-Wilk test, and then conducted a two-way mixed-effects ANOVA for aligned rank transformed [77] (ART) values of quiz scores—both in-person and take-home quiz scores—with two factors: the lecture topic (decorators vs. generators) and interface (code editor vs. experimental system).

Overall, there were no significant effects of either factor on in-person quiz scores. For the take-home quiz scores, we saw a significant effect from the lecture topic ($F(1, 55) = 9.83, p = 0.003$), but no significant effect from the interface or the combination of interface and lecture topic. Thus, we found no evidence to suggest that the interface participants used had a direct effect on their learning outcomes.

In contrast, results from our survey indicate that participants *perceived* the experimental system to be more beneficial for learning. 75.4% of participants felt that the experimental system positively contributed to learning ("Helpfulness to Learning" in Figure 3), while only 36.8% felt the same way about the code editor ($W = 179.5, p < 0.001, |r| = 0.681$). As one student explained, "*(P21) I had the opportunity to listen more as I do not have to type the entire code.*" Notably, the experimental system was also perceived as more engaging, with 80.7% of students reporting feeling engaged ("Engagement" in Figure 3) compared to just 49.1% for the code editor ($W = 229.5, p < 0.001, |r| = 0.628$).

Finally, we found multiple correlations between quiz scores and the notes quality metrics (defined in Section 5.3). To measure correlations, we used Kendall's Tau, a non-parametric test that can handle ties [27]. In-person quiz scores were significantly correlated with idea units present in the notes ($\tau = 0.206, p = 0.002$) and labeled code units present in the notes ($\tau = 0.177, p = 0.008$). Take-home quiz scores were similarly significantly correlated with idea units ($\tau = 0.199, p = 0.002$) and labeled code units ($\tau = 0.145, p = 0.029$).

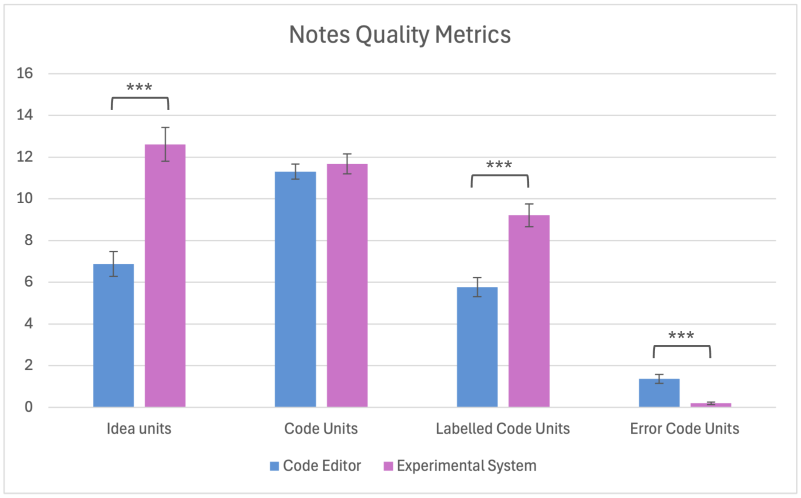


Figure 5: A comparison of the notes quality metrics across the two experimental conditions. Bars indicate average values and error bars indicate the standard errors. Asterisks indicate significance levels of the experimental condition factor based on running a mixed-effects ANOVA: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$, **** $p < 0.0001$. While this graph shows data for both lectures together, grouping the data by lecture yields similar trends.

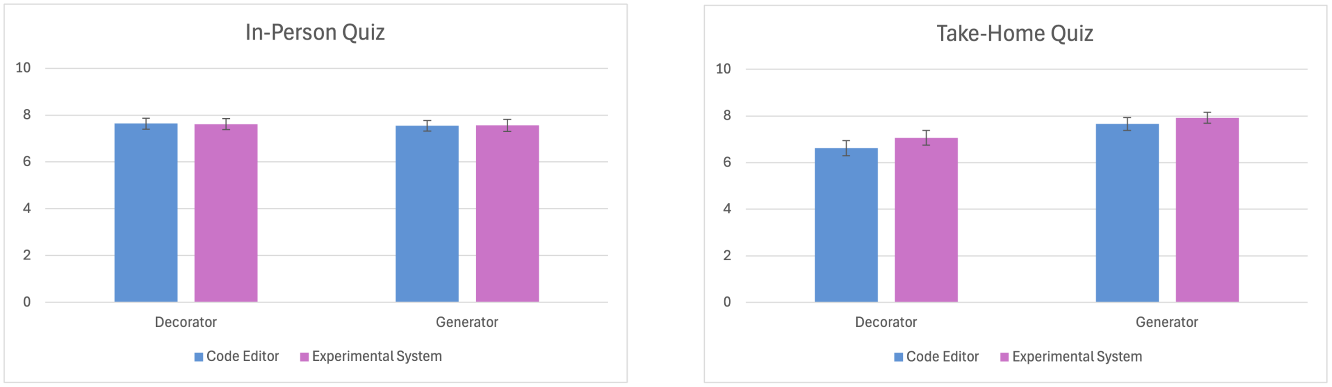


Figure 6: Weighted quiz scores for in-person and take-home quizzes, normalized out of a maximum of 10 points. The in-person quiz consisted of six multiple-choice/short-response questions and one coding question. The take-home quiz, administered one week after the lab study, included five multiple-choice/short-response questions and two coding questions. Coding questions were weighted twice as much as all other types of questions. For both the in-person and take-home quizzes, there were no significant effects from the interface, nor from the interaction of the interface and topic.

We note that while these correlations are significant, they may be considered weak [62]. The full set of correlations are found in Table 3.

7 Discussion

Our findings contribute to understanding the impact of different note-taking strategies in live coding lectures, highlighting the benefits of alternative approaches compared to transcribing the instructor's code. In summary, we saw that our experimental note-taking system—which did not require transcribing the instructor's code—reduced mental workload and led to higher-quality notes. However, we did not find any evidence that it would result in learning gains. In this section, we further interpret these results and discuss implications for live coding lectures and note-taking practices. We also highlight limitations that may impact the generalizability of our results and call for further investigation.

7.1 Implications for Live Coding Lectures

In our lab study, we found our experimental note-taking system improved upon two known drawbacks of live coding: that it feels too fast for students and that it is difficult to take notes [66]. Even with inconclusive learning gains, this suggests that adopting a system

Table 3: Correlations (Kendall's τ) between Note Quality Metrics and Quiz Scores (* $p < 0.05$, ** $p < 0.01$)

	Idea Units	Code Units	Labeled Code Units	Erroneous Code Units
In-Person Quiz	**0.206** **	0.045	**0.177** **	-0.071
Take-Home Quiz	**0.199** **	0.126	**0.145** *	-0.074

like ours may improve students' experiences in live coding lectures. Instructors can approximate our experimental system using multiple existing technologies: for example, using a code editor which supports read-only sharing (like Replit[6]) while asking students to use a rich text editor with code block support (like Notion[7]). We note that this fragmented approach may be less ideal than having a single, integrative tool [43].

Regarding our approach, instructors may be concerned that students will lose the benefits of typing along. From our interviews, those perceived benefits include that typing along may (1) keep students engaged; (2) aid memory; and (3) offer learning opportunities for students to make and correct mistakes. We suggest that using in-class coding exercises is an effective alternative to realize each of these benefits. For keeping students engaged, including in-class exercises in live coding lectures has been shown to have sustained positive effects on engagement [65]. For aiding memory, other studies suggest that typing out something verbatim does not improve memory [44, 48], and we similarly did not see a difference in our study. For learning from mistakes, we argue that making mistakes while typing along during a lecture can involve a risk of students becoming distracted or falling behind, which makes this learning better suited for a time set aside for an activity. We also note that our system may leave more time for class activities: students may be more likely to keep up with a faster-paced lecture and instructors can quickly share starter code templates for class exercises.

Finally, while our lab study did not find learning gains using our experimental system, future work can see if this result holds in a classroom setting. As described below, our results suggest potential avenues for learning gains that might be more likely to show up in the classroom.

First, we found that using the experimental system led to a significantly decreased mental workload for students. Using the frame of cognitive load theory, decreasing extraneous cognitive load (e.g., from copying the instructor's code) should positively impact learning outcomes [70, 72]. While we did not see this result from our study, one explanation is that the extraneous cognitive load from copying the instructor's code was not enough to overwhelm the working memory of our participants, given the relatively short time span of the lecture (15 minutes), and thus had no negative effects on learning. If this were the case, then we might expect typing along in a code editor to be less effective for longer live coding sessions, as working memory can deplete over periods of extended exertion [10].

Second, our study found that students produced more comprehensive notes using our experimental system. While reviewing higher quality notes is likely to lead to better learning outcomes [31, 54], students had little incentive to review their notes in our study. In a classroom setting, however, students may naturally be incentivized to study their notes (e.g., to prepare for exams), and thus are more likely to reap the learning benefits of reviewing notes [31, 35].

7.2 Implications for Note-taking

Our results suggest that using a rich text editor with copy and paste access to the instructor's code helps students to produce more comprehensive notes during live coding lectures. Better notes are generally associated with better learning and academic performance [32, 51, 52], which matches our finding that the number of idea units and labeled code units were both correlated with higher quiz scores (see 6.3). Given the above correlation and our finding that the experimental system led to notes with more idea units and labeled code units (see 6.2), one might have also expected to see learning gains in the experimental condition. The relationship between notes quality and learning may be mediated by a variety of factors, such as prior knowledge and transcription speed [54]—future work can examine which factors may play an important role in live coding lectures.

Our experimental system provides two features which may have led to higher quality notes. First, the primary input method is rich text, while code is supported as an embedded element which is visually distinct. Consequently, it is easy to visualize which part of the notes are generated from the student and which are generated from the instructor (i.e., the code). If a student sees they have not generated their own notes, they may feel like they are not using the interface as intended. Second, our experimental system allows copy-and-pasting the instructor's code without typing it out. If a student does not have to transcribe the code, this may free up more time to take more comprehensive notes. Future work can isolate these two features to test the effects of each separately.

Finally, we believe this type of note-taking, which allows students to capture and embed materials created by the instructor, could prove useful in a variety of lectures, including CS lectures using static code or even lectures on other topics outside of CS. We imagine a system where students can take flexible notes using rich text, selectively embedding relevant artifacts shared by the instructor, such as diagrams, pictures, tables, or even demonstrations (e.g., via video capture), which might not be feasible to copy into their notes otherwise. While some students already use pictures to capture lecture materials, pictures alone without additional notes may not be effective [78]. When given materials like PowerPoint slides, students often annotate those slides directly [45]. Future work can examine how an annotation approach compares to an embedding approach like in our experimental system.

7.3 Limitations

One limitation of our work is that we tested our interface in a lab study rather than a classroom. We chose a lab study because it

[6]https://docs.replit.com/additional-resources/sharing-your-repl
[7]https://www.notion.com/help/code-blocks

offered us more control in guiding student note-taking, and furthermore, we did not want to subject students to an untested intervention that might affect their class performance. Because the encoding benefits of note-taking are sensitive to a variety of factors [34], it is possible we may see different results in a classroom setting (see also 7.1). In a classroom setting, lectures might be longer, the topics would vary, and students might be more motivated to learn the material. On the other hand, students may be more likely to be off-task in a classroom setting, especially when using laptops [56]. Having the instructor's code available to copy might benefit students who multitask in lectures and need a way to catch up [41].

Another limitation of our study is that we primarily tested the encoding function of notes, rather than the storage function. While participants did take follow-up quizzes a week after the lectures, it is difficult to interpret those results directly because (1) performance could have been influenced by taking the same-day quiz (i.e., a testing effect [40]), and (2) participants were allowed to use their notes during the quiz. While our analysis of the notes quality suggests an improvement from using our system, future work might test the storage function more directly by having students review their notes for a set period of time before taking a follow-up quiz [31].

8 Conclusion

In this work, we set out to better understand how to support students during live coding lectures. Through interviews with instructors, we came to understand key challenges faced by students as well as the pedagogical values held by instructors. Through a lab study (n=57), we compared how students followed a lecture either by typing along in a code editor or using a rich text editor that allowed taking snapshots of the instructor's code. We found the latter approach led to a significantly lower mental workload and better lent itself to taking quality notes, though ultimately did not lead to learning gains. Future work can investigate how our findings translate to a semester-long classroom setting.

References

[1] [n. d.]. HarvardX: CS50's Introduction to Computer Science. Retrieved Dec 5, 2024 from https://www.edx.org/learn/computer-science/harvard-university-cs50-s-introduction-to-computer-science

[2] John Aycock. 2018. Stick to the script: lightweight recording and playback of live coding. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. Association for Computing Machinery, New York, NY, USA, 350–351. doi:10.1145/3197091.3205830

[3] Jens Bennedsen and Michael E. Caspersen. 2005. Revealing the programming process. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*. ACM, St. Louis Missouri USA, 186–190. doi:10.1145/1047344.1047413

[4] Caroline Palma Berger. 2023. *"I Feel Like I'm Teaching in a Gladiator Ring": Barriers and Benefits of Live Coding*. Master's thesis. University of Maryland, College Park, United States – Maryland. https://www.proquest.com/docview/2832215582/abstract/CE57A3ECC9A84495PQ/1 ISBN: 9798379758547.

[5] Burke H. Bretzing and Raymond W. Kulhavy. 1979. Notetaking and depth of processing. *Contemporary Educational Psychology* 4, 2 (1979), 145–153. doi:10.1016/0361-476X(79)90069-9

[6] Neil C. C. Brown and Greg Wilson. 2018. Ten quick tips for teaching programming. *PLOS Computational Biology* 14, 4 (04 2018), 1–8. doi:10.1371/journal.pcbi.1006023

[7] Dung C. Bui, Joel Myerson, and Sandra Hale. 2013. Note-taking with computers: Exploring alternative strategies for improved recall. *Journal of Educational Psychology* 105, 2 (May 2013), 299–309. doi:10.1037/a0030367 Publisher: American Psychological Association.

[8] Ankur Chattopadhyay, Drew Ryan, and James Pockrandt. 2022. Scaffolded Live Coding: A Hybrid Pedagogical Approach for Enhanced Teaching of Coding Skills. In *2022 IEEE Frontiers in Education Conference (FIE)*. 1–9. doi:10.1109/FIE56618.2022.9962513 ISSN: 2377-634X.

[9] Charles H. Chen and Philip J. Guo. 2019. Improv: Teaching Programming at Scale via Live Coding. In *Proceedings of the Sixth (2019) ACM Conference on Learning @ Scale (L@S '19)*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/3330430.3333627

[10] Ouhao Chen, Juan C. Castro-Alonso, Fred Paas, and John Sweller. 2018. Extending Cognitive Load Theory to Incorporate Working Memory Resource Depletion: Evidence from the Spacing Effect. *Educational Psychology Review* 30, 2 (June 2018), 483–501. doi:10.1007/s10648-017-9426-2

[11] Yan Chen, Walter S Lasecki, and Tao Dong. 2021. Towards supporting programming education at scale via live streaming. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW3 (2021), 1–19.

[12] Victoria Clarke, Virginia Braun, and Nikki Hayfield. 2015. Thematic analysis. *Qualitative psychology: A practical guide to research methods* 3 (2015), 222–248.

[13] Juliet Corbin and Anselm Strauss. 2014. *Basics of qualitative research: Techniques and procedures for developing grounded theory.* Sage publications.

[14] Han De Vries, Marc N. Elliott, David E. Kanouse, and Stephanie S. Teleki. 2008. Using Pooled Kappa to Summarize Interrater Agreement across Many Items. *Field Methods* 20, 3 (Aug. 2008), 272–282. doi:10.1177/1525822X08317166

[15] Francis J. Di Vesta and G. Susan Gray. 1972. Listening and note taking. *Journal of Educational Psychology* 63, 1 (Feb. 1972), 8–14. doi:10.1037/h0032243 Publisher: American Psychological Association.

[16] Francis J. Di Vesta and G. Susan Gray. 1973. Listening and note taking: II. Immediate and delayed recall as functions of variations in thematic continuity, note taking, and length of listening-review intervals. *Journal of Educational Psychology* 64, 3 (June 1973), 278–287. doi:10.1037/h0034589 Publisher: American Psychological Association.

[17] Hanxiang Du, Dion Udokop, and Bo Pei. 2025. Live Coding Prompts Engagement, But Not Necessarily Grades. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 283–289. doi:10.1145/3641554.3701794

[18] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D. Miller. 2018. Watch Me Code: Programming Mentorship Communities on Twitch.tv. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 50 (Nov. 2018), 18 pages. doi:10.1145/3274319

[19] Alessio Gaspar and Sarah Langevin. 2007. Active learning in introductory programming courses through student-led "live coding" and test-driven pair programming. In *International Conference on Education and Information Systems, Technologies and Applications, Orlando, FL.*

[20] Alessio Gaspar and Sarah Langevin. 2007. Restoring "coding with intention" in introductory programming courses. In *Proceedings of the 8th ACM SIGITE conference on Information technology education (SIGITE '07)*. Association for Computing Machinery, New York, NY, USA, 91–98. doi:10.1145/1324302.1324323

[21] Adam M. Gaweda, Collin F. Lynch, Nathan Seamon, Gabriel Silva De Oliveira, and Alay Deliwa. 2020. Typing Exercises as Interactive Worked Examples for Deliberate Practice in CS Courses. In *Proceedings of the Twenty-Second Australasian Computing Education Conference.* ACM, Melbourne VIC Australia, 105–113. doi:10.1145/3373165.3373177

[22] Nasser Giacaman. 2012. Teaching by Example: Using Analogies and Live Coding Demonstrations to Teach Parallel Computing Concepts to Undergraduate Students. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum.* 1295–1298. doi:10.1109/IPDPSW.2012.158

[23] Rolando Gonzalez and Per Lauvaas. 2017. An experience report using Scrimba: An interactive and cooperative web development tool in a blended learning setting. *Norsk IKT-konferanse for forskning og utdanning* (Nov. 2017). https://www.ntnu.no/ojs/index.php/nikt/article/view/5296

[24] Rolando Gonzalez and Per Lauvås. 2017. *Teaching Introductory Web Development Using Scrimba; An Interactive And Cooperative Development Tool.*

[25] Tor-Morten Grønli and Siri Fagernes. 2020. The live programming lecturing technique: A study of the student experience in introductory and advanced programming courses. In *Norsk IKT-konferanse for forskning og utdanning.*

[26] Sandra G. Hart. 2006. Nasa-Task Load Index (NASA-TLX); 20 Years Later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 50, 9 (2006), 904–908. doi:10.1177/154193120605000909

[27] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. 2015. *Nonparametric Statistical Methods* (1 ed.). Wiley. doi:10.1002/9781119196037

[28] Amber Horvath, Brad Myers, Andrew Macvean, and Imtiaz Rahman. 2022. Using Annotations for Sensemaking About Code. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3526113.3545667

[29] Renee Jansen, Daniel Lakens, and Wijnand IJsselsteijn. 2017. An integrative review of the cognitive costs and benefits of note - taking. doi:10.31234/osf.io/ty4nq

[30] Michael J. Kane, Bridget A. Smeekens, Claudia C. Von Bastian, John H. Lurquin, Nicholas P. Carruth, and Akira Miyake. 2017. A combined experimental and individual-differences investigation into mind wandering during a video lecture. *Journal of Experimental Psychology: General* 146, 11 (Nov. 2017), 1649–1674. doi:10.

1037/xge0000362

[31] Kenneth A. Kiewra. 1989. A review of note-taking: The encoding-storage paradigm and beyond. *Educational Psychology Review* 1, 2 (June 1989), 147–172. doi:10.1007/BF01326640

[32] Kenneth A. Kiewra, Stephen L. Benton, and Lance B. Lewis. 1987. Qualitative Aspects of Notetaking and Their Relationship with Information-Processing Ability and Academic Achievement. *Journal of Instructional Psychology* 14, 3 (Sept. 1987), 110–117. https://www.proquest.com/docview/1416364654/citation/99DE3DD0FD0D4CB4PQ/1 Num Pages: 8 Place: Milwaukee, Wis., United States Publisher: V-U Pub. Co..

[33] Kenneth A Kiewra and Harold J Fletcher. 1984. The relationship between levels of note-taking and achievement. *Human Learning: Journal of Practical Research & Applications* (1984).

[34] Keiichi Kobayashi. 2005. What limits the encoding effect of note-taking? A meta-analytic examination. *Contemporary Educational Psychology* 30, 2 (April 2005), 242–262. doi:10.1016/j.cedpsych.2004.10.001

[35] Keiichi Kobayashi. 2006. Combined Effects of Note-Taking/-Reviewing on Learning and the Enhancement through Interventions: A meta-analytic review. *Educational Psychology* 26, 3 (June 2006), 459–477. doi:10.1080/01443410500342070

[36] Yu-Tzu Lin, Martin K.-C. Yeh, and Sheng-Rong Tan. 2022. Teaching Programming by Revealing Thinking Process: Watching Experts' Live Coding Videos With Reflection Annotations. *IEEE Transactions on Education* 65, 4 (Nov. 2022), 617–627. doi:10.1109/TE.2022.3155884 Conference Name: IEEE Transactions on Education.

[37] Sophie I. Lindquist and John P. McLean. 2011. Daydreaming and its correlates in an educational environment. *Learning and Individual Differences* 21, 2 (April 2011), 158–167. doi:10.1016/j.lindif.2010.12.006

[38] Linlin Luo, Kenneth A. Kiewra, Abraham E. Flanigan, and Markeya S. Peteranetz. 2018. Laptop versus longhand note taking: effects on lecture notes and achievement. *Instructional Science* 46, 6 (Dec. 2018), 947–971. doi:10.1007/s11251-018-9458-0

[39] Mark Mahoney. 2023. Storyteller: Guiding Students Through Code Examples. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1131–1135. doi:10.1145/3545945.3569843

[40] Mary H. Derbish Mark A. McDaniel, Janis L. Anderson and Nova Morrisette. 2007. Testing the testing effect in the classroom. *European Journal of Cognitive Psychology* 19, 4-5 (2007), 494–513. doi:10.1080/09541440701326154

[41] Sahar Mavali, Dongwook Yoon, Luanne Sinnamon, and Sidney S Fels. 2024. Time-Turner: A Bichronous Learning Environment to Support Positive In-class Multitasking of Online Learners. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–15. doi:10.1145/3613904.3641985

[42] Mary L McHugh. 2012. Interrater reliability: the kappa statistic. *Biochemia medica* 22, 3 (2012), 276–282.

[43] Samim Mirhosseini, Austin Z. Henley, and Chris Parnin. 2023. What Is Your Biggest Pain Point? An Investigation of CS Instructor Obstacles, Workarounds, and Desires. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 291–297. doi:10.1145/3545945.3569816

[44] Kayla Morehead, John Dunlosky, and Katherine A. Rawson. 2019. How Much Mightier Is the Pen than the Keyboard for Note-Taking? A Replication and Extension of Mueller and Oppenheimer (2014). *Educational Psychology Review* 31, 3 (Sept. 2019), 753–780. doi:10.1007/s10648-019-09468-2

[45] Kayla Morehead, John Dunlosky, Katherine A. Rawson, Rachael Blasiman, and R. Benjamin Hollis. 2019. Note-taking habits of 21st Century college students: implications for student learning, memory, and achievement. *Memory* 27, 6 (July 2019), 807–819. doi:10.1080/09658211.2019.1569694 Publisher: Taylor & Francis Ltd.

[46] Briana B. Morrison, Brian Dorn, and Mark Guzdial. 2014. Measuring cognitive load in introductory CS: adaptation of an instrument. In *Proceedings of the tenth annual conference on International computing education research (ICER '14)*. Association for Computing Machinery, New York, NY, USA, 131–138. doi:10.1145/2632320.2632348

[47] Mogeeb A. A. Mosleh, Mohd Sapiyan Baba, Sorayya Malek, and Musaed A. Alhussein. 2016. Challenges of Digital Note Taking. In *Advanced Computer and Communication Engineering Technology*, Hamzah Asyrani Sulaiman, Mohd Azlishah Othman, Mohd Fairuz Iskandar Othman, Yahaya Abd Rahim, and Naim Che Pee (Eds.). Springer International Publishing, Cham, 211–231. doi:10.1007/978-3-319-24584-3_19

[48] Pam A. Mueller and Daniel M. Oppenheimer. 2014. The Pen Is Mightier Than the Keyboard: Advantages of Longhand Over Laptop Note Taking. *Psychological Science* 25, 6 (June 2014), 1159–1168. doi:10.1177/0956797614524581 Publisher: SAGE Publications Inc.

[49] Jungkook Park, Yeong Hoon Park, Jinhan Kim, Jeongmin Cha, Suin Kim, and Alice Oh. 2018. Elicast: embedding interactive exercises in instructional programming screencasts. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale (L@S '18)*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/3231644.3231657

[50] John Paxton. 2002. Live programming as a lecture technique. *Journal of Computing Sciences in Colleges* 18, 2 (Dec. 2002), 51–56.

[51] Stephen T. Peverly, Joanna K. Garner, and Pooja C. Vekaria. 2014. Both handwriting speed and selective attention are important to lecture note-taking. *Reading and Writing* 27, 1 (Jan. 2014), 1–30. doi:10.1007/s11145-013-9431-x

[52] Stephen T. Peverly and James F. Sumowski. 2012. What Variables Predict Quality of Text Notes and are Text Notes Related to Performance on Different Types of Tests? *Applied Cognitive Psychology* 26, 1 (2012), 104–117. doi:10.1002/acp.1802

[53] Stephen T. Peverly, Pooja C. Vekaria, Lindsay A. Reddington, James F. Sumowski, Kamauru R. Johnson, and Crystal M. Ramsay. 2013. The Relationship of Handwriting Speed, Working Memory, Language Comprehension and Outlines to Lecture Note-taking and Test-taking among College Students. *Applied Cognitive Psychology* 27, 1 (2013), 115–126. doi:10.1002/acp.2881

[54] Stephen T. Peverly and Amie D. Wolf. 2019. Note-Taking. In *The Cambridge Handbook of Cognition and Education*, John Dunlosky and Katherine A. Rawson (Eds.). Cambridge University Press, Cambridge, 320–355. doi:10.1017/9781108235631.014

[55] Annie Piolat, Thierry Olive, and Ronald T. Kellogg. 2005. Cognitive effort during note taking. *Applied Cognitive Psychology* 19, 3 (April 2005), 291–312. doi:10.1002/acp.1086 Publisher: Wiley-Blackwell.

[56] Eric D. Ragan, Samuel R. Jennings, John D. Massey, and Peter E. Doolittle. 2014. Unregulated use of laptops over time in large lecture classes. *Computers & Education* 78 (Sept. 2014), 78–86. doi:10.1016/j.compedu.2014.05.002

[57] 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. doi:10.1145/3373165.3373182

[58] 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, 1–8. doi:10.1145/3279720.3279725

[59] Katherine A. Rawson and Walter Kintsch. 2005. Rereading Effects Depend on Time of Test. *Journal of Educational Psychology* 97, 1 (Feb. 2005), 70–80. doi:10.1037/0022-0663.97.1.70 Publisher: American Psychological Association.

[60] Patrick Rein, Stefan Ramson, Jens Lincke, Robert Hirschfeld, and Tobias Pape. 2018. Exploratory and Live, Programming and Coding: A Literature Study Comparing Perspectives on Liveness. *The Art, Science, and Engineering of Programming* 3, 1 (July 2018), 1. doi:10.22152/programming-journal.org/2019/3/1

[61] 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. doi:10.1145/2445196.2445388

[62] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia* 126, 5 (2018), 1763–1768.

[63] 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. doi:10.1145/3430665.3456382

[64] 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 2023)*. Association for Computing Machinery, New York, NY, USA, 533–539. doi:10.1145/3587102.3588846

[65] 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* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 513–519. doi:10.1145/3649217.3653537

[66] 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, Vol. 1)*. Association for Computing Machinery, New York, NY, USA, 476–494. doi:10.1145/3568813.3600122

[67] Amy Shannon and Valerie Summet. 2015. Live coding in introductory computer science courses. *Journal of Computing Sciences in Colleges* 31, 2 (Dec. 2015), 158–164.

[68] Adalbert Gerald Soosai Raj, Jignesh Patel, and Richard Halverson. 2018. Is More Active Always Better for Teaching Introductory Programming?. In *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. 103–109. doi:10.1109/LaTICE.2018.00006 ISSN: 2475-1057.

[69] Ben Stephenson. 2019. Coding Demonstration Videos for CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 105–111. doi:10.1145/3287324.3287445

[70] John Sweller, Paul Ayres, and Slava Kalyuga. 2011. Intrinsic and Extraneous Cognitive Load. In *Cognitive Load Theory*, John Sweller, Paul Ayres, and Slava Kalyuga (Eds.). Springer, New York, NY, 57–69. doi:10.1007/978-1-4419-8126-4_5

[71] Sheng-Rong Tan, Yu-Tzu Lin, and Jia-Sin Liou. 2016. Teaching by demonstration: programming instruction by using live-coding videos. In *EdMedia+ Innovate Learning*. Association for the Advancement of Computing in Education (AACE), 1294–1298.

[72] Jeroen J. G. van Merriënboer, Liesbeth Kester, and Fred Paas. 2006. Teaching complex rather than simple tasks: balancing intrinsic and germane load to enhance transfer of learning. *Applied Cognitive Psychology* 20, 3 (2006), 343–352. doi:10.1002/acp.1250 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/acp.1250.

[73] 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. doi:10.1145/3649217.3653562

[74] 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* (Pittsburgh, PA, USA) *(SIGCSETS 2025)*. Association for Computing Machinery, New York, NY, USA, 1190–1196. doi:10.1145/3641554.3701786

[75] Greg Wilson. 2016. Software Carpentry: lessons learned. *F1000Research* 3 (2016), 62. https://doi.org/10.12688/f1000research.3-62.v2

[76] G. Wilson and E. Becker. 2018. *How to Teach Programming (and Other Things)*. Creative Commons. https://books.google.com/books?id=kv1swAEACAAJ

[77] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 143–146. doi:10.1145/1978942.1978963

[78] Sarah Shi Hui Wong and Stephen Wee Hun Lim. 2023. Take notes, not photos: Mind-wandering mediates the impact of note-taking strategies on video-recorded lecture learning performance. *Journal of Experimental Psychology: Applied* 29, 1 (March 2023), 124–135. doi:10.1037/xap0000375 Publisher: American Psychological Association.