Helping job seekers prepare for technical interviews by enabling context-rich interview feedback

Yi Lu

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Sang Won Lee, Chair
Yan Chen
Dwayne Brown

May 7, 2024
Blacksburg, Virginia

Keywords: technical interviews, collaborative editing, real-time feedback

Helping job seekers prepare for technical interviews by enabling context-rich interview feedback

Yi Lu

(ABSTRACT)

Technical interviews have become a popular method for recruiters in the tech industry to assess job candidates' proficiency in both soft skills and technical skills as programmers. However, these interviews can be stressful and frustrating for interviewees. One significant cause of the negative experience of technical interviews was the lack of feedback, making it difficult for job seekers to improve their performance progressively by participating in technical interviews. Although there are open platforms like Leetcode that allow job seekers to practice their technical proficiency, resources for conducting mock interviews to practice soft skills like communication are limited and costly to interviewees. To address this, we investigated how professional interviewers provide feedback if they were conducting a mock interview and the difficulties they face when interviewing job seekers by running mock interviews between software engineers and job seekers. With the insights from the formative studies, we developed a new system for technical interviews aiming to help interviewers conduct technical interviews with less cognitive load and provide context-rich feedback. An evaluation study on the usability of using our system to conduct technical interviews further revealed the unresolved cognitive loads of interviewers, underscoring the requirements for further improvement to facilitate easier interview processes and enable peer-to-peer interview practices.

Helping job seekers prepare for technical interviews by enabling context-rich interview feedback

Yi Lu

(GENERAL AUDIENCE ABSTRACT)

Technical interview is a common method used by tech companies to evaluate job candidates. During these interviews, candidates are asked to solve algorithm problems and explain their thought processes while coding. Running these interviews, recruiters can assess the job candidate's ability to write codes and solve problems in a limited time. At the same time, the requirements for interviewees to talk aloud help interviewers evaluate their communication and collaboration skills. Although technical interviews enable employers to assess job applicants from multiple perspectives, they also introduce interviewees to stress and anxiety. Among the many complaints about technical interviews, one significant difficulty of the interview process is the lack of feedback from interviewers. As a result, it is difficult for interviewees to improve progressively by participating in technical interviews repeatedly. Although there are platforms for interviewees to practice code writing, resources like mock interviews with actual interviewers for job seekers to practice communication skills are costly and rare. Our study investigated how professional programmers run mock technical interviews and provide feedback when required. The mock interview observations helped us understand the standard procedure and common practices of how practitioners run these interviews. At the same time, we concluded the potential cause of cognitive loads and difficulties for interviewers to run such interviews. To answer the difficulties of conducting technical interviews, we developed a new system that enabled interviewers to conduct technical interviews with less cognitive load and provide enriched feedback. After rerunning

mock interviews with our system, we noted that while some features in our system helped make the interview process easier, additional cognitive loads are unresolved. Looking into these difficulties, we suggested several directions for future studies to improve our design to enable an easier interview process for interviewers and support interview rehearsals between job seekers.

Contents

List of Figures

Chapter 1

Introduction

The early state of technical interviews can be traced back to the puzzle interviews popularized by Microsoft, where interviewers test the interviewees' problem-solving skills when facing unexpected situations with brain-teasing questions [24]. In today's IT companies, technical interviews have a significant impact on the interview lifecycle of recruiting and screening job candidates for software engineering positions[35]. Although the process might vary between companies, it usually involves writing code to solve algorithm problems either on whiteboards or text editors[28]. By observing candidates think aloud while solving problems with programmed solutions, interviewers can assess their abilities from different perspectives, including code writing, problem-solving skills, and communication. However, while being a popular assessment process among recruiters, many voices in the industry have expressed their concerns about the technical interview process, describing it as a "leaky pipeline" that introduces anxiety and discouragement before, during, or after the interview[17, 18, 29]. With the fact that technical interviews are the industrial norm for hiring, better ways to help job seekers prepare for these interviews to overcome stress and perform better become an unstopping need.

While studying previous works, we noticed that one of the main factors of stress is the worry of lack of feedback and disengagement with the interviewer during the technical interview [17, 18]. This agrees with the industry standard that feedback is not provided after the interview, and evaluation notes are reported only to the recruitment boards. Although job

candidates can benefit from receiving feedback after interviews to progressively improve their performance while participating, the primary objective of interviewers is to assess candidates' proficiency, not to facilitate their personal development[16].

Since getting feedback from actual interviews is difficult, platforms to practice solving algorithm problems with code gain their audiences. However, as a previous study stated, students' performance in solving algorithm problems can be reduced when doing it publicly under assessment compared to doing it privately because of the students' stress from talking aloud[19]. Therefore, practicing privately by solving algorithm problems can be less effective than expected. With the difficulty of getting immediate feedback and improvement from actual interviews, mock interviews become a good way to prepare for technical interviews by practicing communication to reduce anxiety about talking while programming[16]. In mock interviews, although job seekers could gain the most realistic experience of practicing with actual professional engineers who would be interviewing them in real interviews, the resource of practice sessions with professional engineers are costly[6].

Therefore, the difficulty of getting feedback from real interviews, the lack of practice from practicing individually, and the cost of getting real mock interview experiences from practitioners highlighted the potential of peer-to-peer mock interviews. While feedback is normally absent in the industry setting, it is important to study the difficulties of conducting technical interviews and provide feedback for professional interviewers. After completing a standard procedure for professionals and overcoming difficulties in providing feedback, we can enable feedback exchange in practice interviews where job seekers play the role of the interviewer.

Thus, the goal of this study is to understand how we can design an online interview platform that can facilitate interviewers' feedback in the context of mock interviews. From observations over a series of several formative studies, we noticed several difficulties in the current technical interviews and developed a system to assist interviewers in providing feedback with

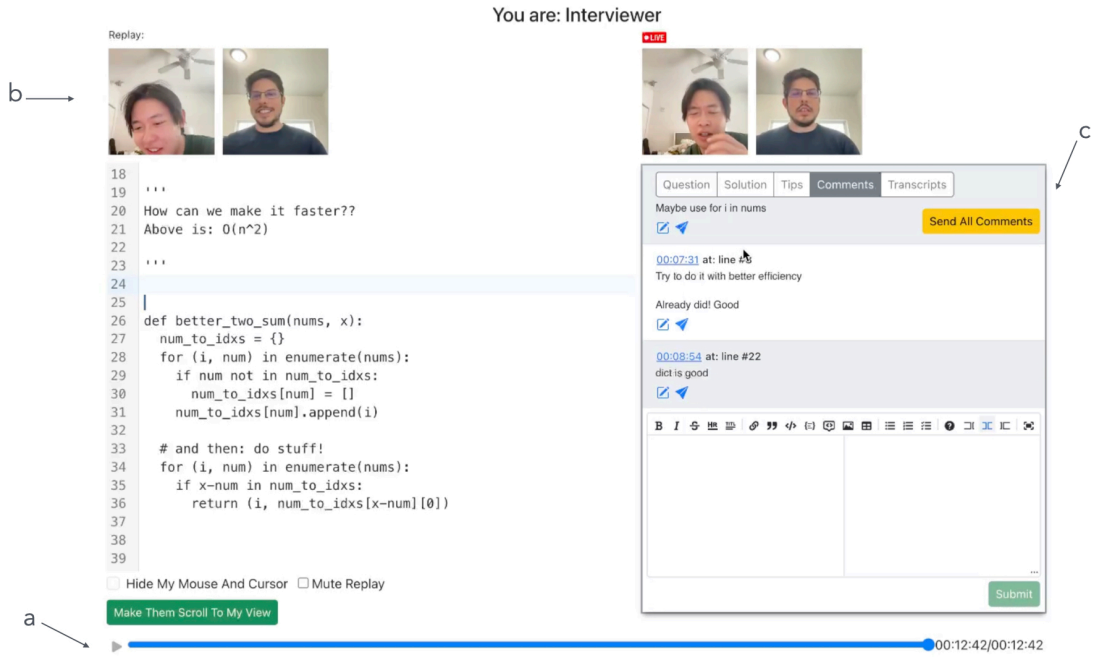

Figure 1.1: Interface of our system after the technical interview when interviewers give feedback. (a)Timeline for interviewers to control the relay of code changes and video recording of their webcam during the interview. (b)The replay of the webcams of the interviewer and interviewee will appear on the left side of the current webcams. (c)The supportive tools for interviewers to conduct the interview and give feedback. The current tab is the comment panel where interviewers can leave comments tied with timestamps and code segments while running the interview. While giving feedback, interviewers can click the timestamp in each comment to navigate the replays.

less cognitive load. Our system includes a shared editor, video calling, and an interviewer tool panel (as shown in figure 1.1. The tool panel includes a shared editor for interview questions, solutions to the interview problem, and assisting information for the interviewers to conduct the interview. During the interview, the video calling and changes made to the code editor will be recorded. The interviewer can leave comments and take notes on the interviewee's performance in the comments tab in the interviewer's tool panels. The note can be connected with the selected code segments and the timestamp at which it was taken. With this, interviewers can choose to replay the code changes and discussions and use the comments to navigate through the replays, adding context to their feedback. With the com-

ments panel and interview guides alongside the code editor, the goal of our system is to minimize the context switching during the interview while saving contexts for interviewers.

After developing the system, we ran four evaluation studies with participants from the same pool as our formative study. The setting of the evaluation study remains the same as our formative study, except for the additional time during each mock interview to introduce the functionalities of our system to the participants. Similar to the formative study, we ran a semi-structured interview with the participants to understand their experiences conducting mock technical interviews using our system. In our evaluation, we found that visual cues of peer presence can help interviewers and interviewees match their area of focus. At the same time, the potential of real-time replays of code changes was not fully discovered because the cognitive load to multitask during the mock interview hindered interviewers from taking notes while communicating with interviewees. This work contributes:

1 Get insights into the understudied mental requirements of interviewers when conducting technical interviews.

2 Grow in understanding how to design systems to support feedback exchange in technical interviews by saving contexts.

3 Direction for future studies to design feedback exchange platforms to facilitate peer-to-peer technical interview practices

Chapter 2

Review of Literature

2.1 Technical interviews

2.1.1 Stress of technical interviews

Although widely employed as a crucial part of the recruitment process in the tech industry, the technical interview process has its underlying problems. Behroozi thoroughly analyzed discussions related to technical interviews on Hacker News, an online technical community, to find out what job candidates are concerned about technical interviews. Their result indicates that technical interviews can be unnecessarily stressful and biased, negatively impacting candidates' self-esteem [17]. To alleviate the stress of a technical interview, Behroozi experimented by comparing the interviewee's performance when doing think-aloud with or without supervision. The result showed that the stress of being supervised while solving a technical problem will bring unnecessary mental loads to the interviewees [19]. Therefore, new designs and innovations in the technical interview system can improve the current platforms by reducing interviewees' sense of being supervised.

2.1.2 Absent of feedback in technical interviews

Behroozi's other study on the interview reviews posted on Glassdoor revealed specific areas and reasons behind job candidates' experiences in the hiring process, whether positive or negative. Among their many findings, it is noticeable that candidates might feel frustrated when interviewers are not engaged or professional enough to provide necessary hints, which agrees with their previous study. In addition to that, their results also showed that job candidates appreciate feedback from interviewers, especially when they are rejected. Without constructive feedback, the job candidates are frustrated as they cannot figure out why they are not proficient for the position and how to improve for future interviews [17, 18]. Despite the negative impacts of the absence of feedback, as HR practitioners showed, feedback is expected to be uncommon in the hiring process because of the time required to compose formal interview feedback and the liability of their feedback[23]. While it might agree with the industry setting that interviewers are not required to provide feedback to interviewees, valuable suggestions from interviewers are essential in practice settings.

2.1.3 What are interviewers looking for

Previous works revealed valuable non-technical skills required to become a successful programmer, including communication, problem-solving, and adapting to ambiguities or real-time changes [15, 31]. Agreeing with the general proficiency of a good software engineer, studies on the employer's need for job candidates showed that employers and technical interviewers value communication skills and problem-solving skills [21, 34, 35]. To overcome the stresses of having to talk aloud while solving the problem [22], as suggested by technical interview guides, one of the best ways to practice communication is to conduct mock interviews to get comfortable communicating while solving the technical problem[16]. Previous work by

Kapoor experimented with employing mock interviews in an undergraduate computer science course. This experiment yields positive results in enhancing the confidence of students in performing well in technical interviews[26]. However, job seekers must commit massive amounts of effort and time to prepare for technical interviews outside of such class environments, which are not necessarily beneficial to their personal technical growth[17]. Existing platforms like interviewing.io and hellointerview.com provide dedicated career coaching from actual software engineer practitioners. However, such services would charge over a hundred dollars per session[6, 14]. Therefore, the time and material cost for job seekers to practice and the difficulties of getting improvements while doing actual interviews made feedback for each interview session more valuable.

2.1.4 Requirements for interviewers

According to Behroozi's studies, job seekers get frustrated when the interviewers become unengaged and fail to provide necessary hints when needed[17, 18]. Another study by Lunn surveying undergraduate students on their technical interview experiences also showed that one of their most crucial negative experiences was the interviewers failed to guide the interview[30]. While interviewees reported that interviewers can appear uninterested and fail to give professional help when solving the interview problem [18], it is also noticeable that the interviewers are doing time-sensitive work in front of the public. As previous work by Klitzman suggested, time pressures, demands from co-workers, and the feeling of being criticized by co-workers can be major factors of work stress [27]. Although interviewers might also experience these stresses, there is a gap in the study of their cognitive loads and pressures during technical interviews.

2.2 Link feedback with context

Feedback is considered an important part of learning and the creative process. Many previous works dedicated to enhancing feedback provide inspiration on how to design a system enabling enriched feedback. Such improved feedback is crucial because non-textual interactions and synchronizations can be vague when many of these feedback exchanges happen online.

One example would be Pavel's previous work on a collaborative video review system. With such a system, the user can play videos and record verbal feedback for the creator. The verbal feedback is segmented and labeled with timestamps and mouse gestures. Later, when the video creator reviews the feedback, they can navigate and scrub the video by clicking the transcribed feedback item and reviewing their video criticizer's interaction while giving comments[32].

Similarly, Warner's work showed another example of mapping temporal orders of feedback and contexts. Their system enables the presenters to discuss the textual comments with their audiences in a practice presentation. Afterwards, the verbal discussion will be listed together with the textual comments. The presenter can review their discussions by using the list of comments with transcribed verbal discussions to scrub and navigate the replay of the discussion[36]. However, one limitation of this system is that the audiences need to create text comment points, view and interact with other's comments, and closely follow the presenter at the same time.

Previous work by Chen introduced a system supporting experiential training on counseling skills and shows an example of the solution to the above limitation. In this system, the two users role-play the therapist and client in turn. During the session, the user who plays the client can pin moments that are worth discussing with a button. After practicing, the users

can use the pins to review the moments and write discussion points[20].

While the previous systems show examples of facilitating temporal connections between feedback and contexts, Jung's work inspires the provision of spatial connections. In their study, they developed a system that automatically detects text items from a video presentation and links these items to the narration. By doing so, the system enables users to use slide items to navigate the presentation video and take notes[25]. A similar approach was used in Peng's previous study, proposing a system that enables users to navigate presentations and videos using slide elements[33].

Chapter 3

Formative study

3.1 Context

We conducted this formative study to analyze the standards of how technical interviews are conducted in the current job market, the difficulties current recruiters face when conducting these interviews, and how the professional interviewers would provide feedback to the job candidates in a scenario designed for practices rather than actual recruiting. Before conducting the studies, we submitted our study plan and had an internal review board for permission to recruit participants. These studies helped us gain design goals that motivate the system design. In our preliminary study, we observed and analyzed the interactions and feedback during 13 online mock technical interviews. Due to the interest of our study, we only observed mock interviews requiring interviewees to code while thinking aloud without the need to answer behavioral questions. For each session, we invited a professional software engineer and a student actively preparing for technical interviews. During each mock interview, we closely watched their interactions and asked the participants questions during the exit interview about the challenges they encountered when communicating and giving feedback.

From our observations, we derived (1) common behaviors and practices of practitioners in the software industry when they are observing the job candidate, providing help, and giving feedback during or after the interview, (2) design goals for a system that can help job seekers

prepare technical interviews through peer practices.

3.2 Participants

There are 13 mock interviews in our study, which includes 13 interviewers paired with 13 interviewees. For each pair, we created a separate information collection form to sign up for the study. The purpose of the form for interviewers, as shown in Appendix B.1, is to screen out fraudulent participants. To serve this purpose, we asked about their professional profile, including Github.com and Linkedin.com, and a brief description of their job roles. At the same time, we also asked for the topics and difficulty levels of the algorithm questions they intend to use to assess the participants and the programming language they are confident interviewing. These questions are related to the questions asked in the information collection form for the interviewees. In their forms, as shown in Appendix B.2, we asked about their confidence in preparation, languages they are familiar with, and the highest difficulty level of leetcode.com problems they are familiar with. Asking both groups similar questions about their technical backgrounds helped us match interviewers and interviewees in the mock interviews. For each session, we tried to have an interviewer who could understand the interviewee's preferred programming language and ask programming questions equal to or slightly higher than the interviewee's ability.

The interviewees were recruited from current Virginia Tech students in the CS department, mainly through the ads posted in the CS department email list. However, it is more difficult to collect participants playing the role of interviewers. Initially, we attempted to post ads to Virginia Tech Alumni Facebook groups, but only one valid interviewer signed up. At the same time, there are numerous fraudulent participants. To deal with this, we primarily collect our participants through our researcher's connections and Upwork.com after pointing out the

frauds. Upwork is a platform that connects freelancers with clients who need professionals for specific projects or jobs[12]. This platform helped us screen out fake participants as it requires the job applicants' professional accounts and brief application statements.

As for compensation, the interviewers get paid 50 to 75 dollars per hour, varying because of the contracts on Upwork.com. At the same time, the interviewees have no monetary compensation since interviewing with actual professionals usually involves paid services. In addition, they can get valuable feedback on their performances from the interviewers.

3.3 Pilot study

To design the process of our study, we first conducted a pilot study with a software engineer who is experienced in technical interviews as the interviewer. The interviewee in our pilot study is an undergraduate student preparing to seek a job. In this study, we let the interviewer lead the mock interview, using their experience conducting technical interviews in the industry setting to get a standard procedure for such interviews. The industrial standards will help us design the formative study environment.

For the pilot study, we planned to let the interviewer bring two programming problems from leetcode.com with difficulty levels according to the information collection form. Then, according to our original plan, the whole interview process will take 1 hour to 1.5 hours in total, with 30-40 minutes of technical interview, 20 minutes of feedback, and an exit interview with questions about the participants' experiences during the process. The exit interview is unstructured, with questions based on what we observed during the interview process.

Based on the result of the pilot study, we revised our actual study design as follows. In the actual study, we switched the exit interview to semi-structured, with several main questions

we are specifically interested in and incidents happening during the interview process. In addition, as we observed in the pilot study, time management is not optimal, as 30-40 minutes for two questions might be too short, especially for interviewers who want to expand an algorithm problem and address some follow-up questions. A brief explanation is also required to conduct this user study to let the interviewer and interviewee better understand what they are expected to do during the interview. Therefore, we modified the time management to 5 minutes for an explanation, 30 minutes for one or two questions up to the interviewer, 15 minutes for feedback, and 10 minutes for an exit interview. During the explanation phase, we introduced our project and addressed the importance of rich interview feedback to our study. In addition, we suggested rubrics for the interviewers to break down their feedback to the interviewee. They can freely add new criteria to the given ones or completely not use the given rubrics.

3.4 Study design

To acquire the insights we need for further analysis and development of the system, we conducted the user study with participants acting as interviewers and interviewees in a mock technical interview. During the one-hour mock interview session, we briefly introduced our study and had the interviewer ask one or two technical questions they had prepared before the mock interview. Then, we asked the interviewers to leave feedback as thoroughly as possible, including their impressions of the interviewee's performance and their possible improvements. During the feedback phase, the interviewer can take a reference on our given rubrics, including overall performance, algorithm-solving skills, code writing, and communication. As we define the tasks of the user study as the interview process and the feedback session, we define task completion as time elapsed or the communication between the two

participants ended naturally. Following the industry standards, each session is conducted in a shared code editor, where participants can write codes together. In addition to the code editor, we asked the interviewee to share their screen. In each mock interview, we recorded the session with the interviewee's screen sharing under both participants' consent. Lastly, after task completion, we conducted a semi-structured exit interview with the participants, asking about their experiences during the mock interview. The interview questions focused mainly on their overall experiences, difficulties of leaving feedback, usage of rubrics, and the strategies the interviewer used to take notes to recall their feedback toward the interviewee.

3.5 Analysis

All the online mock interview sessions were recorded with a webcam and screen sharing for easy analysis. We stored these recordings on Zoom directly with password protection under Virginia Tech-associated user accounts. Then, we transfer the recording to Sonix.ai[11] for AI-generated transcription. We analyzed the recordings of the sessions by reviewing the interview sessions and transcriptions. In the meantime, we coded significant findings from the feedback phase and the exit interview of the mock interviews. Once we got all the video recordings coded, we extracted the codes and conducted other open code sessions to organize them on Microsoft Whiteboard and extract meaningful design opportunities. Figure 3.1 shows the results of our open coding.

3.6 Results

We have observed several difficulties interviewers and interviewees face during and after the interview. The effects of these difficulties range from difficulties running the interview to

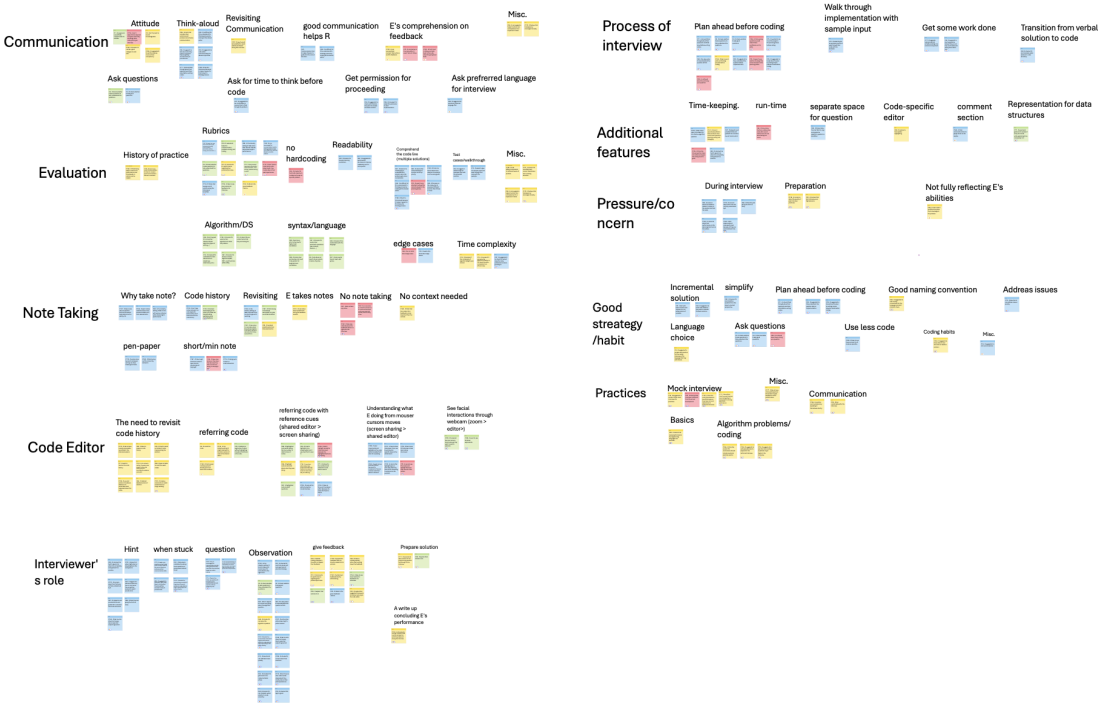

Figure 3.1: We analyzed the transcripts of the mock interview session by open coding to gain insights into the difficulties and common practices of practitioners conducting technical interviews. This figure shows the major findings after analyzing the codes.

communication failures and additional cognitive loads while giving feedback.

3.6.1 Cognitive load on taking notes

Participants indicate that they habitually take notes to remind themselves of what they observe in the interviewees. Some typed notes on their computers, while others took notes by hand on a notepad. These notes can serve as contexts and evidence for their later feedback session with the interviewee. Besides taking interview notes, interviewers mentioned that they regarded understanding the interviewee and helping them with their solutions as their top priority. Therefore, the design challenge here is to help interviewers save contexts for their feedback while not interfering with their priority task of understanding and giving hints.

This challenge might explain why some participants took unorganized quick notes because it is the fastest way to write down the observations. Despite the extra mental bandwidth needed for note-taking, some interviewers also mentioned that they are afraid to distract interviewees while typing notes or selecting codes with highlights.

3.6.2 Cognitive load on understanding the interviewee's code on the fly

According to our observations and interviews with the interviewers, there is a significant amount of cognitive effort interviewers spent to catch up with the interviewee. As the previous design insight mentioned, some interviewers would appreciate the interviewees always thinking aloud while working. In addition, some participants also mentioned that while watching interviewees coding, they pay extra attention to understanding the interviewee's approach to the problem firsthand. The challenge of comprehending interviewees' solutions can become even more complex when their solutions differ from the interviewers'. More than that, when the interviewees act stuck on a problem, the interviewers need to determine whether the interviewees are on the right track, why they are stuck, and what good questions or hints might lead the interviewees to progress. As the interviewers already have the job of evaluating the interviewees' performances, understanding their unique solutions and trying to help them when they go wrong can be too much work for the interviewers, which might ultimately sacrifice the quality of feedback.

3.6.3 Matching visible area in the code editor

In addition to the previous two points, we also noticed that the view-port interviewers' preferences varied during the interview. More specifically, some interviewers prefer to watch

the interviewee over the code editor, as they can directly type or highlight in the code editor. In contrast, others prefer screen sharing as the mouse pointer movements can give them a better sense of the interviewees' current progress. Although shared code editor has the advantage of direct manipulation, it is also noticeable that in some situations, the participants might lose track when their partner is looking out of their viewport. In our formative study environment, the collaborative code editor doesn't have an indicator or a following feature to help the participants match their visible areas. An example of a problem arising from the context of the discussion being outside the visible area of one of the participants is when the interviewer places the algorithm problem in the code editor at the beginning of the session. As the code grows longer, when the interviewee asks a question about the problem statement, they must scroll back and forth to find out where the interviewee is looking.

3.6.4 Design goals

Based on the above qualitative analysis results, we have formulated the following design goals for a technical interview system that supports peer practices.

- [**D1**] Minimize context switching for interviewers.
- [**D2**] Help interviewers give richer feedback.
- [**D3**] Allowing interviewers to make notes discretely without interrupting the interview process
- [**D4**] Help interviewers and interviewees follow each other easily

We summarized our observations on interviewers' multitasking requirements to understand the interviewee's solution, provide the necessary help, and take notes as evidence and remain-

der for the feedback session. As interviewers still want to keep engaged with the interviewee during the mock interview, we determined our design goal [D1] to provide supportive information and tools to the interviewers without the requirement to switch to external documents or platforms.

Among the multitasking requirements, interviewers need to take notes in the context of their feedback. However, they still regard understanding and communicating with the interviewee as their top priority. While our observation showed that revisiting older solutions would be helpful, we would like to build mechanisms to leverage contexts, like recording code progress, to help interviewers save context. This motivates our design goal [D2] to save more context for interviewers, making note-taking a less stressful task.

Regarding typing notes, our participants also expressed their fear of interrupting interviewees while typing them. Therefore, we determined our design goal [D3] to make sure interviewers have the full confidence to take notes if they need to without worrying about interrupting the interviewee.

Last but not least, we observed the trade-offs of interviewers using screen share and code editor to follow the interviewee. As a summary of the trade-offs, interviewers can directly manipulate the code to explain themselves better when they are using the shared code editor. However, using screen share provides them the ability to follow the viewport and mouse pointer of the interviewee. This allows interviewers to better follow the interviewee when they are not making changes or looking at different areas of the code document. This comparison motivates our design goal [D4] to enhance communication between interviewers and interviewees by allowing them to follow each other more easily by leveraging the advantages of screen sharing and shared code editor.

In conclusion, table 3.1 shows a brief summary of our findings and the design goals derived

from them. In this table, R refers to interviewers, and E refers to interviewees.

Table 3.1: The summary of our insights from formative study and derived design goals

Finding	**Summary**	**Design Goal**
R need to understand E on the fly R need to give hints R need to ask questions R need to take notes	Multi-tasking requirements of R during technical interviews	[**D1**] **Minimize context switching for interviewers** (by providing information supporting interviewers to run the interview easily without requiring them to switch to external applications or documents)
R prioritizes being present with E over taking notes when conflict happens R might be benefited from code history	Understanding the interviewee is the top priority for the interviewer, even though they need to take notes	[**D2**] **Help interviewers give richer feedback** (by automatically saving context to allow interviewers pay less attention on note-taking)
R is afraid to distract E	Interviewers don't want to interrupt interviewees	[**D3**]**Allowing interviewers to make notes discretely without interrupting the interview process**

R can directly make changes and highlight them in the shared editor. --- It is difficult for R to follow E through the shared editor when E is discussing texts out of R's visible area --- R can follow E's exact visible area through E's screen share --- R cannot make changes or highlights through E's screen share	There's a trade-off between using screen share and code editor for interviewers to observe and help the interviewee	[**D4**] **Matching visible area in the code editor** (with mechanisms leveraging the advantages of both screen sharing and shared code editor)

Chapter 4

System design

Based on the formative study, we developed a novel interview tool to enhance feedback in mock technical interviews. In such interviews, an interviewer needs to observe and assess the interviewee's ability from multiple perspectives, including communication, coding, and problem-solving. Meanwhile, they need to help the interviewees solve the algorithm problem by monitoring the interviewee's progress and evaluating their performance to determine whether they are in the right direction. By doing so, they can provide hints directly or encourage clarification questions when they observe their interviewee is going the wrong way. One challenge of this set of multitasking is that the interviewers need to understand the interviewee's solution while assessing their performance and taking notes simultaneously. As a solution, our system provides sample solutions and interview hints to assist interviewers in moderating the interview session. There are several platforms to conduct online technical interviews, though not many tools or platforms exist that aim to help job seekers practice. A difficulty when using current interview platforms is that, when taking notes, the interviewers need to either completely leave the shared editor and take quick notes outside or take notes with code copied from the shared editor [1][5][2][3]. The first way would be to make the interviewer detach from the interview process for a short amount of time, which can increase their cognitive load. The second way of taking notes is much better. However, interviewers might lose contexts of notes when the surrounding code around the copied code changes dramatically from the time the interviewers paste these codes in the notepad. In contrast,

our system supports interview session replays, including both webcam and code changes. When taking notes, each note can be connected with the timestamp automatically, as well as the interviewer's code selection. Therefore, later, when the interviewer provides feedback to the interviewee, they can watch their old discussions and code changes together when they want to explain a note made during the interview. In addition, there are transcripts that serve a similar purpose to the notes as timestamps. Together, these features provide a more straightforward process for the interviewers to conduct the interview and support interviewers in providing context-enriched feedback more easily.

4.1 Example User Scenario

Consider an example of two job seekers conducting a practice session for a coming technical interview. For a practice session, one of them would play the role of the interviewer, and the other would play the role of the interviewee. The interviewer can start the session by clicking the start interview button. At that point, the interviewer will see an interface, as shown in 4.1.

Afterward, the interviewer wants to introduce the interview question to the interviewee, so they use the "question" tab, which is a collaborative text editor on the right side of the viewport. The interviewer started breaking down the coding question for this session progressively by highlighting each paragraph of the question. However, when explaining some parts of the question, the interviewee asks about an ambiguity in some corner cases. So, the interviewer further clarified the interview question by adding more example input and output sets in the code editor. The interviewee then ensures their understanding is correct by giving another sample input/output set and asking if it is correct.

After fully explaining the interview question, the interviewer gave permission to the intervie-

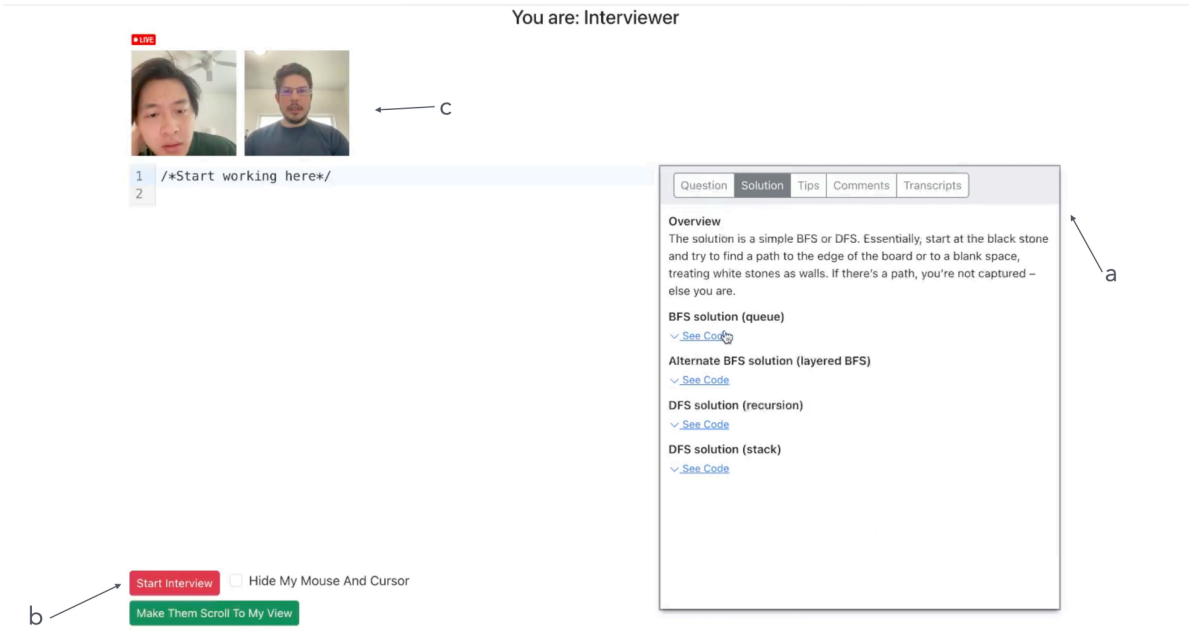

Figure 4.1: Interviewer's interface during the interview. (a)Tools to help the interviewer give feedback and conduct the technical interview. Unlike the interviewer, the interviewee will only see the question tab. (b)Buttons to start the interview, hide their mouse cursor, and scroll the interviewee's editor to match their visible area. Interviewees will not see these buttons. (c)Webcams of interviewer and interviewee.

wee to start the coding process. The interviewer and interviewee both see each other's mouse pointer and editor cursor. In addition, they will see a red indicator placed on top or bottom of the code editor when there's a mismatch in their code editor's visible lines. When the interviewer wants to explain some early code segment in specific and make new changes to it, the interviewee notices a red line on the top border of their code editor. So they scrolled up until the red line disappeared to watch the new changes made by the interviewer.

During the interview, when the interviewer is unclear whether the interviewee's solution is correct, even after the interviewee tries to explain it, they refer to the solution tab where all the solutions are listed. The interviewee asks the interviewer whether they are writing a loop to traverse all inputs correctly by highlighting the code segments. The interviewer approved this solution as it agreed with their hypothesis. They further explained it by using a mouse

pointer to circle another line of code and tell the interviewee that some part they might be missing in the input is already covered here. After explaining, the interviewer notices that the interviewee asked a valuable question and would like to take note of that so they can bring that up again in the feedback session. Therefore, with the concern of distracting the interviewee through highlighting codes, the interviewer hid their mouse cursor and editor cursors by clicking the switch, highlighted some code, and then took a note. They share this note immediately with the interviewee, though at this point during the interview, the interviewee is not able to see any notes.

After the interviewee finishes composing the solution to the interview question, the interviewer asks a few more follow-up questions suggested by the "hint" tab. The "hint tab" includes possible ambiguities, follow-up questions, and common pitfalls the interviewee could encounter.

Afterward, the interviewee clicked the end interview button to end the interview and start giving feedback to the interviewee. The interviewer starts by giving general impressions of the interviewee's performance and compliments their ability to ask clarification questions on whether their solution is on the right track. To further explain it by giving examples, They referred to a note and played the code changes and their discussions by the time the interviewee asked a clarification question. At this time, during the feedback session, the interviewee can see the notes shared with them as well. Later, when the interviewer wants to talk about the interviewee's good communication skills in trying to get approval about whether they are allowed to use the linked list library, they want to give an explanation that they can achieve the same purpose without using the library as well. But, the interviewer realized that they forgot to add a comment to this incident. Therefore, they searched the keyword "linked list" in the transcripts and found the code changes and discussion when they mentioned "linked list" as contexts of the feedback as the interviewee is making code

changes while mentioning the keyword. However, the time associated with the line of the transcript, including the "linked list," is slightly later than the moment, so the interviewer scrubs the timeline a bit more to the left to find the correct code history.

4.2 [D1] Minimize the context switch for interviewers

One of our design goals was to help interviewers alleviate the burden on interviewers to switch between different contexts in traditional systems. To achieve this, we have implemented a set of tabs alongside the collaborative code editor, which can be separated into two categories according to their functions.

The first set comprises questions, hints, and tips, as shown in figure 4.2, figure 4.3, and figure 4.4. These are the tools that the interviewer can use to moderate the interview session. The question tab is a shared text editor where both the interviewer and the interviewee can make editions. It allows the two users to dynamically discuss the interview question by adding more examples or making changes to it. By making highlights or adding new content, the interviewers can easily explain the interview question progressively. In addition, the interviewer can also look back to their discussion in the question tab as needed without having to disengage from the interview. The other two tabs, solution and tips, provide insights into potential solutions to the interview question, common mistakes the interviewee could produce, ambiguities of the question, and good follow-up questions for the interviewer to ask. Having these tools accessible directly beside the shared code editor, the interviewer can have immediate access to the information they need to guide the interviewee without having to navigate away from the interview.

The other category includes the private note-taker, as shown in figure 4.5. Similar to industrial defaults, the interviewer might want to take notes on the interviewee's performance

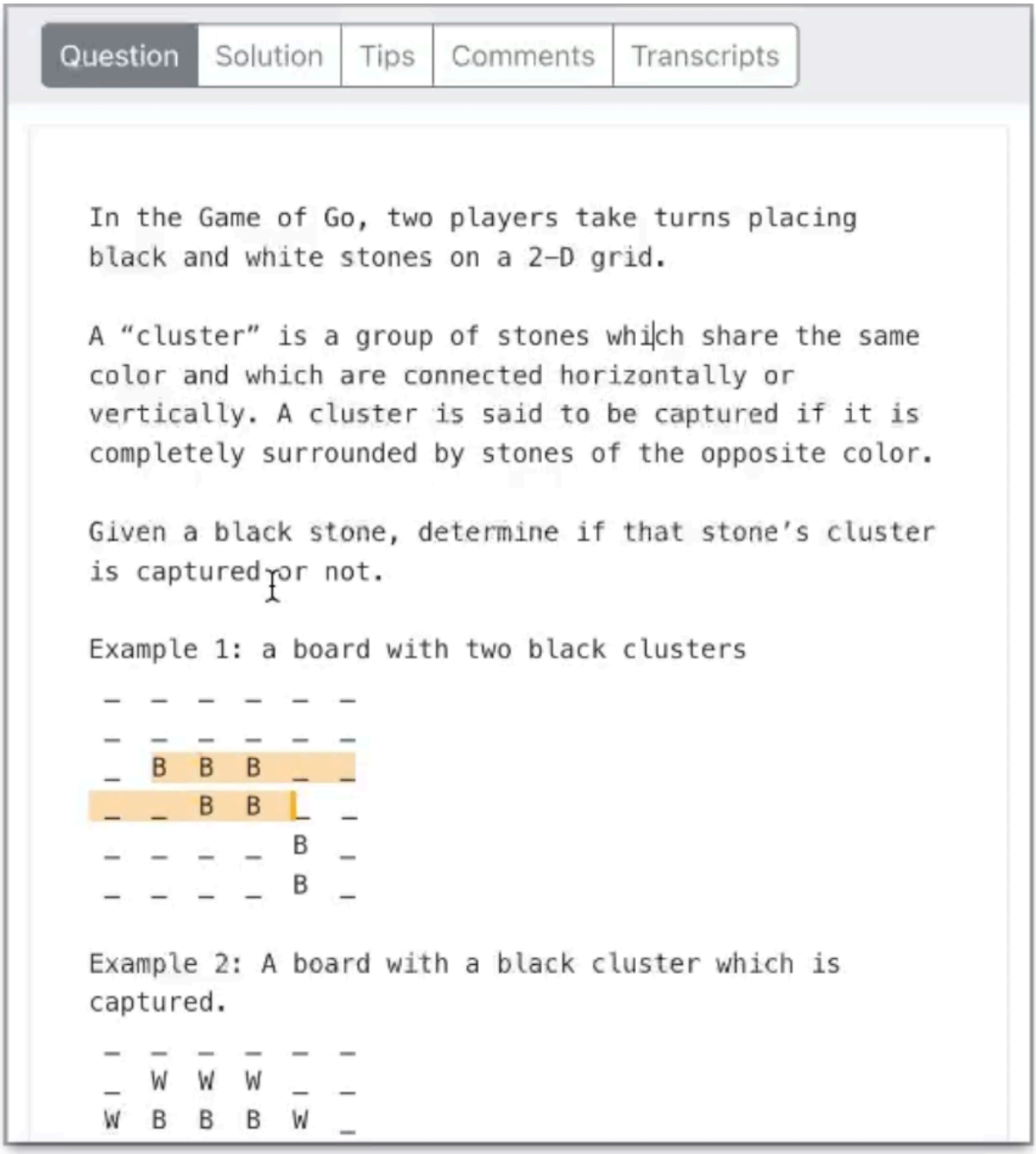

Figure 4.2: The question tab is a shared text editor. Interviewers and interviewees can discuss the interview questions by making changes or highlighting texts in the editor.

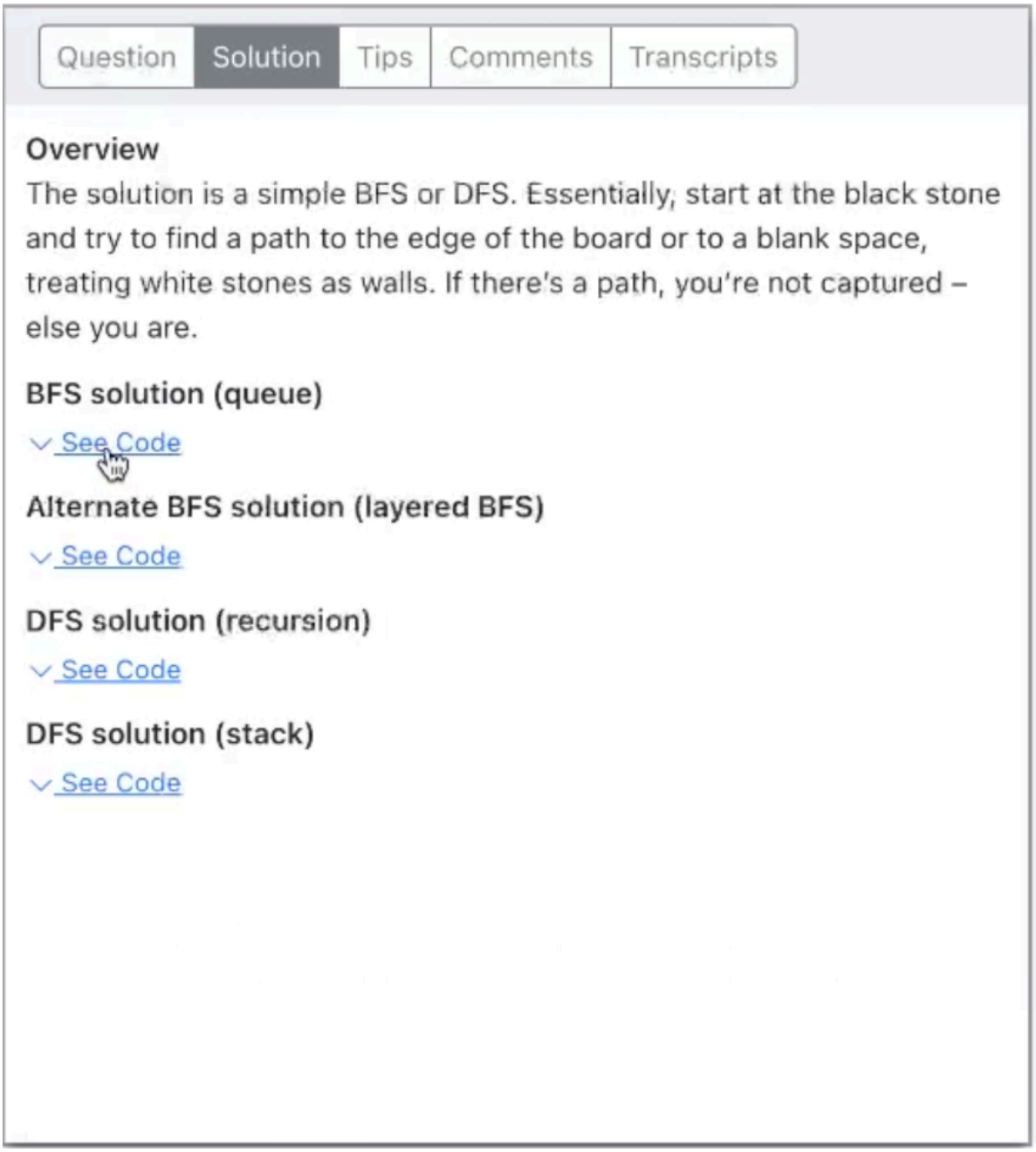

Figure 4.3: The solution tab shows possible solutions to the algorithm problem

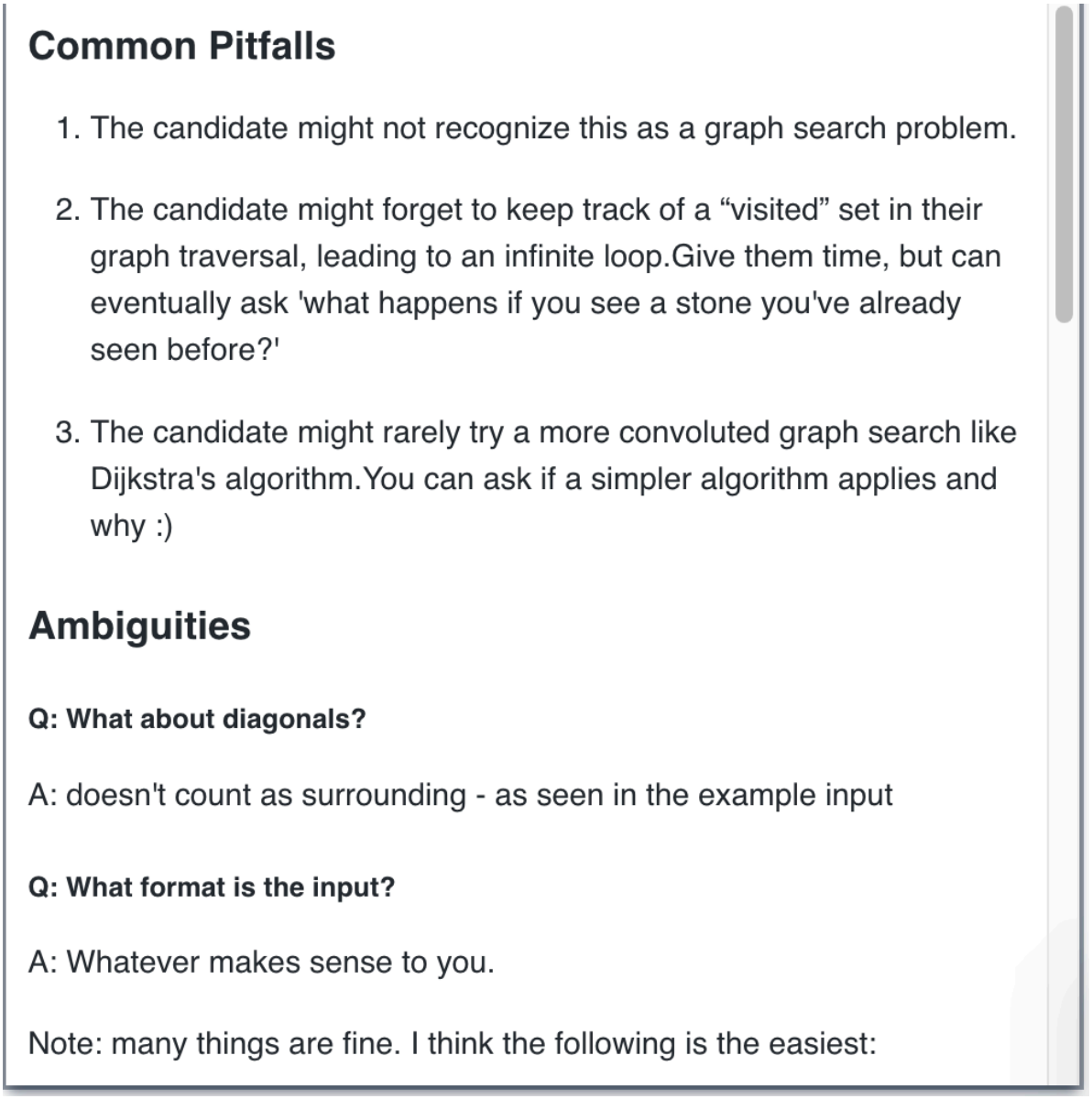

Figure 4.4: The tips tab includes common pitfalls, ambiguities, and follow-up questions serving as supportive materials to help interviewers conduct technical interviews.

during the interview. These notes can later be used as proofs and examples which will justify the interviewer's feedback and comments on the interviewee's performance. Our system provides the note tab directly beside the code editor. The interviewer can directly take notes while observing the interviewee solve the interview problem. This allows the interviewers to seamlessly record their comments and feedback in real-time without having to face the possible blank in observations from switching to other applications or physical documents.

4.3 [D2] Help interviewers give richer feedback

One of our principal design goals centers on helping interviewers deliver contextually rich feedback on the interviewee's performance. To achieve this, our system is designed to maintain contexts from both temporal and spatial perspectives, making sure the feedback is strongly connected to the code development process and interactions between interviewers and interviewees.

4.3.1 Temporal perspective

From the temporal perspective, our system supports the recording of both code changes and video calling. When both users join the session, and the interviewer starts the interview session, the recording will automatically start. With the records of code changes, interviewers can make comments reflecting the progression of the interviewee's approach to the problem. Despite this, preserving code history can unveil key insights into the interviewee's potential weaknesses or strengths, which may only be reflected in the early version of the code. For instance, the interviewee might make quick self-correction on logical errors in the initial brutal-force solution that was later replaced by a more advanced approach. In

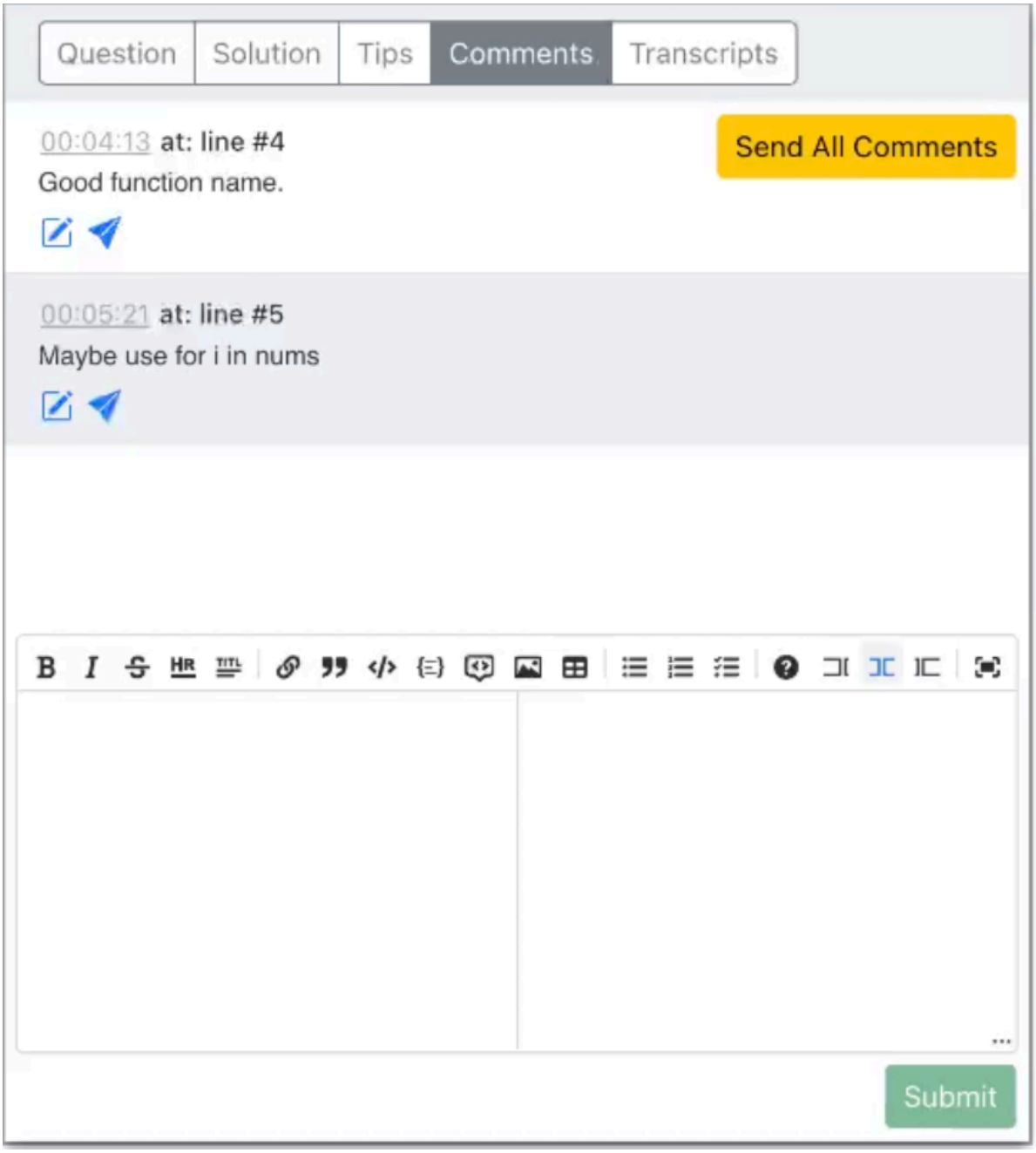

Figure 4.5: Interviewers can use the comments tab to take private notes while conducting the interview. Each note will be linked with a timestamp during the interview and the associated code the interviewer selected.

addition, the code changes will be recorded alongside corresponding timestamps, which will be synchronized with the video replay of the session. As an example, the interviewer could provide feedback on how quickly the interviewee corrects their solution by asking clarification questions and reflect the discussion in the code. Such synthesis of communication and code writing agrees with the basic nature of code interviews on assessing the job candidates' ability to work in a team setting. Through the integration of verbal discussion history and the code development process, the interviewer can provide examples of their feedback made from the perspective of collaboration and code writing.

4.3.2 Spatial Perspective

Building upon the fundamental functionality of preserving the video replays and code history, we also engineered our system to optimize the spatial context of the interviewer's feedback. In the case of industrial defaults, interviewers take notes on their observations as a document of justification for their evaluation of job candidates' performance. As an extension to this, we developed an enhanced note-taking functionality that allows the interviewers to link their observation notes directly to the selected code segments as needed during the interview process. Associating these notes with timestamps creates anchor points that the interviewers can reference during the feedback session. This enables the interviewers to directly navigate to critical moments in the replay to provide specific feedback in detail. The selected code segments linked to the notes are highlighted upon note replay to provide clear visual cues on the connection between the specific feedback and code. The interview notes associated with code segments shall serve as basic evidence of the interviewer's comments on the interviewee's proficiency. In addition, by automatically placing anchor points in the replay for key moments based on note-taking, the interviewer can give enriched feedback effortlessly with concrete examples. Besides, linking code segments with notes without including them in

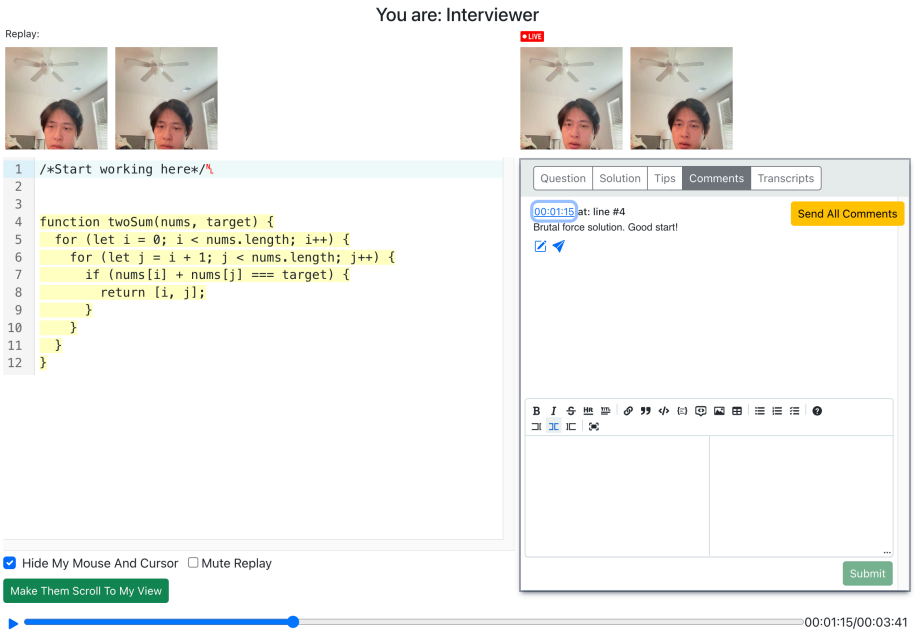

Figure 4.6: As shown, the associated code segment will be highlighted when the user replayed a comment, and the timeline be scrubbed to the corresponding timestamp

the note text body alleviates the burden of reading a massive load of texts when providing feedback. The preserved cognitive load can be used for the interviewer to better explain their comments and recall crucial moments and incidents during the interview.

4.3.3 Integrating spatial and temporal contexts

Figure 4.6 gives an example of the visual effects of replaying an interview note. By integrating these connections between feedback and contexts in the interview process, our system enhances the feedback's relevance and depth. Besides, providing tools for the interviewers to preserve code changes and discussion history can alleviate the workload of manually recording these backgrounds in their notes. This will allow them to make quick notes in real-time with enriched information, which will later be used as foundations for their feedback.

4.4 [D3] Allowing interviewers to make notes discretely without interrupting the interview process

As our collaborative code editor supports shared editor selection and cursor movements, which will be introduced in 4.6 in detail, the interviewers may disrupt the interviewee by making highlights and taking notes associated with code segments. Displaying the visual cues of the interviewer's focus within the code editor has the advantage of allowing the interviewees to follow the interviewer more easily, but it might increase the impression of being monitored and assessed while writing codes. This perception of being monitored might introduce unnecessary stress to the interviewee, potentially affecting their performance.

Despite giving the interviewees a false impression of being monitored, displaying visual cues whenever the interviewer makes a comment might give interviewees extra hints on their possible mistakes during the interview process. The interviewee might notice the mistakes in their solution by following the interviewer's selected code segments while making comments. Although interviewers have the job of leading and giving directions to interviewees toward an optimized implementation of the solution, giving early hints to the interviewees would distort the evaluation of their proficiency.

To mitigate these concerns about the visual cues for the two users to follow each other, we proposed a solution to allow the interviewers to control the visibility of their focus points within the code editor. This feature enables the interviewer to take notes associated with code segments without unintentionally providing leading information to the interviewee. At the same time, the interviewee can benefit from feedback on their true abilities from a genuine interview process without the extra mental loads.

4.5 [D4] Help interviewers and interviewees follow each other easily

Existing shared editors usually support visual cues of the remote editor cursor and text selection. Our system, serving the basic purpose of a collaborative code editor, covers these basic features. In addition, we also implemented visual cues for the mouse pointers. During the mock interview, the interviewer and interviewee might initiate discussion on certain code segments by making mouse gestures naturally, for example, circling their mouse pointers around the code segment and verbally indicating the line number of this segment. We propose three new features for the interviewers and interviewees to follow each other with less effort.

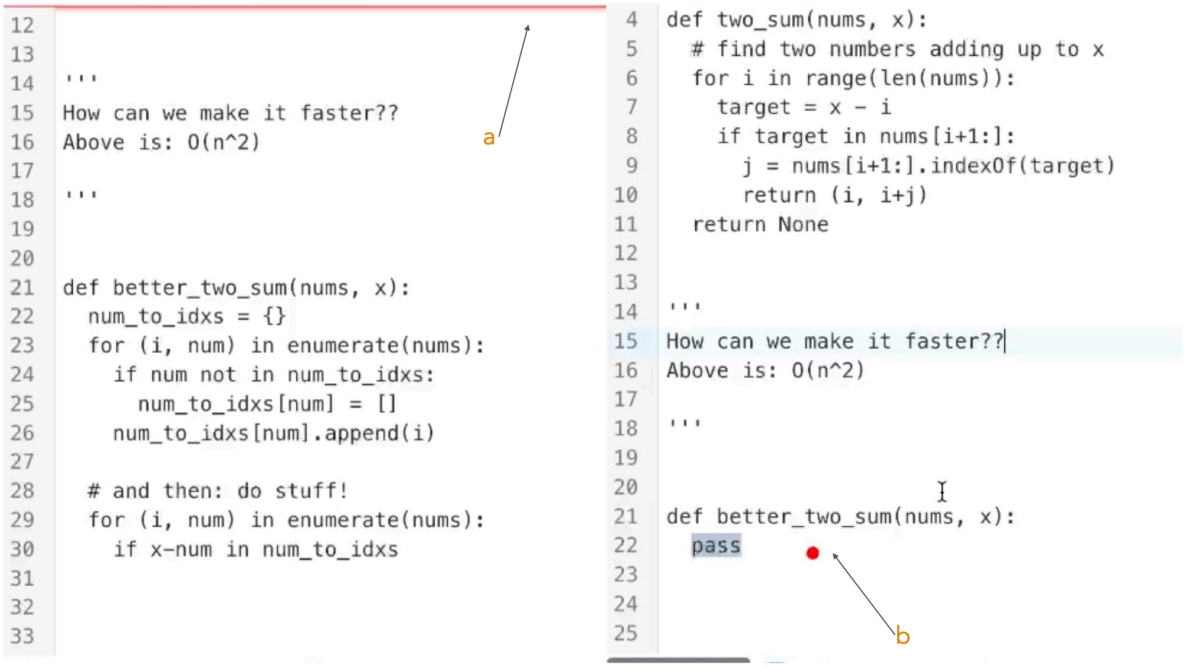

Figure 4.7: Our system provides visual cues for peer mouse and viewport to help users follow their peer's focus more easily. (a)Indicator on the top edge of the editor suggests users scroll up to match viewport with their peer. (b)Projection of peer's mouse pointer.

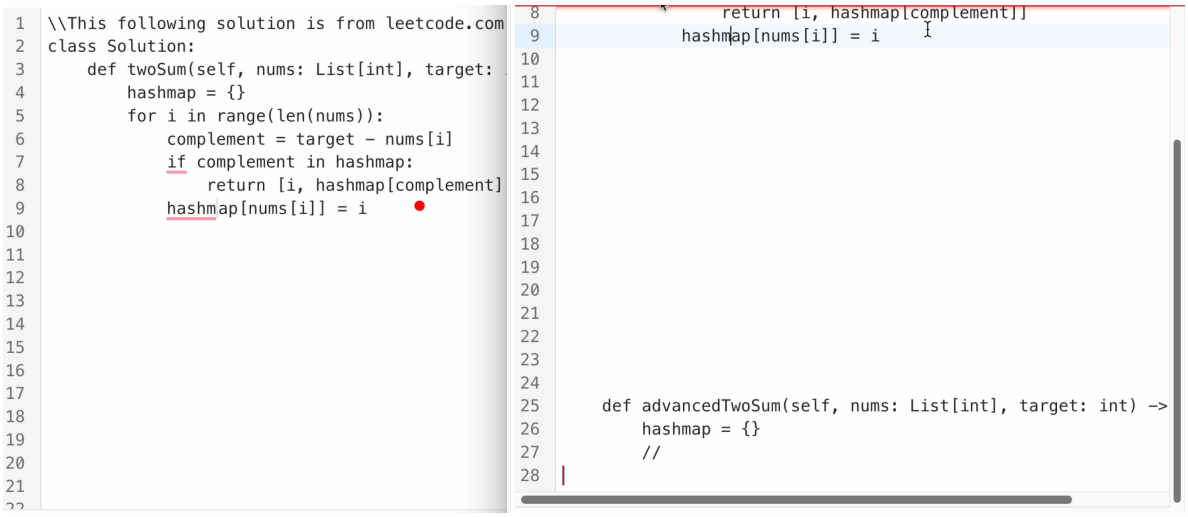

Figure 4.8: As shown, although the ranges of visible lines are different, the remote pointer in the left editor still matches the mouse pointer in the right editor.

4.5.1 Display of mouse pointer

In addition to the visualization of the remote cursor and selection in the code editor, our system provides a projection of the remote mouse pointer. When the other user's mouse moves inside the code editor, the user will be notified by seeing a red dot representing the remote mouse. One difficulty of this is the mismatch of the visible lines from the two users, which might affect the accuracy of the remote mouse pointer. To overcome this, we implemented the remote mouse pointer with the awareness of the visible lines. To be more specific, the remote mouse will only be visible when it is within the user's visible range. In addition, the remote mouse will be displayed with the awareness of the scroll position of the code editor. As a result, its position will be accurate even if the interviewer and interviewee have different visible ranges due to scrolls in their editors, as shown in fig 4.8.

4.5.2 Visual awareness of view-port

Another feature we have is visual cues for view-port disagreements. This feature was designed as an indicator notifying the user to scroll their code editor to match the other user's viewport. With the indicator, the interviewer and interviewee can match the context of their discussion without the extra effort of verbally agreeing on each other's visible lines. To achieve this purpose, we designed the indicator as a red line placed on the top border or the bottom border of the code editor. The placement of the red line indicates the direction the user should scroll to match the other user's visible lines. There are two exceptions to the visibility of the scroll indicators. The first is the situation when the remote mouse pointer is inside the range of the user's visible lines. The other exception is when the interviewers chose to hide their focus point in the code editor, as discussed in 4.3.

4.5.3 Synchronization of view-port

Similar to the view-port mismatch indicator, the interviewer can scroll the interviewee's viewport to match their visible lines. This feature is mostly used to facilitate the interviewer's explanation of specific examples or lines of code.

Combining these three features, the interviewer and interviewee can seamlessly follow each other without verbally matching their visible lines and points of interest. We did not develop the functionality to strictly match the visible lines by making one user follow the other's scroll position. The reason behind this is the potential conflicts on giving which user the control of scroll. For example, when the interviewer and interviewee both select to follow each other's position, who should control the scroll position of the two of them is unclear. Therefore, we proposed a mismatch indicator to let them scroll to match their visible lines and let the interviewer have control over changing the interviewee's visible area as needed.

4.6 Implementation details

4.6.1 Code editor

The code editor of our system was built on the open-source project Codemirror[4]. It is a code editor component for web applications with expandable features like widget decoration and collaborative editing. While it provides the basic features of a code editor, we implemented shared cursor movements, code history, and collaborative code changes as different extensions following their official examples. In addition to the code editor, as discussed in 4.2, the interview question is displayed in a collaborative text editor. This separate text editor was developed based on the collaborative text editing package from Liveblock[7].

4.6.2 Video calling and peer connection

The video calling feature was developed based on the WebRTC API[13]. It enables web applications to communicate directly with other web applications running on different browsers. With WebRTC, we can make real-time video calls with users directly streaming their webcam captures to each other without the extra network loads to send data to a central server. The matching of users is achieved by a central socket server developed based on the socket.io project[10]. Once matched, both the video stream and user actions, like the start and end of the interview, are passed as "signaling data" through the WebRTC peer data channel developed based on [9]. In addition, the code replay and video replay are developed separately. We developed the code recorder as an extension of the Codemirror editor while using the MediaStream Recording API[8]. During an interview session, each user will record their own webcam and the peer's webcam and save the recorded videos as text blobs, which later can be converted back into videos as needed.

Chapter 5

Evaluation Study

5.1 Study Procedure

To understand how effective the key element of our design was in reducing interviewers' cognitive load when conducting technical interviews and giving feedback, we conducted a qualitative study with four pairs of interviewer and interviewee. The interviewees were recruited from Virginia Tech students who have signed up for our formative study. The interviewers were recruited from the professional engineers who participated in our formative study, along with one new hire from Upwork.com. Similarly to the formative study discussed in Chapter 3, this study is designed as mock technical interviews with software engineering practitioners as interviewers and job seekers as interviewees. We recruited the interviewees from the same pool of participants for the formative study. At the same time, we invited interviewers who participated in our formative study to participate in the evaluation study again. All participants playing the role of interviewers have experience interviewing job candidates in the tech industry. Similarly, all interviewees in our study are active job seekers or students who majored in computer science and have at least a basic understanding of the procedure of technical interviews. The study setting is similar to the formative study. However, we extended the mock interview session to one and a half hours, allowing us to explain our system to the participants. Unlike the formative study, we asked the interviewers to run the mock interview with pre-defined questions to test the usability of the solution and tips

features. After the mock interview, the interviewers shall give feedback to the interviewees, mimicking a regular practice interview session. We interviewed both participants about their experiences using the system immediately after each mock interview. These post-interviews are designed as semi-structured interviews under the guideline in Appendix A.1, where we ask questions based on whether they actively used the special features of our system or not.

5.2 Results

This section will include our findings after running the evaluation study. In our examples and discussions about our observations, we will code interviewers as R and interviewees as E. Table 5.1 summarized user experiences related to the features we created. This chapter described these findings and provided more user impressions regarding the general experience of our system and missing features that users might want.

5.2.1 Interviewers' cognitive load is still high

From our observations, it is noticeable that, although not found in previous studies, interviewers experience a heavy cognitive load when conducting technical interviews. In this example, R1 was unable to actively use the solution panel or comments panel during the interview because it required too much mental load in addition to being able to present in the mock interview as an evaluator and help provider at the same time:

> "It's good to have the solutions here. But I don't think I would ever use the solution during the live interview. It just takes too much mental bandwidth to actually absorb the solution... you have to already know a solution to interview effectively, or you just do the same journey as the

> interviewee. I think it's good to have the solutions there, but I don't think I would ever use it" - R1

Unlike the fear of the burden of reading long solution code, R1 complimented the interview tips we provided alongside the code editor:

> "It's always helpful to have a cheat sheet including things like what follow-up questions I would like to ask. These are easier for quickly scanning without derailing the interview" - R1

Unlike R1, R3 raised concerns about the interview tips by stating the texts in this tab is way too much to read while conducting the interview. To them they

> "It's definitely helpful content, but I'm not sure if there's a better way to search through the context or a simplified presentation. But it's a little hard to flip back and force and scan through a lot of texts to find the answer" - R3

The same concern occurred with the experience of taking notes. For example, R1 stated they could not take notes because they needed to be "hyper-present" in the interview with the interviewee. As a result, they didn't have time and energy to detach from communicating with the interviewee and take notes for the feedback. To further explain this, they mentioned that if they were doing a one-to-two interview in which two interviewees collaborate to solve the algorithm question, they might have more free time to take notes. Similarly, R3 stated that it was a fair amount of multitasking to switch back and forth in the interview tools when they needed to take notes. However, R3 commented that the extra task of taking notes in our system was not stressful for them, and it has the advantage of saving the contexts that would have been lost without the note features:

> "I'm multitasking more than I am during a normal technical interview.

> But, I also have more confidence because my thoughts are accurately captured even without the recording. Having notes with specific lines and specific moments is really helpful" - R3

The phenomenon that the interviewers did not engage extensively with our system might also be related to their professional experience. As they stated, they are deeply acquainted with the current interview procedures in the industry. Therefore, they did not find it intuitive to try the replay functionality, as it is uncommon in industry settings.

5.2.2 Peer awareness features

During the interview, our system provides a collaborative code editor and text editor with indicators of peer presence. Our goal behind this design is to help users follow each other's focus seamlessly, reducing the need for verbal confirmation and minimizing disruptions to communication and the thought process. As a result, participants in our user study have reported a generally positive experience after using our system with the design to enhance peer awareness.

For instance, one of our designs to display peer presence in the code editor is to add red lines on the top or bottom edge of the code editor when there are mismatches in their visible lines in the code editor. At the same time, there will be a visual representation of the mouse pointer when the peer is pointing in the area within the visible lines. Besides the code editor, the shared text editor for questions will have basic collaborative editor features like shared editor cursors and highlights. Reflecting on the utility of these visual cues, interviewer R1 commented:

> "If I was explaining something, I have some sense that whether they were looking at the things I was trying to get them to look at…Being able to

> highlight in the demo or in the question was a lot easier to (make sure we are) looking at the same thing or pointing at the same thing. So that was definitely faster for communication." - R1

Later, R1 also suggested that the ability to replay an interview session with comments connected with timestamps can be helpful if our system is used in real technical interviews. However, following the positive comment, we also sought to understand the potential drawbacks of this design. In this context, interviewee E1, participating in the same session as R1, confirmed that the visual cues for R1's mouse and viewport did not disrupt their coding process because they are subtle enough to be ignored when the interviewee is focused. E3 shared the same experience with E1 in that these visual cues did not distract them because they did not notice them when they were focused.

Similarly, R3 and E3 reported positive experiences regarding our design of peer presence indicators. However, R3 reported that these visual cues can fail if the interviewee is looking at the interview tips or making comments while the interviewee is highlighting texts in the question tab.

Unlike other participants, E4 reported feeling distracted by R4's mouse pointer visual cues while writing code. Along with the complaint of distraction, R4 mentioned that they wish to have their mouse cursor hidden on the interviewee's screen all the time and shown as needed.

When considering how to enable the users of our system to follow their partner's area of focus easily, we referenced existing systems on the market. One of the existing solutions is to allow users to follow the other user strictly. Commonly, systems utilizing this feature have a toggle button for the users to sync the viewport with peers. When asked to compare the user experience of our design and the existing solution of syncing the viewport, both R1 and

E1 stated that they prefer our system, as they both agreed that they would like to retain the ability to fully control the viewport. For example, R1 stated they want:

> "more controls toward what they are looking at rather than being snapped to the other person's screen, so visual indicators is good enough." And "scroll to prepare to ask another follow-up question. " - R1

Despite the visual cues, our system also enables the interviewers to align the interviewees' viewport with their own with a dedicated button. However, interviewers did not use this feature because the code file was not long enough. As R1 suggested, in scenarios with longer codes, where visual cues of visible line mismatch might be less effective, this button design could become more helpful for matching areas of focus.

5.2.3 Comments might be useful but distracting

Although we designed a comment panel alongside the shared code editor for interviewers to leave comments and notes as contexts of feedback, R1 did not use this feature at all during the interview. Despite the reason for the cognitive load of multitasking on communicating with the interviewee and doing other things simultaneously, R1 further explained their concerns stopping them from using the comment feature in two perspectives. Firstly, placing the comment panel alongside the shared code editor can obscure the boundary between the shared and non-shared areas. As R1 stated, they did not use this feature because:

> "I think the comment scares me because knowing what's shared and what's not shared is really tricky for me. I would hate to fat finger and get confused on what's public and private." - R1

In addition to the interface design not showing a clear separation between shared and private contents, R1 did not make many comments using the system because they were afraid to

interrupt the interviewee while making comments:

> "I have a mechanical keyboard. I did not want to distract the interviewee by clicking on my keyboard." - R1

5.2.4 Asynchronous feedback

Although our design aimed for real-time feedback right after the mock interview, the user study results still highlighted the need for delayed feedback after the mock interview. As our interviewee E3 stated, they would appreciate both real-time and asynchronous feedback equally:

> "Getting the feedback immediately helps calm the nerves because you can get to know exactly what you did wrong and what you did well. But also, I think the feedback would be much better if the interviewers could calm themselves too and think about it… Also, for me at least, I also can have a typed version of feedback so I can keep it and look them up before other interviews." - E3

R3 agreed with E3 after this, stating their preferences on giving feedback:

> "I generally prefer immediate feedback just to give the interviewee a general sense of how they did during the interview. But I think there are cases where it is helpful to have a follow-up (later) which is more thoughtful and actionable." - R3

Unlike R3, R1 preferred non-real-time feedback over providing feedback right away because they could have more time to think about what happened during the interview and give more accurate and actionable feedback. In addition, they could have time to thoroughly

think about which observation is worth discussing to help interviewees improve.

Our system allows users to replay code changes and video chat during the feedback, but these features were not actively used in the user studies. R1 and R3 both agreed that replay would be much more helpful if they were to review interviewees internally after the mock interviews. To be specific, R3 stated that having the ability to tie comments with specific code segments and time stamps can be a huge advantage for reviewing job candidates in real technical interviews. Having the notes along with the replay also has a possible benefit when used in internal team discussions and real technical interviews.

5.2.5 Seperate text editor for question reduced context switching

Our system allows users to see interview questions alongside the code editor in order to reduce the need to switch contexts between codes and other documents. Participate R3 complimented this design based on two reasons. Firstly, compared to displaying questions in a static format, it's much easier for them to explain the question and answer questions by making new examples dynamically. Secondly, compared to displaying questions as part of the code, normally as comments at the top of the code, using a separate question editor is better because it stays on the screen and is not at risk of being accidentally deleted or replaced. R4 shared the same experience with R3, indicating:

> "I prefer prompt separate from the code... I can scroll to the middle of the prompt to see one case that I'm working on and still have the code... versus going back and forth through the document and losing time and focus. " - R4

5.2.6 Need for runtime environment

When users use our system, they will write the solution to the algorithm question on a shared code editor without a compiler. As we asked participants what some immediate improvements to our system are, several of them mentioned the need for a runtime environment. According to R2 and R3, having a runtime environment makes the interview process closer to a real programming environment. At the same time, R3 and E2 also noted the importance of a runtime environment to test the logic and syntax of code.

Table 5.1: The summary of our insights from evaluation study

Replay	Pros	Comments tied with timestamps and code segments have the potential to help write internal reports
	Cons	Replay is not intuitive enough for them to use
		Interviewers found the potential of the system in writing reports rather than real-time feedback.
Notes	Pros	Being able to associate notes with code and timestamps was useful.
	Cons	The typing sound captured by the audio input stopped them from typing notes because they didn't want to interrupt the interviewees.
		The interviewers were worried that notes, although private, would be immediately visible to interviewees and distracting to them.
		Interviewers did not take notes because they needed to be active with the interviewee.

Cursor and mouse visualization	Pros	Generally, positive experiences were reported by participants.
		It made them more confident in communicating.
		One interviewer specifically stated that they would like to hide their mouse cursor all the time.
	Cons	One interviewee felt distracted seeing the visual cues while coding.
Question Info Tabs	Pros	Interviewers think having a question tab separate from the code reduces the need to switch contexts.
		The editable question tab made it easier to explain the question.
	Cons	Interviewers think solutions, tips, and hints are useful information but too long to read during the interview, rather useful for pre-interview prep.

Chapter 6

Discussion

6.1 Real-time feedback is new to interviewers

We implemented code replay and video replay tied with interview notes to enable interviewers to spend a less cognitive load on taking notes and saving contexts so they can pay more attention to communicating with interviewees and understanding the code, which was regarded as a priority job for interviewers. During our evaluation study, we noticed that the interviewers were not actively using the replay feature of our system. As a result, our design to enable interview replay to reduce the mental bandwidth for interviewers remains untested.

However, as claimed by interviewer participants, they are too familiar with the current interview feedback to provide feedback. In industrial settings, they would report to their boards asynchronously after a technical interview rather than giving real-time feedback. These observations align with our literature reviews, which show that feedback is commonly absent in the industrial setting. As a result, the interviewers did not utilize our interview notes panel and replay features actively as we expected. Therefore, the evaluation of our designs to replay and link comments with code and timestamps has not been tested intensively. Thus, whether these features are useful in mock interviews to enhance feedback is unknown.

As a solution, future studies can benefit from long-term usage of the system. In doing this,

interviewer participants can become familiar with providing real-time feedback right after a mock interview. By removing the potential bias in the evaluation caused by interviewers' unfamiliarity, we can better understand whether our design for replays and note panels can reduce interviewers' cognitive loads.

6.2 How to reduce the cognitive load for interviewers

To reduce the multitasking need for interviewers to conduct technical interviews, we designed a toolbox placed alongside the code editor for interviewers. The interviewers can get access to supportive tools and information without the need to switch to different platforms or documents. These utilities can be separated into two categories: one is the comment panel serving as private interview notes, and the other includes solutions, tips, and questions as a separate text editor.

Unlike our design goals to reduce cognitive loads and minimize context switching, the supportive information received mixed feedback from the interviewers. Generally, interviewers regarded the solution and tips we provided as useful information. However, interviewers also claimed that this information is too tedious to read while being active in the mock interview. At the same time, interviewers appreciated our design to place the interview question in a separate editor as it allows them to break down the question dynamically by making changes to it or point out specific areas by highlighting them. Although these can also be achieved by pasting the question to the shared code editor, interviewers wouldn't need to scroll back and forth to review the question while their interviewee is writing code. However, combined with the comment section, these utilities raise new requirements for context switching since interviewers still need to switch between questions and comments. As reported by interviewers, they might lose the context of discussion when they are taking notes in the comments panel

while the interviewee is asking a question about specific examples in the question editor.

In addition to the supportive tools and information, our system also allows users to replay the video calling and code changes to provide contexts for their feedback. However, as discussed in the previous section, the interviewers were not actively using replay. Therefore, when testing the effectiveness of our system in future studies, especially for peer-to-peer feedback exchange, we would allow interviewers to spend some time on their own before providing feedback. For both interviewers and interviewees, asynchronous feedback is as beneficial as real-time feedback since it allows interviewers more time to consider carefully before making suggestions to interviewees. Therefore, without the requirement of providing feedback right after the mock interview, interviewers can rely more on replay to recall details of their assessments. This can benefit our future studies in two ways. Firstly, interviewers can use replays to revisit their communications with interviewees to consider their assessments. Therefore, they can spend less energy taking notes to save context. Secondly, when interviewers use replays more, we can better assess the effectiveness of our design to support replays so that interviewers can save contexts automatically.

6.3 Limitations of evaluation due to the pool of participants

Although we conducted mock interviews to test the effectiveness of our designs, several difficulties can limit the effectiveness of our evaluation.

Firstly, we only conducted four sessions with eight participants in the evaluation study. The main reason behind the limited number of participants was the difficulty of recruiting professional engineers as interviewers and coordinating mock interview sessions around the

busy schedules of professional software engineers and students. Likewise, similar difficulties happened in formative studies as well. The incidences of fraudulent participants recruited from a broader pool of participants made us rely solely on participants from our personal connections and a freelancing platform, Upwork.com, which supported us in screening participants by their professional profiles and past experience on the platform. However, the pool of candidates is still limited on freelancing platforms as there are workers who are working full-time freelancing rather than actively in technical companies.

Secondly, we only relied on qualitative data for both formative and evaluation studies. This issue, along with the limited number of participants, affected the generalizability of our findings. For a more robust evaluation of the effectiveness of our design, a comparison between interviewers' experiences with and without our system would be beneficial.

Thirdly, although we wanted to build a platform to enhance feedback exchange between job seekers, we only tested the system with professional engineers. As job seekers are less experienced with the technical interview procedures, issues like multitasking, especially understanding the interviewee in real time, could be more crucial when we have job seekers playing the role of interviewers. In addition to the existing difficulties of professional interviewers, job seekers might have other unknown difficulties we overlooked when building this system. Therefore, the usability of our system under the setting of peer interview practices remains unknown.

As a solution to these problems, future studies can benefit from running more evaluations with job seekers conducting technical interviews and playing the role of interviewers in turn. By having more participants, we can get more justified assessments of our system's usability. At the same time, with the observations from mock interviews with job seekers using our system, we can get insights into our system's generalizability and areas for improvement, especially in supporting peer practices.

Chapter 7

Summary

In our study, we collected insights into the difficulties and standard practices of professional programmers conducting technical interviews. According to our observations, the interviewees experience high cognitive load when running technical interviews as they need to multitask between observation and evaluation, communicate with the interviewee, and understand the interviewee's code. In addition, there is an extra need for interviewers and interviewees to verbally confirm the lines of codes they would like to mention during discussions. We used these insights to create design goals and developed the system to facilitate interviewers' providing context-enriched feedback and running technical interviews more efficiently. Our system has the essential functionalities required for technical interviews, including video chat and shared code editor. To reduce the need for verbally syncing visible lines in the code editor, we designed our system to display visual cues of the peer presence in the code editor.

Meanwhile, we developed the functionality to replay code changes and verbal communication over the video chat during the interview. Interviewers can take private interview notes automatically tied with timestamps and manually linked with code segments. When giving feedback, they can use the notes to navigate the replay to provide contexts for their feedback. Including the private note panel, five tools, including tips, solutions, transcripts, and a separate shared text editor for the interview question, are placed alongside the code editor to reduce the need for interviewers to switch contexts between different documents.

After running evaluation studies to test the usability of our system, we noticed that our

approach to enhancing peer awareness and displaying interview questions in a separate text editor provided participants with a positive user experience. Interviewers did not actively use the interview replay features during the user study because of the cognitive load and their unfamiliarity with giving feedback. [24]

However, interviewers expressed interest in using the interview replay with associated notes when they needed to review the interview session rather than giving real-time feedback. Studies could be conducted to further test the effectiveness of the design for notes and replay and how effective our system was in giving asynchronous feedback. In addition, to further reduce the cognitive load of interviewers, future development could leverage LLM models to generate automatic notes, assess the correctness of interviewees' code, and evaluate the interviewees' communication skills.

Bibliography

[1] CodeInterview. https://codeinterview.io/, . Accessed: 2024-04-10.

[2] CodeShare. https://codeshare.io/, . Accessed: 2024-04-10.

[3] CodeSubmit. https://codesubmit.io/, . Accessed: 2024-04-10.

[4] CodeMirror. https://codemirror.net/, . Accessed: 2024-04-10.

[5] CoderPad. https://coderpad.io/, . Accessed: 2024-04-10.

[6] interviewing. https://interviewing.io/. Accessed: 2024-04-20.

[7] LiveBlock. https://liveblocks.io/. Accessed: 2024-04-10.

[8] Recorder. https://developer.mozilla.org/en-US/docs/Web/API/MediaRecorder. Accessed: 2024-04-10.

[9] SimplePeer. https://www.npmjs.com/package/simple-peer. Accessed: 2024-04-10.

[10] Socket.IO. https://socket.io/. Accessed: 2024-04-10.

[11] Sonix.Ai. https://sonix.ai/. Accessed: 2024-04-10.

[12] Upwork. https://www.upwork.com/. Accessed: 2024-04-10.

[13] WebRTC. https://webrtc.org/. Accessed: 2024-04-10.

[14] hellointerview. https://hellointerview.com/. Accessed: 2024-04-20.

[15] Faheem Ahmed, Luiz Fernando Capretz, and Piers Campbell. Evaluating the demand for soft skills in software development. *It Professional*, 14(1):44–49, 2012.

[16] Adnan Aziz, Tsung-Hsien Lee, and Amit Prakash. *Elements of Programming Interviews in Java: The Insider's Guide*. EPI, 2012.

[17] Mahnaz Behroozi, Chris Parnin, and Titus Barik. Hiring is broken: What do developers say about technical interviews? In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9. IEEE, 2019.

[18] Mahnaz Behroozi, Shivani Shirolkar, Titus Barik, and Chris Parnin. Debugging hiring: What went right and what went wrong in the technical interview process. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*, pages 71–80, 2020.

[19] Mahnaz Behroozi, Shivani Shirolkar, Titus Barik, and Chris Parnin. Does stress impact technical interview performance? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 481–492, 2020.

[20] Tianying Chen, Michael Xieyang Liu, Emily Ding, Emma O'Neil, Mansi Agarwal, Robert E Kraut, and Laura Dabbish. Facilitating counselor reflective learning with a real-time annotation tool. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–17, 2023.

[21] Denae Ford, Titus Barik, Leslie Rand-Pickett, and Chris Parnin. The tech-talk balance: what technical interviewers expect from technical candidates. In *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 43–48. IEEE, 2017.

[22] William Gant and William Gant. Why software development interviews are hard. *Surviving the Whiteboard Interview: A Developer's Guide to Using Soft Skills to Get Hired*, pages 1–6, 2019.

[23] S Heathfield. Why employers don't give feedback to rejected candidates, 2019.

[24] Jeremiah Honer, Chris W Wright, and Chris J Sablynski. Puzzle interviews: What are they and what do they measure? *Applied HRM Research*, 11(2):79, 2007.

[25] Hyeungshik Jung, Hijung Valentina Shin, and Juho Kim. Dynamicslide: Reference-based interaction techniques for slide-based lecture videos. In *Adjunct Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, pages 23–25, 2018.

[26] Amanpreet Kapoor, Sajani Panchal, and Christina Gardner-McCune. Implementation and evaluation of technical interview preparation activities in a data structures and algorithms course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 882–888, 2023.

[27] Susan Klitzman, James S House, Barbara A Israel, and Richard P Mero. Work stress, nonwork stress, and health. *Journal of behavioral medicine*, 13(3):221–243, 1990.

[28] Gayle Laakmann. Cracking the technical interview. 2009.

[29] A Lerner. You can't fix diversity in tech without fixing the technical interview, 2016.

[30] Stephanie Lunn, Monique Ross, Zahra Hazari, Mark Allen Weiss, Michael Georgiopoulos, and Kenneth Christensen. The impact of technical interviews, and other professional and cultural experiences on students' computing identity. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, pages 415–421, 2021.

[31] Gerardo Matturro, Florencia Raschetti, and Carina Fontán. A systematic mapping study on soft skills in software engineering. *J. Univers. Comput. Sci.*, 25(1):16–41, 2019.

[32] Amy Pavel, Dan B Goldman, Björn Hartmann, and Maneesh Agrawala. Vidcrit: video-based asynchronous video review. In *Proceedings of the 29th annual symposium on user interface software and technology*, pages 517–528, 2016.

[33] Yi-Hao Peng, Jeffrey P Bigham, and Amy Pavel. Slidecho: Flexible non-visual exploration of presentation videos. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, pages 1–12, 2021.

[34] Christopher Scaffidi. Employers' needs for computer science, information technology and software engineering skills among new graduates. *International Journal of Computer Science, Engineering and Information Technology*, 8(1):1–12, 2018.

[35] Anna Stepanova, Alexis Weaver, Joanna Lahey, Gerianne Alexander, and Tracy Hammond. Hiring cs graduates: What we learned from employers. *ACM Transactions on Computing Education (TOCE)*, 22(1):1–20, 2021.

[36] Jeremy Warner, Amy Pavel, Tonya Nguyen, Maneesh Agrawala, and Björn Hartmann. Slidespecs: Automatic and interactive presentation feedback collation. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 695–709, 2023.

Appendices

Appendix A

Semi-structured interviews to understand user experiences in mock interviews

A.1 Evaluation study

- About D3: Minimize context switching for interviewers.
  - We placed the question tab as a real-time shared text editor like Google Docs, separate from the code editor. How was it different from how you usually present questions to an interviewee? How would you compare this approach with the existing approach if they are different? (asked to interviewer/interviewee twice)
  - (Mostly for interviewers) We also have solutions and interview tips. Did you use that information before, during, or after the interview? If so, how? (after responses) How do you find having this information useful, or do you think it is not necessary (if so, why)?
- About D1: Help interviewers give richer feedback.
  - If not using replay at all: Why's that? : I noticed you never replayed the code replay or meeting recording. As a person who you had to give feedback on the

spot, how do you feel about having access to the video? Do you think they are not necessary? (interviewer)

  - If only using the comments but without replaying anything:
    * I noticed that you clicked the timestamps of the comments and revisited a previous version of the code. Why did you revisit the code?
    * I noticed that you didn't necessarily play the video and listen to the conversation again. Do you think the replay is still necessary? if so why, if not why not?
    * Did you find having the comments and notes on the side of the editor help you provide feedback?
  - If using the comments and replaying:
    * How does being able to leave comments and replay videos helpful in giving feedback?

- About D2: Allowing interviewers to make notes discretely without interrupting the interview process
  - Not using (highlighting and leaving comments): The system has a function that hides the cursor so that you can select code and leave comments like Google Docs without letting the interviewee know you're highlighting. I noticed that you have not used the function. Do you think it is not necessary?
  - (not using hide cursor but highlighting and leaving comments) I noticed that you left comments on the code and did not hide a cursor. Do you prefer to let the interviewee know when you take the note or do you think you forgot about it?
  - Using it (hiding selection): How did you like or dislike the function that allows you to leave comments on code like Google Docs without showing the cursor?

- About D4: Help interviewers and interviewees follow each other easily
  - Did you follow the red lines to try to sync the viewport?
    - Yes: How was knowing where the interviewee was looking at helpful?
    - No: Do you think it is not necessary to visualize where the interviewee is looking at?
    - During the interview, would you rather automatically follow the interviewee the whole time or have the ability to scroll up and down as you did today?
  - Did use the scroll button: On what occasions would you use this button?
  - Did not use the scroll button: Why?
  - Did you find the mouse pointers helped you understand interviewee (interviewer)
- Overall, what did you like about the system for giving feedback to interviewees?
- What did you not like about the system?
- Do you think the system will be helpful for conducting technical interviews without giving feedback, if so how?
- Any immediate improvement that you can think of? (to both)

Appendix B

Information collection forms

B.1 Interviewers

- Email Address
- Have you ever interviewed a job applicant with a programming question? Technical interviews mean an interviewee is asked to write code (or pseudo-code) on a code editor or whiteboard before the interviewer. In our study context, asking simple computational concepts to interviewees is not a technical interview.
- Have you ever helped others prepare for technical interviews by conducting mock interviews with them before and giving them feedback?
- Are you confident in conducting a technical interview and giving them feedback after the interviews to help them prepare for the actual job interviews?
- (If you have helped others conduct mock interviews, have you ever felt hindered from providing helpful feedback to the person you helped?
- If you have helped others conduct mock interviews, when do you prefer to give feedback to the interviewees when you see something wrong in their codes?
- If you frequently provide feedback and suggestions while the interviewees are still working on a question, do you think their progress on that question improved after

your feedback?

- What job roles best describe you in the IT industry?
  - Software Development Engineer (SDE)
  - Machine Learning Engineer (MLE)
  - Data scientist/Data analyst
  - UI/UX designer
  - Web developer
  - Cyber security analyst
  - DevOps Engineer
  - Cloud engineer
  - Other...
- Please specify your most comfortable programming languages and skill sets.[Multiple selection questions]

  Languages:
  - Java
  - JavaScript
  - TypeScript
  - Python
  - C
  - C sharp
  - C++

  - SQL
  - GO
  - Kotlin
  - Other: open-ended answer

  Skill set:

  - Web development
  - Signalling
  - Sorting
  - Greedy algorithm
  - Dynamic programming
  - Concurrency
  - Hash table
  - Heap
  - Bit manipulation
  - Two pointers
  - Trees
  - Graphs
  - Data structures
  - Other:

- Can you bring two technical interview questions when we pair you with an interviewee for their practice? We are going to spend 20 minutes per question and 10 minutes for feedback for each question.

- What is the highest level of technical interview questions you are comfortable conducting as an interviewer, with a standard similar to LeetCode.com?
  - Easy
  - Medium
  - Hard

B.2 Interviewees

- Email address
- What type of jobs are you seeking?
  - Full time
  - Part-time/internship
  - Other…
- What job roles are you seeking in the IT industry? Please input "unsure" in the text box if you are unsure of that now.
  - Software Development Engineer (SDE)
  - Machine Learning Engineer (MLE)
  - Data scientist/Data analyst
  - UI/UX designer
  - Web developer
  - Cyber security analyst
  - DevOps Engineer

  - Cloud engineer
  - Other...

- Please specify comfortable programming languages for doing technical interviews.
  - Java
  - JavaScript
  - TypeScript
  - Python
  - C
  - C sharp
  - C++
  - SQL
  - GO
  - Kotlin
  - Other: open-ended answer

- Please specify technical interview questions you are familiar with (check all that apply).
  - Sorting
  - Greedy algorithm
  - Dynamic programming
  - Concurrency
  - Hash table
  - Heap

  - Bit manipulation
  - Two pointers
  - Trees
  - Graphs
  - Data structures
  - Other:

- How many technical interviews have you done in the past? Technical interviews mean an interviewee is asked to write code (or pseudo-code) on a code editor or whiteboard before the interviewer. [Please input your answer in numerical forms, e.g., 0, 1, 2, etc.]
- Have you been preparing for the technical interviews recently? Please specify how many hours you spend each week preparing technical interviews on average. [Please input your answer in numerical forms, e.g., 0.5, 1, 1.5, etc.]
- How many mock technical interviews have you done in the past as part of your interview preparation? [Please input your answer in numerical forms, e.g., 0, 1, 2, etc.]
- How do you practice/prepare communication skills when you practice for technical interviews?
- Have you ever used Leetcode.com to practice algorithm questions for technical interviews? If so, please select the highest level of problem you are able to solve consistently.
- If you don't use Leetcode.com, are you using other platforms to practice algorithm questions? Please specify the platform you are using as well as the level of difficulty you can consistently solve on this platform.
- Please select which of the following best describes your level of confidence in solving the coding questions in the technical interview for your desired job.