ETH zürich

Department of Computer Science
Programming, Education, and
Computer-Human Interaction

Coducate: A Code Editor Extension to Streamline Instructor-Led Live Coding

Lukas Mast

Bachelor's Thesis

2024

Supervisors: Prof. Dr. April Yi Wang
PhD student Xiaotian Su

ABSTRACT

Live coding is a pedagogical technique used in programming education to demonstrate coding and problem-solving in real-time. While it engages students and enhances learning outcomes, it imposes a significant cognitive load on instructors who must simultaneously type, explain, debug, and manage classroom interactions. This thesis proposes Coducate, a code editor extension designed to streamline instructor-led live coding by providing supportive features for instructors. Coducate also improves the student experience by reducing the need for note-taking and simplifying in-class collaboration. Key functionalities include notes written directly inside the code editor and the use of AI code suggestions without participants noticing it. To evaluate the system's usability, a small usability study was conducted using the System Usability Scale (SUS) questionnaire and open-ended questions. The study resulted in a SUS score of 86.5, which is considered "excellent" according to industry benchmarks, indicating high user satisfaction.

ACKNOWLEDGMENT

This thesis is conducted under the guidance of the Programming, Education, and Computer-Human Interaction (PEACH) Lab at ETH Zürich. Special thanks to Prof. Dr. April Yi Wang and Xiaotian Su for their supervision. We acknowledge the use of GPT-based models in revising sections of this document.

1 Introduction

Live coding is a widely used teaching technique in programming education, especially in introductory courses. It involves instructors writing, debugging, and explaining code in real-time, providing students with a dynamic and transparent view of the coding process. This approach makes abstract programming concepts more tangible and helps students connect theory to practice.

However, live coding presents significant challenges. Instructors face a high cognitive load as they must simultaneously manage typing, explaining, debugging, and handling classroom interactions. Studies indicate that these demands can lead to errors, slower pace, and reduced teaching effectiveness (see **Appendix A.1**). Additionally, traditional live coding setups often lack effective collaboration mechanisms, leaving students as passive observers rather than active participants in the learning process.

To address these challenges, this thesis introduces Coducate, a code editor extension designed to streamline instructor-led live coding sessions. Coducate focuses on reducing instructors' cognitive load by automating routine tasks and providing tools that enable more effective collaboration. For example, instructors can grant students write access to code during sessions, allowing them to actively participate by contributing directly. Moreover, features like a diff editor for comparing code changes and the ability to integrate AI-powered code suggestions further enhance the live coding experience.

The main contributions of this thesis are: (1) the development of Coducate, a code editor extension tailored to address the unique challenges of live coding sessions by reducing cognitive load and enabling collaboration; (2) an empirical evaluation of Coducate's usability using the SUS and open-ended feedback, which demonstrated its potential to improve teaching and student engagement; and (3) insights into future directions for live coding tools, including strategies for enhancing collaboration.

By tackling the limitations of traditional live coding setups, Coducate aims to enhance the teaching experience for instructors while fostering greater student engagement and collaboration.

2 Related Work

This section explores existing tools and platforms for collaborative live coding, focusing on their capabilities, limitations, and how Coducate differentiates itself by addressing specific challenges faced in educational contexts. Prior work can be grouped into three main themes: (1) real-time collaborative coding tools, (2) live coding solutions for educational contexts, and (3) limitations and gaps in existing tools.

2.1 Real-Time Collaborative Coding Tools

A range of tools exists to support collaborative coding by enabling multiple users to work on the same codebase in real-time. Visual Studio Code's *Live Share* extension [1] allows participants to edit and debug shared code collaboratively. Similarly, JetBrains' *Code With Me* plugin [2] extends these capabilities to JetBrains integrated development environments (IDEs), supporting pair programming, multi-user editing, and debugging.

Recent research has examined how developers engage in real-time collaborative programming and the challenges associated with these tools. For instance, a study specifically focused on *Live Share* identified 18 usage scenarios, such as pair programming and code review, alongside 17 critical requirements, including live editing, terminal sharing, and mechanisms for focusing and following collaborators' actions [6]. However, the study also revealed significant

[1] https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare
[2] https://www.jetbrains.com/help/idea/code-with-me.html

challenges, such as lagging, permissions management, and conflict resolution, which hinder the effectiveness of *Live Share* for seamless collaboration.

2.2 Live Coding Tools for Educational Contexts

In the context of education, tools like *Improv* have been designed specifically for live coding in instructional settings [2]. Unlike general-purpose collaboration tools, *Improv* supports code-based presentations with features like synchronized code blocks, output with slides, and preset waypoints, informed by Mayer's principles of multimedia learning [3]. These features aim to reduce cognitive load, enable dynamic presentations, and support audience interaction. The findings emphasize the importance of tools tailored to instructional needs, highlighting a gap in existing general-purpose tools.

Browser-based platforms, such as *Replit*[3] and *Google Colab*[4], also offer live coding capabilities by removing the need for local installations. These platforms make coding activities more accessible and convenient by allowing students to participate using only a browser. However, they often require instructors to shift away from their preferred IDEs, such as Visual Studio Code, which, according to the 2024 Stack Overflow Developer Survey, is the most widely used IDE in 2024 [5]. Moreover, these platforms lack advanced classroom-specific features, such as distraction-free presentation modes and controlled mechanisms for managing student contributions during live coding.

2.3 Limitations and Gaps in Existing Tools

While tools like *Live Share*, *Code With Me*, and browser-based platforms address various aspects of collaboration and education, they exhibit notable limitations in classroom settings. For example, *Live Share* lacks advanced mechanisms for controlling or reviewing student contributions, making it less suitable for instructor-led sessions. Similarly, *Code With Me* requires software installation, which can pose barriers in diverse technical environments, and has restrictive licensing in its Community edition. Platforms like *Replit* and *Google Colab*, though accessible, require instructors to move away from familiar IDEs and lack tailored tools for managing live coding workflows.

2.4 How Coducate Addresses These Challenges

Coducate is explicitly designed for instructor-led live coding sessions in classrooms, bridging the gap between general-purpose tools and educational needs. It eliminates the need for software installation through browser-based accessibility, allowing students to participate seamlessly. Moreover, Coducate provides several features tailored to educational contexts. Instructors can review and manage student contributions effectively using a diff editor, ensuring control over the collaborative process. Distraction-free interfaces are integrated to enhance focus during live coding sessions, aligning with the needs of classroom environments. Additionally, Coducate incorporates AI-powered code suggestions, such as those provided by GitHub Copilot [5], enabling instructors to enhance productivity without students being aware of AI involvement.

By combining these features, Coducate enhances live coding workflows and addresses the unique challenges faced in educational contexts, providing an improved experience for both instructors and students.

[3] https://replit.com/collaboration
[4] https://colab.google/
[5] https://github.com/features/copilot

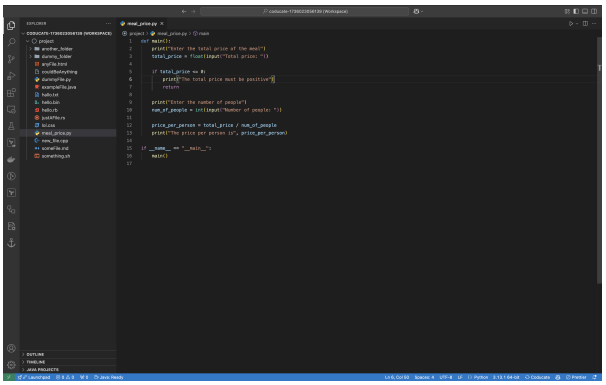
3 Design

3.1 Goals

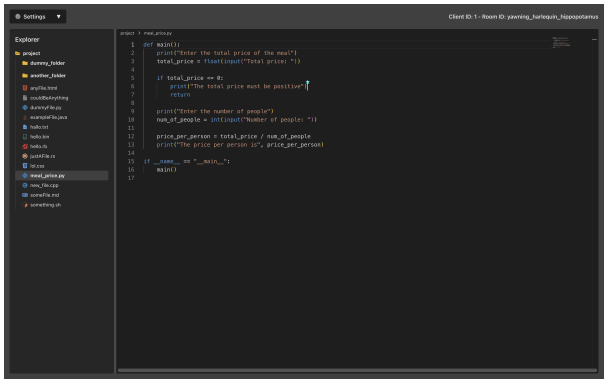
The design of Coducate focuses on reducing cognitive load for instructors while enhancing classroom collaboration. It aims to address the time-intensive and error-prone aspects of live coding by providing features that streamline the process and improve both teaching and learning experiences.

3.2 Structure

Coducate is designed to allow instructors to conduct live coding sessions entirely within Visual Studio Code without needing to modify the editor's interface for participant visibility. For example, instructors are not required to adjust font size, theme, or the width of the file explorer inside Visual Studio Code to accommodate participants. Inspired by the presentation view of PowerPoint, Coducate separates the instructor's view, referred to as the *editor view* (see **Figure 1a**), from the participants' view, referred to as the *web view* (see **Figure 1b**). The web view can be accessed on any device with a browser, offering flexibility for participants.



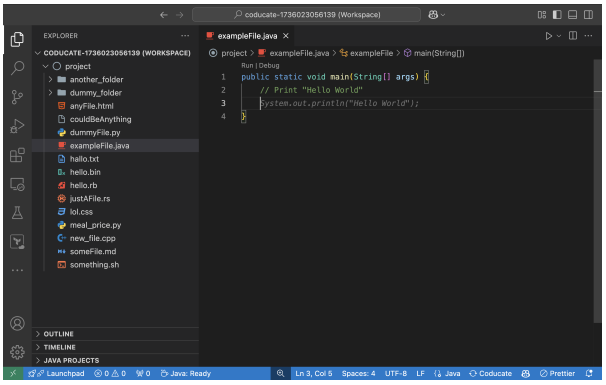
(a) Editor view (instructor).



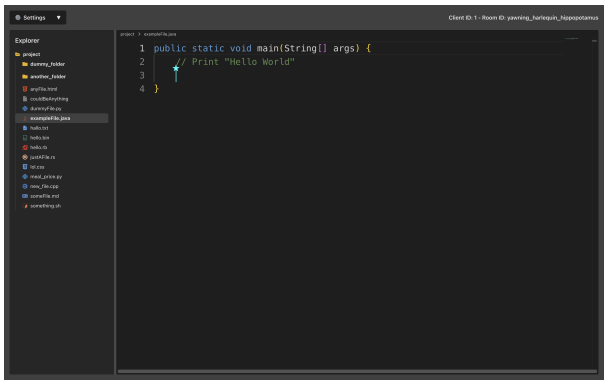
(b) Web view (participants).

Figure 1: Comparison between the editor view and the web view in Coducate.

This separation provides several advantages. Instructors can use coding aids, such as prewritten notes or AI code suggestions, without participants being aware of their use. Such tools can improve the flow and quality of the live coding session, ensuring a smoother experience for both instructors and participants (see **Figure 2**).



(a) Editor view with AI suggestions.



(b) Web view without AI suggestions.

Figure 2: AI code suggestions are visible in the editor view but not in the web view.

Moreover, while it is technically feasible to sync back code written by participants in the web view to the instructor's original Visual Studio Code file, this functionality has been intentionally excluded. Instead, Coducate includes a diff editor that allows the instructor to review, accept, or reject participant contributions selectively (see **Figure 3**). This design choice prevents potential disruptions, such as malicious edits or accidental deletions, while maintaining control over the coding session.

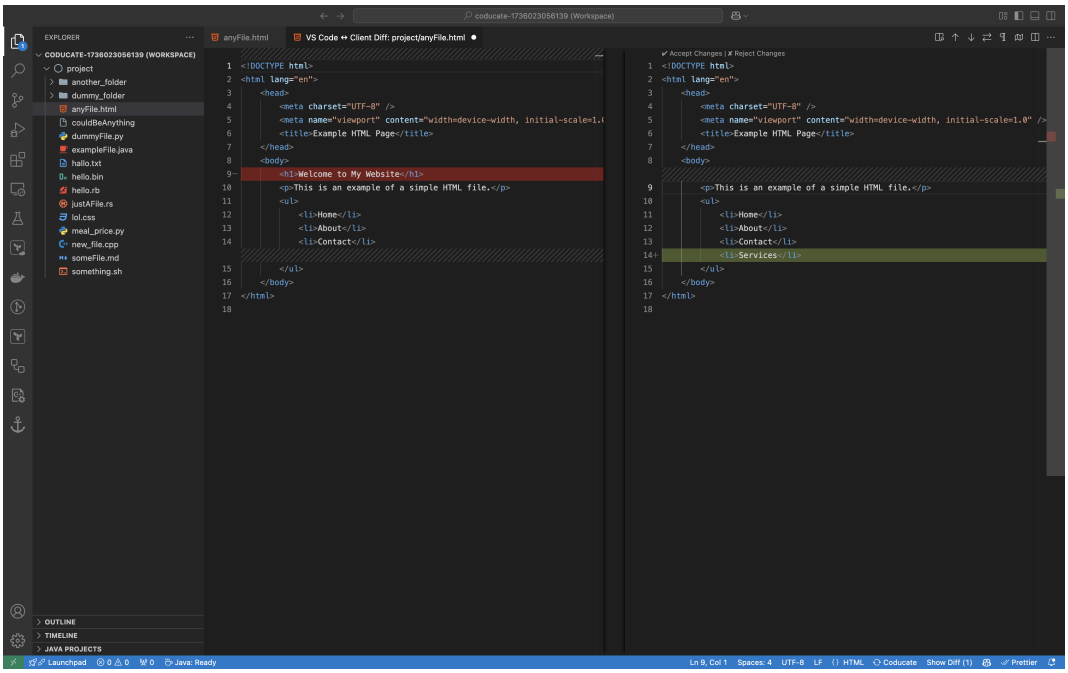
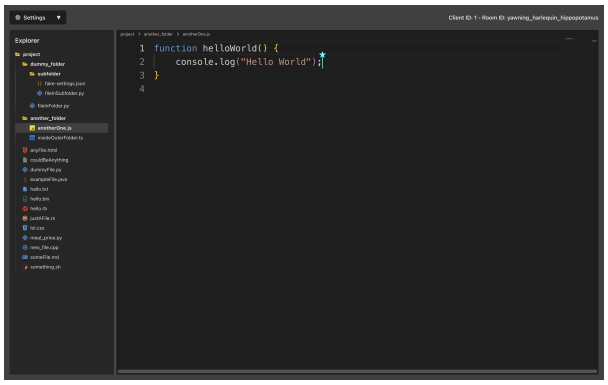


Figure 3: The diff editor in Coducate, allowing instructors to selectively accept or reject participant contributions.

Another benefit of the web view is its simplified and distraction-free interface. By hiding much of the code editor's user interface (UI), Coducate creates a cleaner live coding experience for participants, making it easier for them to focus on the content being taught. The web view is further divided into two types: the *participant's web view* (no room password required) and the *instructor's web view* (room password required). This distinction allows certain UI modifications to affect only the instructor's web view, ensuring the participant's view remains unaffected and personalized, for example, with their preferred font size or theme (see **Figure 4**).

(a) The instructor's web view updated to the light theme.



(b) The participant's web view remains in the dark theme.

Figure 4: The `Coducate: Change Theme` command updates the theme exclusively for the instructor's web view.

3.2.1 Workflow

Setup Phase

Instructor

- At the beginning of each live coding session (e.g., an exercise session or programming class), the instructor creates a room with a randomized room ID by using the `Coducate: Start Session` command in Visual Studio Code's Command Palette.
- The instructor then projects the web view for the participants to see. To join the newly created room, the instructor enters the room ID and selects the *I'm the instructor* option, which prompts them to enter the room password.

Participants

- Participants can decide whether to join the session locally on their own devices by entering the provided room ID or to simply observe the live coding on the instructor's projection. Those joining on their own devices will be assigned a unique client ID.

Live Coding Phase

Instructor

- After completing the setup, the instructor controls the live coding session directly within Visual Studio Code.
- The instructor's coding actions are displayed in real-time on all web views. If the instructor switches files in the code editor, the change is reflected in all web views that are following the instructor. All workspace operations, such as adding folders, creating, deleting, moving, or renaming files, are supported and will be updated accordingly in the web view's file explorer.
- The instructor can invite participants to collaborate by granting them write access. To do this, the instructor can request the participant's client ID and use the `Coducate: Grant Write Access` command. When a participant makes changes, the instructor will see a status bar icon such as *Show Diff (1)* next to the *Coducate* status bar icon, where the number in parentheses indicates the number of files with changes. By clicking on *Show Diff (1)* and selecting a file, the instructor can open a diff editor. The left side of the diff editor displays the original code from Visual Studio Code, while the right side shows the participant's changes from the web view. A Code Lens at the top of the diff editor provides options to either *Accept Changes* or *Reject Changes*. Selecting *Accept Changes* applies the participant's changes to the original file in Visual Studio Code, while choosing *Reject Changes* resets the web view to match the original file of the instructor in Visual Studio Code.
- The instructor can revoke write access from participants by using the `Coducate: Revoke Write Access` command.

Participants

- Participants accessing the session on their own devices can navigate through the files independently by selecting a file in the file explorer. This action will automatically disengage them from following the instructor. Consequently, any changes the instructor makes, such as switching files or scrolling within the same file, will not be reflected in the participant's web view. To resume following the instructor, participants can click the *Follow* button.

- By default, participants only have read access. They can ask the instructor to grant them write access.
- Each participant can adapt it's web view individually using the drop down menu.

Clean-up Phase

Instructor

- After completing the live coding session, the instructor can stop synchronization by using the `Coducate: End Session` command.
- Session-specific data, such as notes, will remain stored locally in Visual Studio Code's global store. To permanently delete a session, the instructor can use the `Coducate: Manage Sessions` command and select the session to remove.

Participants

- After the live coding session ends, participants can continue exploring the files on their own devices, as the code remains synced peer-to-peer. They can also download the workspace files as a ZIP by using the *Download Workspace* button from the dropdown menu.
- To disconnect from the session, participants can use the *Leave Room* button from the dropdown menu.

3.2.2 Commands

All the features implemented by Coducate can be accessed through Visual Studio Code's Command Palette. Each command follows the format `Coducate: <CommandName>`. The commands provided by Coducate are as follows:

`Coducate: Start Session` Starts a live coding session. The instructor can choose between a *New Session* or an *Existing Session*. If *New Session* is selected, the instructor is prompted to set an easy-to-remember room name and a room password. Additionally, the instructor can optionally select a *Task Description file* and/or a *Learning Goals file*.

`Coducate: End Session` Ends a live coding session.

`Coducate: Manage Sessions` Displays all previously created sessions. Upon selecting a session, the instructor can view the room password, rename the session, or delete the session.

`Coducate: Grant Write Access` Grants write access to participants. The instructor can either grant write access to a specific participant by entering the client ID or to all participants. Participants with write access can edit the code from their participant's web view.

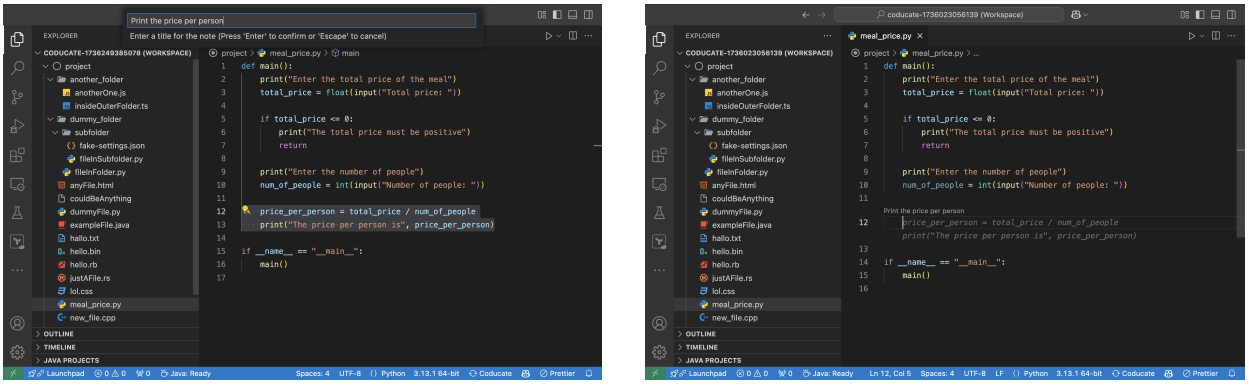
`Coducate: Revoke Write Access` Revokes write access from participants. The instructor can either revoke write access for a specific client by selecting their client ID from a list of clients with write access, or revoke access for all participants.

`Coducate: Emulate Terminal` Spawns a pseudo-terminal running `bash` for macOS and Linux, or `WSL` using `bash` for Windows. Input and output inside the pseudo-terminal are displayed in real-time on all web views. The emulated terminal in the web view is always read-only, allowing users to observe the terminal activity without directly interacting with it. The terminal closes both in the web view and in Visual Studio Code when the `exit` command is used.

`Coducate: Create Note` Requires some consecutive lines of code to be selected. Once a selection is made, the instructor is prompted to provide a name for the note. The selected lines are removed from the codebase and replaced with a Code Lens displaying the note name (see **Figure 5**). The note can be used in the following ways:

1. Word by word using `Cmd + →` on macOS or `Ctrl + →` on Windows and Linux.
2. Line by line using `Cmd + Shift + →` on macOS or `Ctrl + Shift + →` on Windows and Linux.
3. All at once using the `Tab` key.

By clicking on the Code Lens of a created note, its content can be inserted at the cursor's position.



(a) Creating a note using `Coducate: Create Note`.

(b) Inline code suggestions based on the created notes.

Figure 5: Inline code suggestions derived from the created notes.

`Coducate: Remove Notes` Prompts the instructor to remove notes either from the currently open file or from the entire workspace. Additionally, specific notes can be deleted by clicking on their corresponding Code Lens.

`Coducate: Toggle Suggestions` Enables or disables inline suggestions from notes. This feature is useful if the instructor wants to use AI code suggestions (e.g., GitHub Copilot). When both are active, their inline suggestions are merged. Inline suggestions can also be toggled on or off using `Ctrl + Shift + U`.

Affects Instructor and Participant Web Views:

`Coducate: Open Terminal` Opens the terminal in the web view.

`Coducate: Close Terminal` Closes the terminal in the web view.

Affects Only the Instructor's Web View:

`Coducate: Change Font Size` Allows the instructor to increase or decrease the font size in the code editor and the emulated terminal of the web view.

`Coducate: Change Theme` Enables the instructor to change the web view's theme by selecting either Dark or Light mode.

`Coducate: Open Explorer` Opens the file explorer in the web view.

`Coducate: Close Explorer` Closes the file explorer in the web view.

3.3 Implementation Details

The project comprises a frontend, a backend, and the Visual Studio Code extension, each managed in its own GitHub repository. The frontend and the backend are both running in separate Docker containers. There is a third Docker container providing a MySQL database to store the room data. The three parts of Coducate communicate using web sockets (see **Figure 6**).

3.3.1 Frontend

The frontend is set up and written using Vite together with React and TypeScript (TSX components). It is a single-page application with a combined total of 2765 lines of user-written code (excluding blank lines, comments, and setup files).

3.3.2 Backend

The backend is written in TypeScript using Express as the backend web application framework. The total amount of user-written lines of code is 739 (excluding blank lines, comments, and setup files).

3.3.3 Visual Studio Code extension

The Visual Studio Code extension is written in TypeScript using esbuild as a module bundler. This part of the project was scaffolded using *Yeoman* [6] together with the NPM package *generator-code* [7]. Real-time synchronization of code is achieved through event listeners provided by the Visual Studio Code API, which work in conjunction with *Yjs* [8], a shared peer-to-peer editing framework. *Yjs* provides shared data structures that are automatically synchronized, enabling seamless collaboration. All user-written extension files (excluding blank lines, comments, and setup files) have a combined length of 3425 lines of code.

4 Methodology

The design and development process for Coducate followed an iterative approach, encompassing research, prototyping, development, and evaluation phases.

4.1 Development

The development of Coducate began with a thorough review of scientific literature on live coding as a pedagogical method. This was followed by a need-finding and brainstorming phase to identify key pain points faced by instructors during live coding sessions and to generate ideas for addressing these challenges. As part of the need-finding process, lecture observations were conducted in sessions where live coding was practiced, allowing us to identify common issues and better understand the challenges instructors face in real-time teaching environments.

Recognizing the importance of accessibility and extensibility, we chose to develop a code editor extension for Visual Studio Code, as it is the most popular IDE according to the 2024 Stack Overflow Developer Survey. We then explored the technical capabilities of Visual Studio Code extensions, including UI modifications and potential limitations, to ensure feasibility.

[6] https://yeoman.io/
[7] https://www.npmjs.com/package/generator-code
[8] https://yjs.dev/

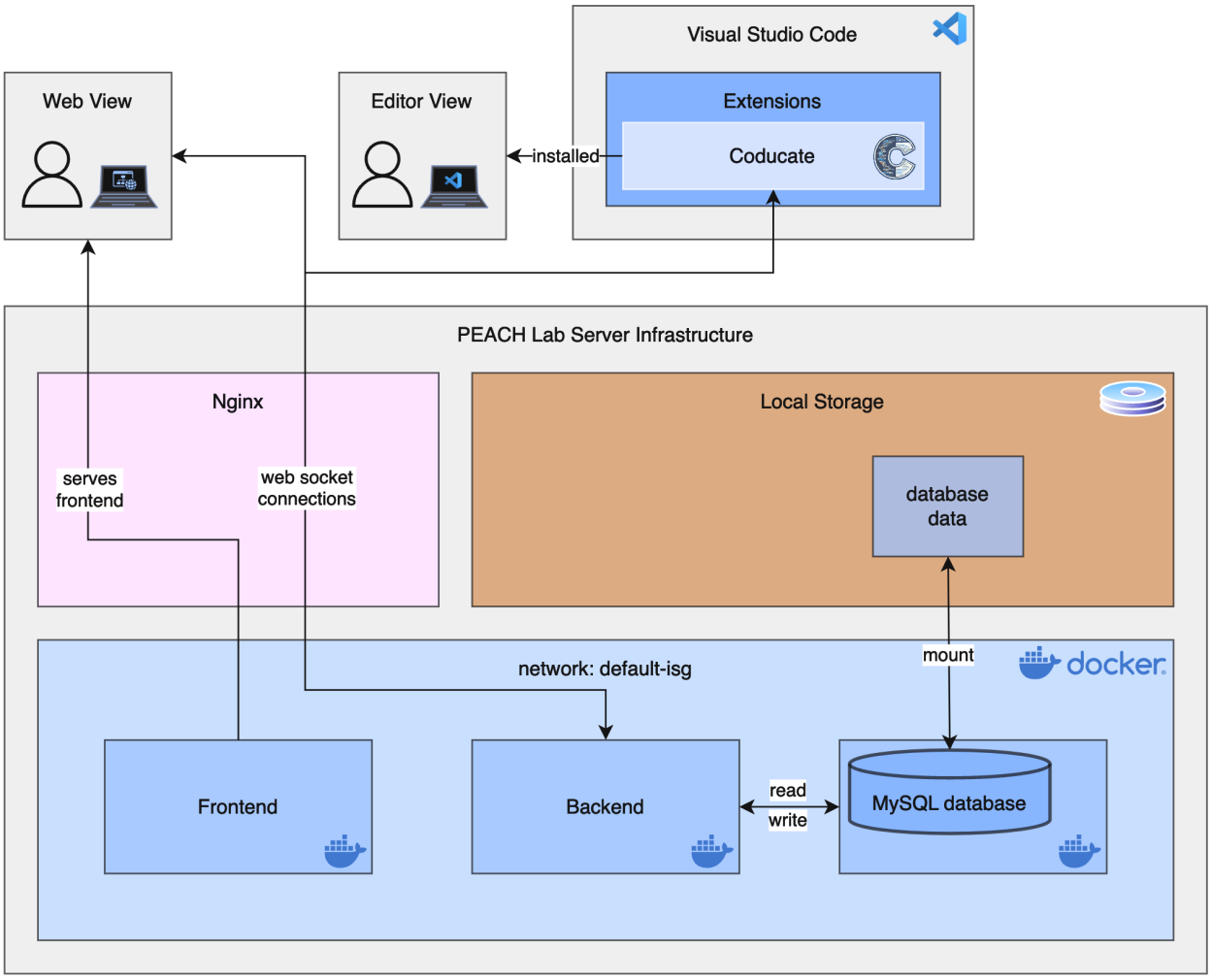


Figure 6: Architecture of Coducate showing how the frontend, backend, and database work together. The diagram illustrates the communication between the web view (frontend), editor view (Visual Studio Code extension), backend, and MySQL database using web sockets.

To address identified pain points, we conceptualized a PowerPoint-inspired presentation mode using a web view, designed to separate the instructor's view from that of participants. This approach enabled us to implement features that would not have been possible with a standard Visual Studio Code extension.

The next step involved creating initial paper prototypes, which quickly evolved into low-fidelity prototypes in Figma [9]. Development proceeded iteratively, focusing on three core components: the frontend, backend, and the Visual Studio Code extension. These components were developed in a round-robin fashion, with extensive testing on localhost to ensure seamless integration.

In the final phase, the frontend and backend were containerized and deployed on the PEACH Lab web server at ETH Zürich. Coducate [10] was published on Microsoft's Extension Marketplace to make it accessible to all users.

[9] https://www.figma.com/
[10] https://marketplace.visualstudio.com/items?itemName=coducate.coducate

4.2 Usability Study

To evaluate the usability and effectiveness of Coducate, a usability study was conducted involving five computer science students. The study design and execution were approved by the ETH Zürich Ethics Commission without reservations.

4.2.1 Study Design and Objectives

The primary objective of the study was to assess whether Coducate effectively reduces the instructor's mental load and improves convenience during live coding sessions. The hypothesis was that Coducate significantly enhances the usability and experience of live coding for instructors.

Study participants engaged in a controlled classroom environment where one participant acted as the instructor, while others played the role of students. Each participant took turns being the instructor, with the researcher also participating as a student. Observational notes were taken during the sessions, and post-session surveys, including the SUS (see **Appendix B.1**) and open-ended questions (see **Appendix B.2**), were administered to gather feedback. No student learning data was collected.

4.2.2 Participant Recruitment and Inclusion Criteria

The study involved five participants, all computer science students with prior coding experience. Participants were recruited through direct invitations. Exclusion criteria included individuals with no coding experience or those unable to attend in person.

4.2.3 Data Collection Methods

Data collection methods included:

- Observational notes documenting participant interactions with Coducate.
- A post-session usability survey using the SUS to evaluate the tool's user experience.
- Open-ended questions to collect qualitative feedback, including suggestions for new features and recommendations for improving existing functionalities.

4.2.4 Deployment and Procedure

The study was conducted in a controlled classroom setup. Participants alternated roles between instructor and student, using Coducate for live coding sessions. At the end of the session, participants completed the usability survey.

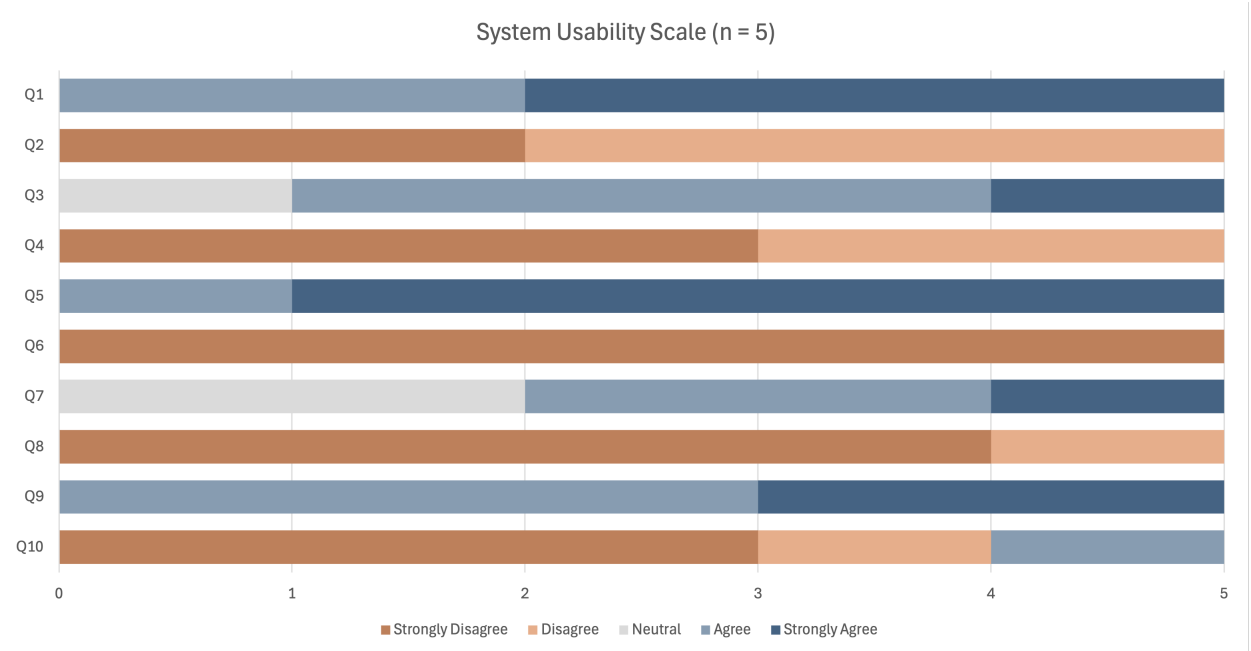
This methodology ensured a structured approach to designing, developing, and evaluating Coducate, with insights gathered from both technical development and participant feedback.

5 Results

The usability study of Coducate involved five participants, all of whom tested the system in a controlled classroom environment. Data was collected through observational notes, the SUS questionnaire, and open-ended qualitative feedback.

5.1 Quantitative Results

The SUS questionnaire yielded an average score of 86.5, which is considered "excellent" according to industry benchmarks (see **Appendix B.3**). This high score indicates strong user satisfaction and usability of the system (see **Figure 7**). Observational data also revealed that Coducate effectively mitigated many common issues frequently encountered during our observations of lectures where live coding was practiced.



Q1:	I think that I would like to use Coducate frequently.	
Q2:	I found Coducate unnecessarily complex.	*(Reverse scored)*
Q3:	I thought Coducate was easy to use.	
Q4:	I think that I would need the support of a technical person to be able to use Coducate.	*(Reverse scored)*
Q5:	I found the various functions in Coducate were well integrated.	
Q6:	I thought there was too much inconsistency in Coducate.	*(Reverse scored)*
Q7:	I would imagine that most people would learn to use Coducate very quickly.	
Q8:	I found Coducate very cumbersome to use.	*(Reverse scored)*
Q9:	I felt very confident using Coducate.	
Q10:	I needed to learn a lot of things before I could get going with Coducate.	*(Reverse scored)*

Figure 7: SUS evaluation results for Coducate with five study participants. The results demonstrate strong user satisfaction, with an average score of 86.5 categorized as "excellent" according to industry benchmarks.

5.2 Qualitative Results

Open-ended questions provided valuable insights into user experience and potential improvements. Participants highlighted the convenience of having notes integrated directly into the code editor and appreciated that AI code suggestions, such as those provided by GitHub Copilot, could be utilized discreetly without students noticing their use during live coding. Another feature that participants highly valued was the diff editor view, which provides a clear side-by-side comparison of changes and allows instructors to easily accept or reject changes made by participants.

5.3 Emergent Themes

From the qualitative feedback, several themes emerged:

- **Enhanced Productivity**: The streamlined workflow helped reduce cognitive load, enabling instructors to focus on teaching and problem-solving more effectively.
- **Improved Collaboration**: Study participants highlighted the ability for the instructor to grant write access as a significant improvement for collaboration. This feature allowed participants to write code directly, saving time and reducing confusion that often arose from verbal dictation. They noted that it addressed issues such as unclear verbal instructions, difficulty hearing quieter participants, or confusion about which line of code was being referenced, making the collaborative process much smoother.
- **Suggestions for Future Enhancements**: Proposed enhancements included differentiating code suggestions by using distinct colors for those generated by Copilot versus those from Coducate's notes, adding a graphical user interface (GUI) for commands to reduce reliance on Visual Studio Code's Command Palette, and making smaller adjustments to the UI of the web view to improve usability.

These results collectively demonstrate the potential of Coducate to enhance instructor-led live coding sessions by addressing common challenges and improving both the instructor and student experience.

6 Discussion

The findings of this study suggest that Coducate effectively addresses significant challenges faced by instructors during live coding sessions. By streamlining routine tasks, such as managing files and organizing notes, Coducate reduces cognitive load and allows instructors to focus on teaching and student interaction. This aligns with the study's hypothesis that Coducate would enhance the usability and experience of live coding for instructors.

One particularly noteworthy result was the positive reception of the feature that allows instructors to grant write access to students. This capability eliminated the common issue of students verbally dictating code, which often led to confusion or delays. Participants appreciated the smoother collaboration process, which saved time and minimized misunderstandings, particularly in identifying the intended code lines or handling unclear verbal instructions. The study participants also highlighted the utility of the diff editor view, which provided a clear, side-by-side comparison of changes and enabled instructors to easily accept or reject contributions.

The usability study's SUS score of 86.5 further underscores the tool's effectiveness in improving the live coding experience. This high score reflects strong user satisfaction and validates Coducate as a tool that meets the needs of its intended users. Additionally, the ability to use AI code suggestions, such as those provided by GitHub Copilot, without students noticing their use was seen as a significant advantage by instructors.

Despite these successes, the study revealed areas for improvement. Participants suggested enhancements, such as differentiating colors for code suggestions from Copilot and Coducate's notes, adding a GUI for commands to reduce reliance on the Visual Studio Code Command Palette, and making minor adjustments to the web view's UI. These proposed features indicate opportunities for future iterations to improve the user experience.

6.1 Limitations

While the study provides promising results, it has certain limitations. First, the dependency on Visual Studio Code may restrict adoption among instructors who prefer other IDEs. Second, the study involved a small sample size of five participants, which limits the generalizability of the findings. Moreover, none of the study participants were actual instructors. They merely acted as instructors in this controlled classroom environment, which may not fully capture the complexities of real-world teaching scenarios. Future studies should include a larger and more diverse group of participants to validate the results further. Lastly, while the study demonstrated the tool's effectiveness in a controlled classroom environment, real-world variability in teaching contexts could reveal additional challenges or limitations.

6.2 Future Directions

The study highlighted several areas where Coducate could be further improved. Enhancing its UI, adding features like differentiated colors for code suggestions from GitHub Copilot and Coducate's notes, and introducing a GUI for commands were among the suggestions from participants. These changes could enhance the usability and accessibility of the tool. Additionally, expanding the user base with a larger-scale usability study would help validate the findings and ensure the tool's adaptability across diverse teaching contexts.

In conclusion, Coducate demonstrates potential to transform instructor-led live coding by addressing key pain points and fostering a more efficient and collaborative environment.

7 Future Work

A key direction for future work is to expand Coducate's compatibility to include a wider range of code editors and development environments. This enhancement would make the tool more accessible to instructors who prefer platforms other than Visual Studio Code.

Another promising avenue for future work is enhancing collaboration features for participants. One idea is to introduce a new mode distinct from live coding, where participants can write code locally without it being synced with others. This would allow the instructor to let participants work independently on coding tasks and later "publish" a solution. The published solution would include a diff view in the participants' web view, showing the instructor's code alongside their own, helping them understand differences and learn from the process.

Additional collaboration-focused features could include interactive exercises, such as showing participants a set of code snippets (one correct and others incorrect) and asking them to select the correct snippet. This would foster engagement and active learning during live coding sessions.

Finally, incorporating more features powered by large language models (LLMs) offers exciting possibilities. For example, when participants are new to coding, best practices could be identified during live coding with the help of an LLM. These best practices could then be displayed in an info box on the participants' web view, enhancing their learning experience.

Long-term studies could help us understand how Coducate impacts teaching and learning over time. These studies could explore whether it consistently improves student engagement, instructor efficiency, and classroom dynamics in different settings.

8 Conclusion

This thesis introduced Coducate, a code editor extension designed to streamline and enhance the experience of instructor-led live coding sessions in programming education. By addressing key challenges such as the cognitive load on instructors and the barriers to effective collaboration, Coducate enhances the teaching experience and student engagement, ultimately contributing to the field of educational technology.

This work also provides an empirical foundation for the usability and effectiveness of Coducate. With a SUS score of 86.5, the tool demonstrates strong user satisfaction and practical value. However, the results of this study should be interpreted with caution, as the number of participants was limited to five, and none of them were actual instructors; they merely acted as instructors in a controlled classroom environment. The feedback gathered during the study highlights opportunities for further refinement and underscores the potential of such tools to improve programming education.

Ultimately, this work aims to inspire further research and innovation, contributing to the development of technologies that meaningfully support both educators and learners.

A Literature Review

A.1 Benefits and Drawbacks of Live Coding

Perceived benefits of live coding	# of studies	Studies which perceived these benefits
Improves debugging skills	9	[3, 4, 15, 31, 35, 36, 39, 41, 44]
Exposes programming as a process	8	[3, 17, 22, 35, 36, 41, 44, 45]
Increases student engagement	7	[6, 17, 31, 35, 36, 39, 42]
Teaches how to apply programming concepts	5	[16, 21, 31, 36, 47]
Improves testing skills	5	[1, 3, 22, 36, 39]
Teaches incremental Coding	5	[3, 22, 35, 39, 44]

Table 1: The six most frequently perceived benefits of live coding across all reviewed studies. Source: [4]

Perceived drawbacks of live coding	# of studies	Studies which perceived these drawbacks
Relatively time-consuming	4	[6, 17, 36, 39]
Hard for students to take notes	1	[45]
Hard for students to keep up with the pace of programming	1	[17]

Table 2: The perceived drawbacks of live coding across all reviewed studies. Source: [4]

B Study

B.1 System Usability Scale questionnaire

The following survey was used to evaluate the usability of Coducate using the SUS. Participants were asked to rate the following statements by selecting one of the options: Strongly Disagree, Disagree, Neutral, Agree, Strongly Agree.

1. I think that I would like to use Coducate frequently.

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

2. I found Coducate unnecessarily complex. *(Reverse scored)*

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

3. I thought Coducate was easy to use.

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

4. I think that I would need the support of a technical person to be able to use Coducate. *(Reverse scored)*

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

5. I found the various functions in Coducate were well integrated.

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

6. I thought there was too much inconsistency in Coducate. *(Reverse scored)*

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

7. I would imagine that most people would learn to use Coducate very quickly.

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

8. I found Coducate very cumbersome to use. *(Reverse scored)*

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

9. I felt very confident using Coducate.

   ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

10. I needed to learn a lot of things before I could get going with Coducate. *(Reverse scored)*

    ☐ Strongly Disagree ☐ Disagree ☐ Neutral ☐ Agree ☐ Strongly Agree

B.2 Open-ended questions

In addition to the SUS questionnaire, study participants were asked the following open-ended questions:

- How did Coducate impact your mental workload during live coding?
- What specific features of Coducate did you find most useful?
- Were there any features you found confusing or difficult to use? Please elaborate.
- How did Coducate compare to other tools or methods you have used for live coding?
- Would you recommend Coducate to other instructors? Why or why not?
- Do you think Coducate improved the overall experience of live coding for students?
- What improvements or additional features would you suggest for Coducate?

B.3 System Usability Scale: Adjective Ratings and Scoring Scales

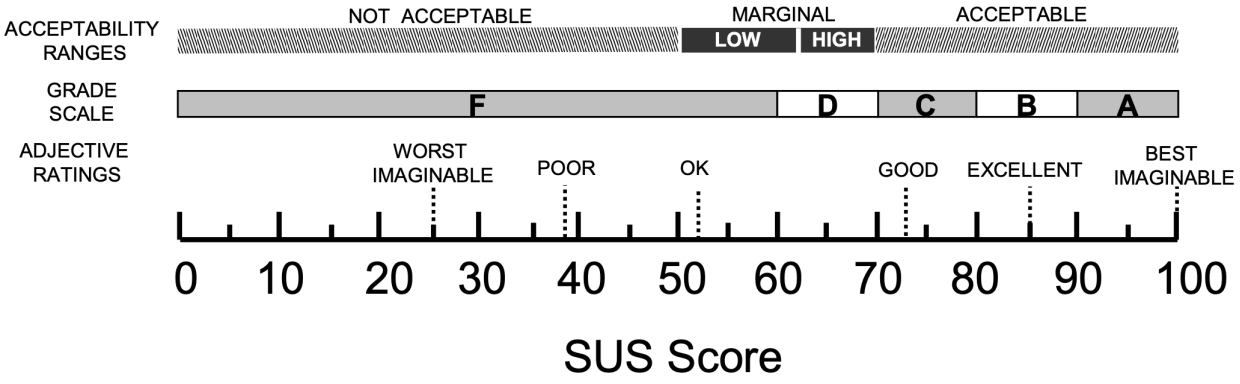


Figure 8: A comparison of the adjective ratings, acceptability scores, and school grading scales, in relation to the average SUS score. Source: [1]

References

[1] Aaron Bangor, Philip Kortum, and James Miller. 2009. Determining what individual SUS scores mean: adding an adjective rating scale. *J. Usability Studies* 4, 3 (May 2009), 114–123. https://dl.acm.org/doi/pdf/10.5555/2835587.2835589

[2] 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* (Chicago, IL, USA) *(L@S '19)*. Association for Computing Machinery, New York, NY, USA, Article 9, 10 pages. https://doi.org/10.1145/3330430.3333627

[3] Richard E. Mayer. 2001. *Multimedia Learning*. Cambridge University Press, Cambridge.

[4] 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

[5] Stack Overflow. 2024. Technology | 2024 Stack Overflow Developer Survey. https://survey.stackoverflow.co/2024/technology#most-popular-technologies-new-collab-tools Accessed: 2024-12-31.

[6] Xin Tan, Xinyue Lv, Jing Jiang, and Li Zhang. 2024. Understanding Real-Time Collaborative Programming: A Study of Visual Studio Live Share. *ACM Trans. Softw. Eng. Methodol.* 33, 4, Article 110 (April 2024), 28 pages. https://doi.org/10.1145/3643672