DiffCoder: A GPT-Powered WorkFlow for Collaborative Qualitative Analysis (CQA)

Anonymous Author(s)*

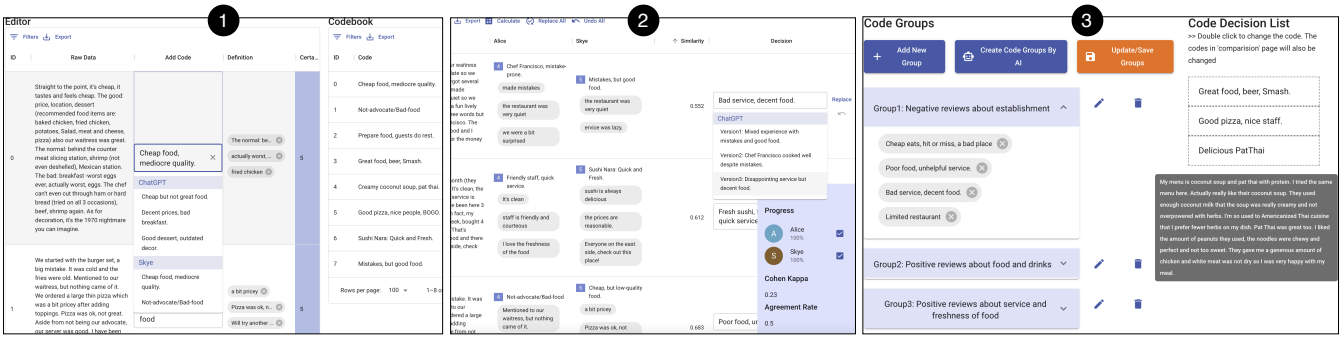

Figure 1: DiffCoder leverages LLMs during three key steps of CQA: 1) code suggestions (on demand) during independent open coding, 2) conflict mediation during iterative discussion among the coding team, and 3) group suggestions when creating code groups for a codebook.

ABSTRACT

The Collaborative Qualitative Analysis (CQA) process can be time-consuming and resource-intensive, requiring multiple discussions among team members to refine codes and ideas before reaching a consensus. To address these challenges, we introduce DiffCoder, a system leveraging Large Language Models (LLMs) to support three CQA stages: independent open coding, iterative discussions, and the development of a final codebook. In the independent open coding phase, DiffCoder provides AI-generated code suggestions on demand, and allows users to record coding decision-making information (e.g. keywords and certainty) as support for the process. During the discussion phase, DiffCoder helps to build mutual understanding and productive discussion by sharing coding decision-making information with the team. It also helps to quickly identify agreements and disagreements through quantitative metrics, in order to build a final consensus. During the code grouping phase, DiffCoder employs a top-down approach for primary code group recommendations, reducing the cognitive burden of generating the final codebook. An evaluation involving 16 users confirmed the usability and effectiveness of DiffCoder and offered empirical insights into the LLMs' roles in CQA.

CCS CONCEPTS

• **Human-centered computing** → **Collaborative and social computing systems and tools**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

KEYWORDS

data annotation, agreement, interrater reliability, AI-assited

ACM Reference Format:
Anonymous Author(s). 2018. DiffCoder: A GPT-Powered WorkFlow for Collaborative Qualitative Analysis (CQA). In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 21 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Collaborative Qualitative Analysis (CQA) is an essential qualitative method that ensures a reliable and comprehensive interpretation of qualitative data [2, 13]. CQA process involves multiple researchers conducting individual coding and sharing their perspectives to reach a final consensus [44]. To establish this consensus, they need to discuss iteratively for several rounds [35] (see Figure 2).

Although CQA is critical to ensuring the credibility, rigor, and representativeness of the interpretation, research has found that it is time-consuming and effort-demanding due to its iterative nature [14, 19, 38, 50]. This challenge limits a more common usage of CQA, leading individuals to resort to independent coding, which might be perceived as a faster and more efficient alternative. Nonetheless, independent coding may give rise to biases and produce less reliable results [2], as the absence of discussion and collaboration could cause the final outcomes to contain the individual coder's inherent biases.

In the past few years, various prototypes and systems have been proposed to explore the potential of applying artificial intelligence (AI) in qualitative analysis to assist in the process of code proposition. For instance, Cody [45] predicts users' codes for selected data based on the coding history, Scholastic [24] provides users with a comprehensive coding workflow that includes sampling, coding, and categorizing, while PaTAT [20] offers coding pattern prompts as references during the coding process. Each of these

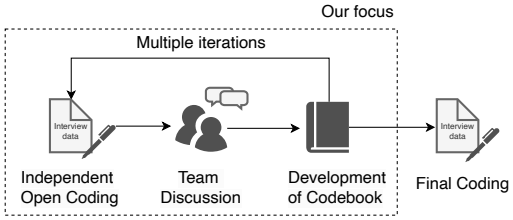

Figure 2: Collaborative Qualitative Analysis (CQA) [12, 44, 47] is an iterative process involving multiple rounds of discussion among coders to reach a final consensus.

AI systems showcases the capability of AI to aid qualitative researchers in expediting the coding process and minimizing human effort [18, 24, 28, 45], but only on specific phases of the coding process.

Despite the focus on AI integration in individual qualitative analysis, a comprehensive study of its application in various stages of CQA is still lacking, as the majority of current CQA assistance continues to depend on traditional methods. For example, Zade et al. [50] suggested to enable coders to order different states of disagreements by conceptualizing disagreements in terms of tree-based ranking metrics of diversity and divergence. Ganji et al. [19] introduced Code Wizard, which utilizes the uncertainty of all coders towards the codes they have assigned to draw attention to highly ambiguous codes. Drouhard et al. [15] developed Aeonium, a visual analytics system that highlights ambiguity and inconsistency and offers features to navigate through and resolve them. These works underscore a key aspect of CQA: reconciling agreements and disagreements among codes from all coders and fostering a shared understanding among coders who might have diverse viewpoints and biases [36]. In our research, we build on their findings and integrate an AI agent to assist on this critical aspect.

Recently, as Large Language Models (LLMs)[1] such as OpenAI's GPT series [2] advance and demonstrate exceptional capabilities in understanding and generating text, the commercial qualitative analysis platform, Atlas.ti, has integrated OpenAI's GPT model[3] into its system, allowing for one-click code generation in a short period. Consequently, we capitalize on the potential of LLMs and apply them to the critical CQA processes mentioned earlier, in order to investigate the role of LLMs during this process.

To this end, we propose DiffCoder, a prototype incorporating the GPT-3.5 model, which can: 1) offer AI-generated code suggestions and allow users to document their coding decision-making information (e.g., keywords and certainty) during the individual open coding phase, 2) share decision-making data and provide quantitative metrics to swiftly identify (dis)agreements during the discussion phase, and 3) enable a top-down method for organizing codes to streamline the codebook creation process. DiffCoder was developed to address both collaboration challenges among coders and investigate the roles LLMs can fulfill at each stage of qualitative analysis (QA).

In our assessment of DiffCoder involving 16 participants, we had them work in pairs (8 pairs in total) to carry out multiple collaborative qualitative analysis tasks using DiffCoder and Atlas.ti Web. We discovered that during the initial phase, it is essential to balance the LLM's capabilities while preserving user autonomy. In the discussion phase, by improving coders' understanding of the same data units and presenting their initial coding decisions, as well as identifying agreements and disagreements through quantitative metrics, we found that our participants were able to build mutual understanding and elevate discussion quality, thereby streamlining the consensus-building process. In the final phase, utilizing a top-down approach to generate code groups was useful to reduce their cognitive load. When compared to an existing coding platform (Atlas.ti Web), our participants expressed a preference for DiffCoder, primarily attributing this to the AI assistance offered and the user-friendly interface, specifically crafted for collaboration.

Our study consequently paves the way for LLMs-empowered QA and CQA tools and also uncovers critical challenges and insights into both human-AI and human-human interactions within the context of qualitative analysis. We make the following contributions to the field of AI-assisted CQA:

(1) We design and develop a system, DiffCoder, which enables coders to perform independent coding with the assistance of LLMs, facilitating consensus-building through more efficient discussions and generating a final codebook using a top-down approach.
(2) We offer a range of design considerations and guidelines that should be taken into account when incorporating LLMs into the CQA process.
(3) We present insights from user evaluations concerning the role of LLMs at various stages of the CQA process, which can inform the design of future AI-assisted CQA systems.

2 BACKGROUND AND RELATED WORK

2.1 Collaborative Qualitative Analysis (CQA)

CQA is a qualitative data analysis method that involves multiple researchers with diverse perspectives [2, 13, 44], collaboratively working on shared data for a comprehensive, agreed-upon interpretation [13], promoting critical reflection and reducing individual bias [51]. It also boosts research credibility and rigor [13], yielding inter-coder reliability (IRR) [3], thus improving trustworthiness and significance.

From a practical standpoint, Richards et al. [44] outlined a six-step CQA process grounded in established qualitative theories and methods, such as Grounded Theory [12, 22] and Thematic Analysis [6]. The six steps include "planning and organization", "open and axial coding", "development of a primary codebook", "pilot testing the codebook", "final coding process", and "reviewing the codebook and finalizing the themes", which enables teams to detect, examine, and present qualitative data patterns. Researchers frequently utilize tools like Excel, Google Sheets, or qualitative software for these steps [21], but the process can be manual, laborious, and

[1]This paper uses AI, LLMs, and GPT interchangeably to refer to the wider field of Artificial Intelligence, specifically, large language models. GPT, which stands for Generative Pre-trained Transformers, is one such large language model, and in this paper, it specifically refers to products developed by OpenAI, such as ChatGPT.
[2]https://platform.openai.com/docs/models/gpt-3
[3]Announced on 28th March 2023: https://atlasti.com/ai-coding-powered-by-openai

time-consuming [19, 50]. Extensive discussions are needed to resolve differences and reach a consensus [29], with a final codebook established for coding consistency.

To tackle this issue, AI has been considered to enhance the efficiency of qualitative analysis [18, 28]. From a theoretical standpoint, Muller et al. [37] examined the intersections between Grounded Theory and Machine Learning, suggesting that both methods have great overlap, as both require constant comparisons among data vs. topic modeling, coding families vs. ground truth, and iterative processes to achieve the final interpretation.

In this paper, we apply Richards et al.'s practical guide, focusing on its key steps including "open coding", "iterative discussion", and "codebook development", and aim to investigate how AI can promote more efficient qualitative analysis, with a focus on collaboration.

2.2 (AI-assisted) QA Systems

The utilization of AI to aid in different aspects of the qualitative coding process has garnered increasing interest [9, 18, 28, 37]. Feuston et al. [18] provided a summary of different ways in which AI can be beneficial in qualitative coding at various stages of QA. For example, AI may assist in semantic searches for early insights, inductive coding for generating new perspectives, developing clusters of codes created by humans, identifying similar words and patterns after initial identification, and learning human coding patterns for application to unseen data. In particular, for the early exploration stage [33, 41], AQUA [33] utilizes unsupervised methods to generate topic categories and hierarchical representations of free-text responses, which aid in expediting data interpretation. For the late stage, Xiao et al. used the newest version of GPT-3 to assist with labeling codes after the codebook had already been created [48]. On the whole stage of coding, Cody [45] utilizes supervised techniques to enable researchers to define and refine code rules that extend coding to unseen data, while PaTAT [20] provides a program synthesizer that learns human coding patterns and serves as a reference for users.

Although AI has been widely investigated in individual coding processes, research on aspects of qualitative analysis involving multiple parties [21] still relies on traditional methods. These systems aim to facilitate code comparison and discussion among coders, as well as to identify disagreements and ambiguities [9]. For instance, Aeonium [15] helps coders identify the ambiguity of the CQA process, allowing coders to interact with shared code definitions, and track coding history during both code development and the final coding process. Ganji et al. [19] developed Code Wizard, an Excel-embedded visualization tool that supports the code merging and analysis process of CQA, allowing coders to aggregate each individual coding table, automatically sort and compare the coded data, calculate IRR, and generate visualization results.

There are also several commercial software (e.g., NVivo, MaxQDA, and Atlas.ti) available that support collaborative coding in various ways. For code comparison, they allow users to export coded documents and merge codes across different coders. For code discussion, they enable coders to enter memos to note concerns and ambiguities that can be addressed during discussions. In particular, Atlas.ti Web version allows coders to collaborate simultaneously on a shared online space to share data and codes. To accelerate coding efficiency, Atlas.ti Web has already integrated OpenAI's GPT model for one-click code generation. Nonetheless, this generation is rather simplistic and overlooks user autonomy [28], potentially replacing human evaluation. Other software predominantly depends on manual human evaluation or basic AI applications, such as word frequency counting or sentiment analysis. The utilization of AI at various stages of qualitative analysis, particularly when involving multiple coders, remains largely unexplored.

With the recent advancements in LLMs, such as GPT-4 and GPT-3.5, their impressive text generation, comprehension, and summarization capabilities offer significant potential for integration into the CQA process. In this work, we aim to explore the roles that the capabilities of LLMs could play during various stages of the CQA process.

2.3 Agreements and Consensuses Building

Constructing consensus [35] via a repetitive dialogue process among coders is a crucial stage in CQA, and it is also the most labor-intensive and resource-demanding phase. In particular, qualitative codes may be broadly defined, enabling their application to various text units, leading to inconsistencies in comprehension among different coders [23]. Nevertheless, to establish an agreement and even calculate IRR [35], coders must apply their codes to the same unit of analysis or text. Two prevalent approaches to achieve this are: 1) allowing the initial coder to finish coding a unit before another coder commences work on the same unit [14, 30, 40], and 2) predefining a fixed text unit, such as sentences, paragraphs, or conceptually significant "chunks" [30, 40].

Furthermore, attaining high levels of consensus can be difficult due to the inherent subjectivity and intricacy of qualitative data. During discussions, coders should identify agreements and disagreements, explain their reasoning for coding choices, and make final decisions. For instance, codes with varying expressions but identical meanings should be consolidated, while codes with differing meanings and expressions must be appropriately discussed or clarified to other coders. This may necessitate revisiting the text if there is an excessive amount or if they have forgotten their initial coding rationale after a long period, making the process cognitively demanding. To facilitate this process, researchers [9, 10, 19, 50] suggest sorting the text from according to its ambiguity, allowing coders to concentrate on disagreements and ambiguities to save time and effort. Drouhard et al. [15] also recommend that users document the definitions for their proposed codes. However, creating a mutually understood basis for communication, as described in Clark's concept of "grounding" [11], is crucial for facilitating transparent communication and the sharing of knowledge and decision-making information among coders. This is essential for achieving consensus but has not been prioritized in current assistant tools for CQA.

In summary, supporting the essential stages of CQA, including independent open coding, iterative discussion, and codebook development, using the text comprehension and generation abilities of LLMs, as well as building a common ground to promote efficient discussion is a promising but unexplored topic. We are inspired to investigate the potential roles of LLMs in these stages and design improved interfaces that can facilitate efficient consensus building.

3 DESIGN CONSIDERATION AND EXPLORATION

3.1 Methodology

Drawing on previous qualitative analysis theories and guides, we first made a triangulation with the existing QA applications, e.g., Atlas.ti Web[4], MaxQDA Team Cloud[5], nVivo Collaboration Cloud[6], and Google docs. We examined their public CQA documents and developed a set of design considerations for various stages of CQA, specifically designed to enhance discussion efficiency during collaboration.

After implementing a first version of the prototype, we invited five HCI experts (see Appendix Table 2) with an average of 3-year qualitative analysis experience to join a pilot evaluation and interview session. They were shown the CQA flow with the system first, and then discussed their feelings, questions, and provided suggestions on how to improve the prototype. We then derived additional design considerations based on their feedback, concerns and suggestions. We summarized a set of design considerations (DC) presented below.

3.2 Design Considerations

DC1: Facilitating Proper Data Sharing Among Coders Across Various Stages. Team members should have seamless access to projects and coding data within their interface, thereby eliminating the cumbersome export, upload, and download steps commonly seen in software like Atlas.ti Desktop, nVivo, and MaxQDA. Atlas.ti Web tackles this problem by employing a web-based project system that enables the lead coder to invite others to view and collaborate on the same data in real-time, as long as they have an account. Consequently, we opt for a web-based interface for optimal simplicity and ease of use. One concern we found with the Atlas.ti Web is that it does not facilitate independent coding, as all raw data, codes, and quotations are constantly visible by everyone, which could potentially influence other coders' coding processes.

To tackle this challenge, we intend to develop a feature that allows users to work independently on their individual coding phase without seeing others' codes, with their work being automatically saved. Nonetheless, we recognize the importance of tracking team members' progress for efficient project management and mutual awareness of each other's availability [4, 43], particularly when coders are remotely collaborating and communicating [7]. As such, a progress panel should be designed to check any coder's progress. During the discussion and grouping phases, coders should be able to effortlessly view their peers' information with a single click.

Additionally, to facilitate easy codes comparisons and IRR calculations [19, 30, 40], standardizing data units for coding is essential. As per Saldana's qualitative coding manual [46], coders may use a "splitter" (e.g., line-by-line) or a "lumper" (e.g., paragraph-by-paragraph) approach. We thus predefined two unit-of-analysis options: sentence coding and paragraph coding. The lead coder selects the level when creating a project, ensuring consistent coding and discussion across all coders.

[4]https://atlasti.com/atlas-ti-web
[5]https://www.maxqda.com/help-mx20/teamwork/can-maxqda-support-teamwork
[6]https://help-nv.qsrinternational.com/20/win/Content/projects-teamwork/nvivo-collaboration.htm

DC2: Offering AI-Generated Code Recommendations While Preserving User Autonomy. Atlas.ti Web has integrated the OpenAI GPT model into its coding system, enabling users to generate codes for a document in just seconds, which is quite impressive and time-saving. Their "AI coding" feature auto-highlights quotations, which users can edit afterward; however, this limits user control over the generation process and often requires time-consuming re-editing. As Jiang et al. [28] suggested, AI should not replace human autonomy, participants in their interview said that "I don't want AI to pick the good quotes for me...". AI should only offer recommendations when requested by the user, after they have manually labeled some codes, and support the identification of overlooked codes based on existing ones. For our system, we view user autonomy as a vital factor during the coding process. Therefore, our system may provide AI-generated suggestions when editing an existing code. Suggestions would be provided by a drop-down list for the selected unit after a short period, giving them time to also read the text. Additionally, highlighting similar codes in their coding history, which already include their editing patterns, serves as a valuable reference source. We hope this approach can enhance coding consistency across the entire process.

DC3: Facilitating Mutual Understanding and Concentrated Discussion. Common ground [39, 42] pertains to the information that individuals have in common and are aware that others possess, a notion rooted in the grounding process in communication [5, 11]. Grounding is achieved when collaborators engage in communication and convey understanding confirmation [5]. A lack of common ground can lead to distrust, misunderstandings, poor team performance, and decision-making. During the CQA discussion, two coders identify their codes for the same data, share these codes, compare them, and then detect discrepancies through communication. Ideally, coders would possess identical code expressions and comprehension. However, they may assign different meanings to the same codes or have similar interpretations for vastly dissimilar codes, as shown in Figure 3. Coders must address these situations, reach a final consensus, and then proceed to the next group. For traditional CQA work, the Aeonium prototype [15] features a "comparison panel" that displays the codes and definitions of two coders for a side-by-side comparison of a single unit of analysis, providing an easily accessible "common ground" for sharing and identifying differences. Meanwhile, Zada et al. [50] introduced an intuitive tree-based disagreement ranking method to help individuals quantitatively comprehend their differences. With this in mind, we considered employing a side-by-side comparison for each unit of analysis and enabled two coders to employ metrics like similarity ranking, IRR, and agreement rate to quickly highlight agreements and disagreements with a single click. By pinpointing disagreements, coders can rapidly navigate through codes and concentrate on areas that require thorough examination [15].

However, solely depending on code words might not offer a complete understanding of each coder's codes. The Aeonium prototype [9, 15] addressed this by requesting users to provide code definitions for each code. This approach, though, may impose additional burdens on human coders to supply definitions. As an alternative, we opted to allow coders to select keywords or phrases as supporting evidence for their suggested code, which has been

demonstrated to be a critical source for matching potential data that needs coding [34, 45]. Additionally, Ganji et al. [19] suggested that determining users' certainty for codes is crucial for assessing data analysis quality and identifying ambiguity. We thus considered proposing incorporating users' self-labeled certainty scores as a reference for comparison.

DC4: Facilitating Code Decision-making When A Consensus Cannot Be Reached. When a coding team needs to make a final decision on codes, they often encounter diverse coding proposals due to the varying personalities and perspectives of the coders. To reach a consensus, the team engages in debates or spends time proposing code expressions that satisfy all coders [17]. This can significantly lengthen the discussion. One observation as per Jiang et al. [28], is that team leaders or more senior members may be the ones to decide the final codes, potentially introducing bias. As a result, achieving a good coding outcome that is cost-effective and fair, and avoids negative effects can be a difficult but crucial challenge [17, 27]. To address these challenges and help build a consensus, we decided to leverage LLMs as a third-party mediator, or group recommender system [27], integrating users' codes and original data to provide alternative suggestions. We believe that coders would be more receptive if the additional suggestions are not strongly pushed by another human, but instead by an AI agent.

DC5: Facilitating High Level Code Trees Formation. A crucial feature in prevalent QA software (Atlas.ti, MaxQDA, and nVivo) is the code manager, which enables coders to monitor and modify their codes while obtaining a comprehensive view of the presently assigned codes. Notably, the code manager facilitates discussions and propositions of multiple code groups, as well as assisting users in reusing codes throughout the coding process. We opted to offer two types of codebook managers: an individual codebook manager and a group codebook manager. The individual version is accessible only within the individual open coding page, whereas the group version can be shared and edited by all coders. Additionally, Feuston et al. [18] discovered that some participants employed AI tools to assist them in automatically generating final code groups for human-assigned codes. We recognized an opportunity to explore the latest advancements of LLMs in generating code groups, which could potentially streamline the coding process and improve code organization.

4 DIFFCODER SYSTEM

With the above design considerations, we propose our final version web-based online CQA system, DiffCoder, with a workflow integrating GPT in different phases (see Figure 3).

4.1 DiffCoder Workflow & Usage Scenario

This section provides an example scenario to demonstrate the usage of DiffCoder. Suppose two coders Alice and Bob are conducting qualitative coding for their qualitative data. The lead coder, Alice, first creates a new project on DiffCoder, then imports the data, specifies the level of coding as "paragraph", and invites Bob to join the project (as shown in Figure 4). After clicking on "create project", DiffCoder's parser will split the imported raw data into units (paragraph in this case). The project can be seen on both coders' interfaces.

4.1.1 Phase 1: Independent Open Coding. In the first phase, Alice and Bob can individually propose codes for each unit (see Figure 5). If Alice wants to propose a code for a sentence describing food, she can either craft her own code (e.g., "Quiet but bad"), choose from code recommendations provided by the GPT model (e.g., "Quiet but bad service", "Wrong choice of food", "Negative eating experience"), or picking one of the top three most relevant codes discovered in her code history (e.g., "Mistake and bad experience"), and making modifications as needed. She can select relevant keywords/phrases (e.g., "made mistakes", "the restaurant was quiet") from the "Raw data" cell that support her proposed code, which will be added into the "Keywords Support" cell beside her proposed code. She can also assign a certainty level, ranging from 1 to 5, to the code. This newly generated code will be included in Alice's personal codebook list and can be viewed at any time. Additionally, they can check the progress of each other in progress bar during coding process (see Figure 6).

4.1.2 Phase 2: Code Merging and Discussion. After completing her coding, Alice can select the checkbox next to Bob's name once she sees that his progress is at 100%. Subsequently, she can click the "calculate" button to generate quantitative metrics such as similarity scores and IRR (Cohen's Kappa and Agreement Rate) for all units. The rows can then be sorted according to the ranking of the similarity scores from the highest to the lowest. Alice can share her screen via a Zoom meeting with Bob to compare their codes, starting from code pairs with high similarity scores. For instance, Alice's code "Quite but bad" with a certainty of 4 includes "waitress made mistakes" and "the restaurant was quiet" supports, while Bob's code "Quiet but lazy service" with a certainty of 5 includes "the restaurant was quiet" and "service was lazy" as Keywords Support. The similarity score might be 0.7, showing a high overlap between each other. During the discussion, they both agree that the final code should contain the word "service" due to their similar Keywords Support. Eventually, they agree on the merged code "quiet but bad service," but if they cannot reach a consensus, they can ask GPT to provide suggestions (e.g., "quiet but wrong service" from GPT). Alice then needs to go back and check if the original text contains the word "service." Once they arrive at a final code decision, she can click on "Replace" to replace the original codes, resulting in an update of the Cohen's Kappa and Agreement Rate. This action can be undone by clicking on "Revert."

4.1.3 Phase 3: Code Groups Generation. Once Alice and Bob have agreed on the final code decisions for all the units, the code decision list will be displayed in the code group interface, as shown in Figure 7. They can also view the same data (decision list and code groups) using their own interface. For further discussion, Alice can continue to share her screen with Bob on Zoom. She can hover over each code decision to refer to the corresponding raw data and double-click to edit the code decision. They can collaborate to propose the final code groups by clicking on "Add New Group." For instance, a group name like "Good food with minor setbacks" can include "Good food, beer, Smash" and "Mistakes but good food." Alternatively, they can request GPT assistance by clicking on the

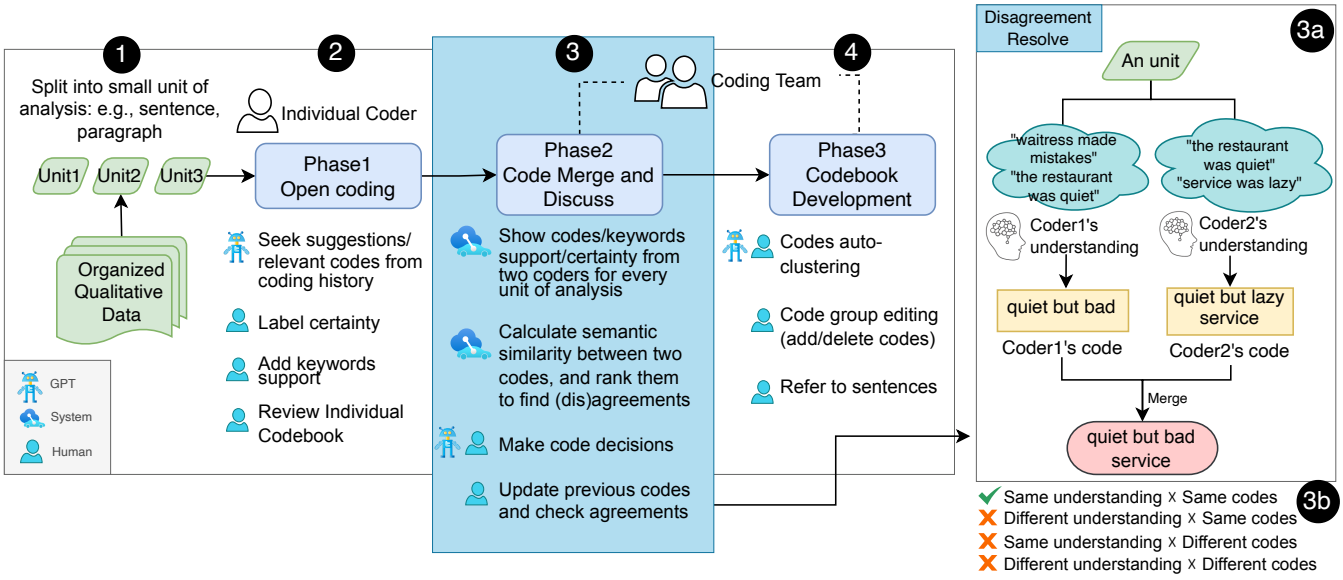

Figure 3: The Whole Workflow of Applying LLMs in CQA Using DiffCoder. The lead coder first 1) split qualitative data into small units of analysis, e.g., sentence, paragraph. Two coders 2) independently perform open coding with GPT assistance; 3) merge, discuss, and make decisions on codes, assisted by GPT; 3a) merge codes when coders have different understandings × different codes; 3b) address potential (dis)agreements; 4) utilize GPT to generate code groups for decided codes and perform editing.

"Create Code Groups by AI" option to automatically generate five code groups and place the individual code decisions into them. These suggestions can still be manually edited by Alice and Bob. Once they finish grouping, they can proceed to report their findings as necessary.

4.2 Key Features

4.2.1 Creating New Project. As outlined in DC1, a web-based platform is designed to provide the team with a shared space that facilitates efficient data sharing and uniform access to the unit of analysis. To initiate the coding process, the lead coder can first create a project and preprocess the data, as demonstrated in Figure 4.

Dividing Data into Units. Before the data is imported into the editing interface, the qualitative data needs to be segmented into the same data units, such as sentences or paragraphs, that have been previously agreed upon by both coders in the project. If the data is to be coded sentence by sentence but has not yet been divided into individual sentences, the system can assist in splitting it. The data may be in CSV format with distinct units in separate cells or in a txt format, in which case it will be divided accordingly.

Inviting A Collaborator. The project owner has the authority to invite another coder to join the project, which will be displayed on both coders' project interfaces. While both coders can edit the project, only the owner can delete it.

4.2.2 Editing. Once the two coders have produced a final code decision list, they can move on to Phase 1: Independent Open coding, as depicted in Figure 5. As emphasized in DC2 and DC5, the dropdown list suggestions and individual codebook are designed to enable AI support while preserving user autonomy and helping users monitor their codes. As mentioned in DC3, the inclusion of keyword support and a labeling certainty feature have been designed to facilitate mutual understanding and information sharing.

Adding a Code. By clicking on the "Adding Code" cell, the user can add a code to the interface. The user can either create their own code, select a code from the code history, or get code suggestions from the AI agent. In case the user wants to select a code from the history, the AI agent will suggest the three most relevant codes, which can then be modified to generate the final code.

Labeling Certainty. Additionally, the user can indicate their certainty of the code on a scale of 1 to 5, where 1 represents the lowest certainty and 5 represents the highest certainty.

Adding Keywords Supports. To aid in code discussion and merging in the second phase, the user can select keywords or phrases and add them as support by right-clicking on them. These selected words will be added to the "Keywords Support" cell, recording the user's understanding of the text.

Codebook. The newly added code will appear in the codebook list, located on the right side of the interface. This codebook list helps the user refer to the codes in the code history. The user can modify the code in the Codebook, and the corresponding codes in the original position will be updated accordingly.

4.2.3 Code Merging and Discussion. Once the two coders have produced a final code decision list, they can move on to Phase 2:

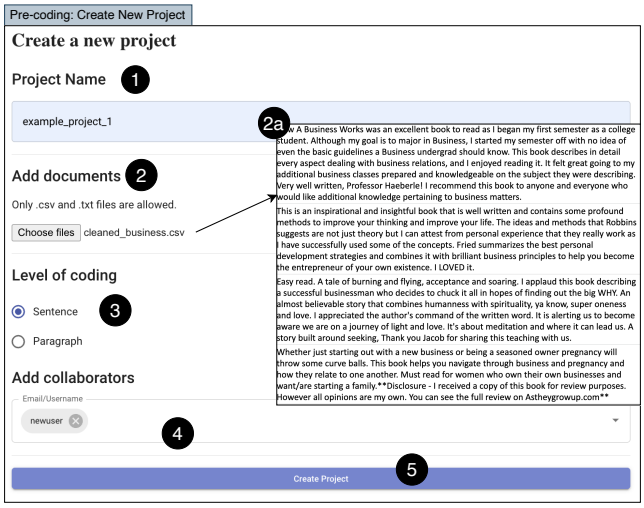

Figure 4: Creating New Project Interface. The lead coder can 1) input the project name, 2) import previously agreed upon data units between coders, 2a) see an example of the imported data units, 3) specify the level of coding (i.e., sentence or paragraph), 4) invite another coder to join the project, and 5) create project.

Merging & Discussion, as depicted in Figure 6. As DC3 described, for the purpose of facilitating discussion and merging, the codes from all coders are displayed side-by-side along with the corresponding original sentences, the keyword support, and labeling certainty indications. The quantitative metrics can be shown once requested, and the (dis) agreements can be highlighted accordingly. To satisfy DC1, this side-by-side comparison will only be shown only after both coders finished independent coding. To satisfy DC4, a GPT agent will provide three versions of suggestions based on the raw data and two codes from two different coders upon request.

Coding Progress and IRR. The coding progress of two coders is displayed in the interface as a percentage (0-100%). The checkbox next to each coder's name can only be checked by the user after both coders have completed their individual coding. Additionally, the interface displays Cohen's Kappa and Agreement Rate.

List of Code Pairs. Upon selecting the checkbox, two lists of code are presented on a single page, whereby the codes for each unit from both coders are automatically aligned. The corresponding degree of certainty and code support, such as keywords or phrases, is shown beside and below each code.

Identifying Agreements and Disagreements. By clicking on the "Calculate" button, the user can compute the Similarity and Cohen's Kappa for each code pair of every unit. This operation can be completed within 3-10 seconds for all the sentences. The similarity score between two codes varies between 0 and 1, where 0 indicates a low level of similarity, and 1 denotes a high level of similarity. The similarity scores can be sorted from highest to lowest, with the codes having the highest scores indicating the greatest agreements.

Discussing and Making Coding Decisions. Users can collaborate and arrive at a coding decision by discussing with each other. The original codes, along with the corresponding certainties of both coders and the code definitions, are displayed. Additionally, users can seek suggestions for coding decisions from the AI agent. Based on the raw data and two existing codes, it can generates three possible code suggestions. Both coders' codes are utilized to account for their perspectives while also considering raw data to address any potential coding discrepancies within the text. Users can select a suggestion and modify it as needed to arrive at a final version.

Replacing Codes and Updating IRR. After arriving at a coding decision, the user can click on the "Replace" button to substitute the original codes with the final version. Then, by clicking on "Calculate," the user can update the IRR and similarity scores. It is possible to undo all replacements.

4.2.4 Code Groups Generation Interface. Once the two coders have produced a final code decision list, they can move on to Phase 3: Code Group Generation, as depicted in Figure 7. In order to meet the requirements of DC5, a feature that enables GPT to automatically generate code groups has been designed. The group codebook manager is synchronized and shared in real-time across the interfaces of both coders.

Code Decision List. The system deduplicates the final code decisions and presents them on the right-hand side. Users can edit the code decision by double-clicking on the code decision list, which updates the previous codes accordingly. Hovering over the code displays the original raw data.

Adding New Code Group. The interface includes buttons for creating, renaming, or deleting a code group. Users can drag code from the code decision list to the code group and also delete code within each code group.

Generating Code Groups by AI. To help users begin with predefined groups rather than starting from scratch, the interface enables them to request GPT to automatically organize code decision lists into multiple code groups. Users can also regenerate, rename, and modify these code groups, providing further customization without starting from scratch.

Saving and Updating Code Groups. Once users have formed the final code groups, they must click the "Save and Update" button to access them at any time.

4.3 System Implementation

4.3.1 Web Application. The front-end implementation makes use of the react-mui library[7]. Specifically, we employed the DataGrid component[8] to construct tables in both the "Edit" and "Compare" interfaces, allowing users to input and compare codes. These tables auto-save user changes through HTTP requests to the backend, storing data in the database to synchronize progress among collaborators. For each data unit, users have their own code, keyword supports, certainty levels, and codebook in the Edit interface, while sharing decisions in the "Compare" interface and code groups in the

[7]https://mui.com/
[8]https://mui.com/x/react-data-grid/

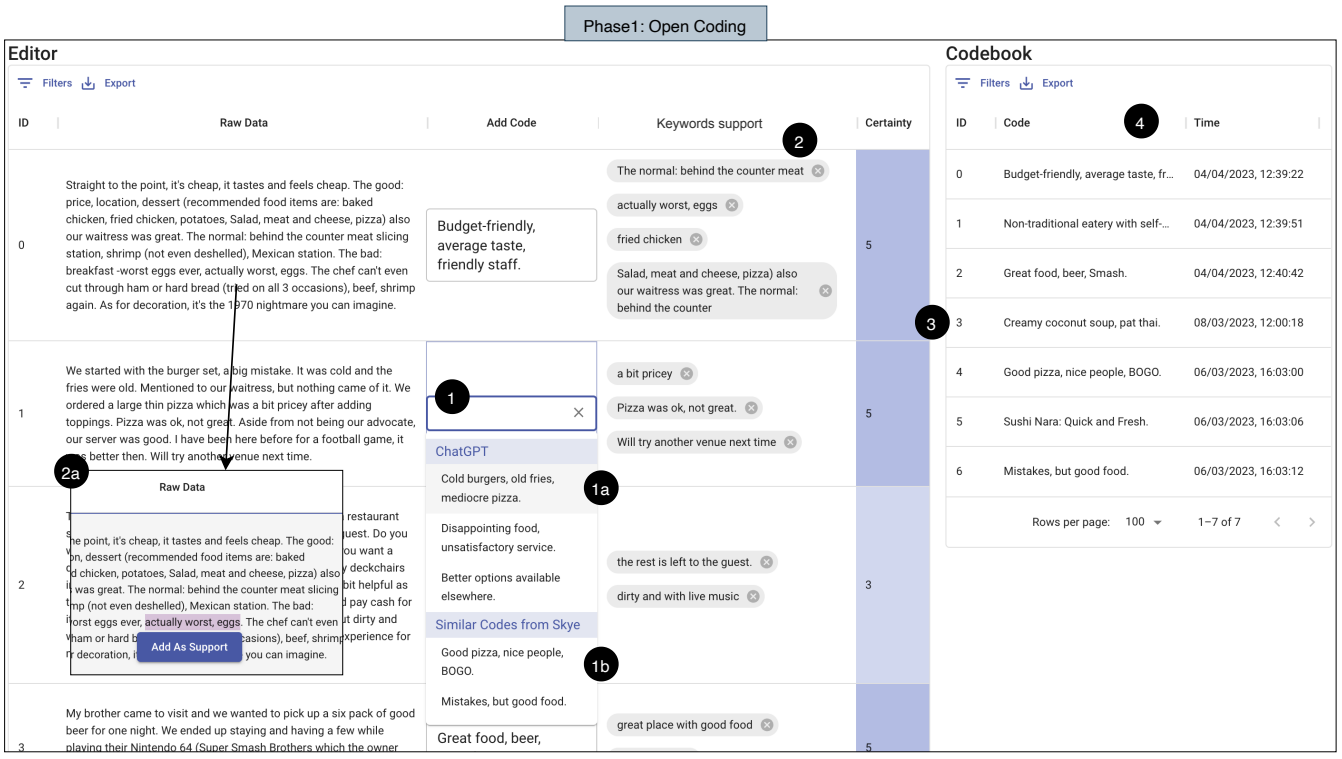

Figure 5: Editing Interface: The editing interface enables users to assign codes to data units through several steps: 1) input custom codes, 1a) opt for the AI agent's recommendations, 1b) choose from the top three pertinent codes; 2) include supporting keywords or phrases by 2a) selecting from raw data; 3) assign a certainty level ranging from 1 to 5; and 4) review and modify the individual codebook.

"Codebook" interface. To prevent users from viewing collaborators' codes before editing is complete, we restrict access to other coders' codes and only show everyone's progress in "Compare" interface. We also utilized the foldable Accordion component[9] to efficiently display code group lists in the "Codebook" interface, where users can edit, drag and drop decision objects to modify their code groups. The backend leverages the Express framework, facilitating communication between the frontend and MongoDB. It also manages API calls to the GPT-3.5 model and uses Python to calculate statistics such as similarities.

4.3.2 Data Pre-processing. We partitioned raw data from CSV and txt files into data units during the pre-processing phase. At the sentence level, we segmented the text using common sentence delimiters such as ".", "...", "!", and "?". At the paragraph level, we split the text using \n\n.

4.3.3 Prompts Engineering. DiffCoder leverages OpenAI's ChatGPT model (gpt-3.5-turbo) [10] to provide code and code group suggestions. All prompts are listed in Appendix Table 3.

Code Suggestions. In the Editing Interface, code suggestions are offered in two ways: 1) descriptive codes for raw data, and 2) relevant codes derived from coding history. For 1), we prompt GPT with: *Create three general summaries for [text] (within six-word).* The six-word constraint was introduced after observing GPT's tendency to generate long summaries during testing. This limitation ensures GPT delivers concise and targeted code suggestions. For 2), DiffCoder produces the three most relevant codes from coding history using the following prompt: *Identify the top three codes relevant to this [text] from the following code list: 1. [Code] 2. [Code]...*, along with one shot prompt: *Here is the format of returned results: 1. code content 2. code content 3. code content.* Upon a user's request, the prompt and original text (specified in *[text]*) are sent to the OpenAI API, which generates three distinct code suggestions and the three most relevant codes from coding history. The responses for both parts appear in a dropdown list when the user clicks on the editing cell, allowing for easy selection. To ensure code suggestions have diversity without being overly random, the temperature parameter is set at 0.7.

Making Code Decisions. To generate code decisions upon user request, we prompt GPT with: *Create three concise, non-repetitive, and general six-word code combinations for the [text] using Code1 ([Code]) and Code2 ([Code])*, where Code1 and Code2 represent codes from Coder1 and Coder2 for the given raw data unit ([text]).

[9]https://mui.com/material-ui/react-accordion/
[10]https://platform.openai.com/docs/models/gpt-3-5

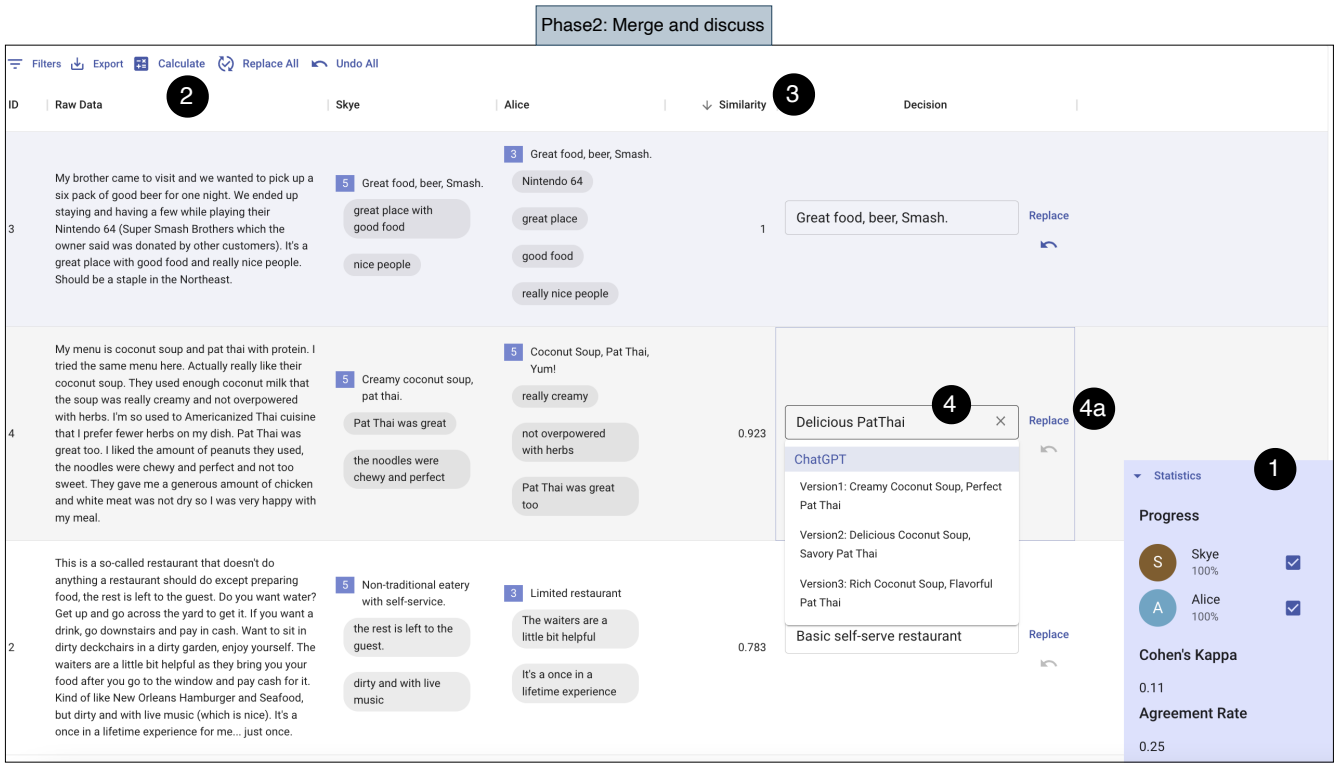

Figure 6: Comparison Interface. Users can compare and reach a consensus by following these steps: 1) reviewing another coder's progress and clicking on the checkbox once both individuals complete their coding, 2) computing the similarity between code pairs, 3) sorting the similarity scores from highest to lowest and identifying agreements, and 4) making a decision based on the initial codes, raw data, and code supports through discussion or by utilizing the AI Agent's three potential code decision suggestions to amend the decision. Additionally, users have the option to 4a) substitute the original codes proposed by two coders and revert back to the original codes if required. They can also replace or revert all code decisions with a single click on the top bar.

To ensure GPT provides results in a consistent format, we use a one-shot prompt for the return result format: *Here is the format of results: Version1: Version2: Version3:*, accompanied by our specific requirements: *Requirements: 6 words or fewer; No duplicate words; Be general; Three distinct versions.*

Generate Code Groups. To facilitate the creation of primary groups, we prompt GPT with *Organize the following codes into 5 thematic groups without altering the original codes, and name each group: 1. [Code], 2. [Code]...* To ensure the returned results are consistent, we provide GPT with an example format: *Here is the format of the results: Group1: [theme], 1.[code], 2.[code], 3.[code].*

4.3.4 Semantic Similarity and IRR. In DiffCoder, the IRR is measured using Cohen's Kappa[11] and Agreement Rate. To calculate Cohen's Kappa, we used the "cohen_kappa_score" method from scikit-learn package backend[12]. Cohen's Kappa is a score between -1 (total disagreement) and +1 (total agreement). Subsequently, we calculate the Agreement Rate as a score between 0 and 1, by determining the percentage of code pairs whose similarity score exceeds 0.8, indicating that the two coders agree on the code segment. We employ the *sentence-transformers* package[13] to determine the semantic similarity between pairs of code from two coders.

5 USER EVALUATION

To evaluate the effectiveness of DiffCoder, we carried out a within-subject user study involving 16 participants who used two platforms: DiffCoder and Atlas.ti Web, for qualitative coding on two sets of qualitative data. The goal was to address the following research questions:

- **RQ1.** Can DiffCoder support qualitative coders conduct CQA effectively?

[11] Cohen's Kappa is a statistical measure used to evaluate the IRR between two or more raters, which takes into account the possibility of agreement occurring by chance, thus providing a more accurate representation of agreement than simply calculating the percentage of agreement between the raters.

[12] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html

[13] https://www.sbert.net/

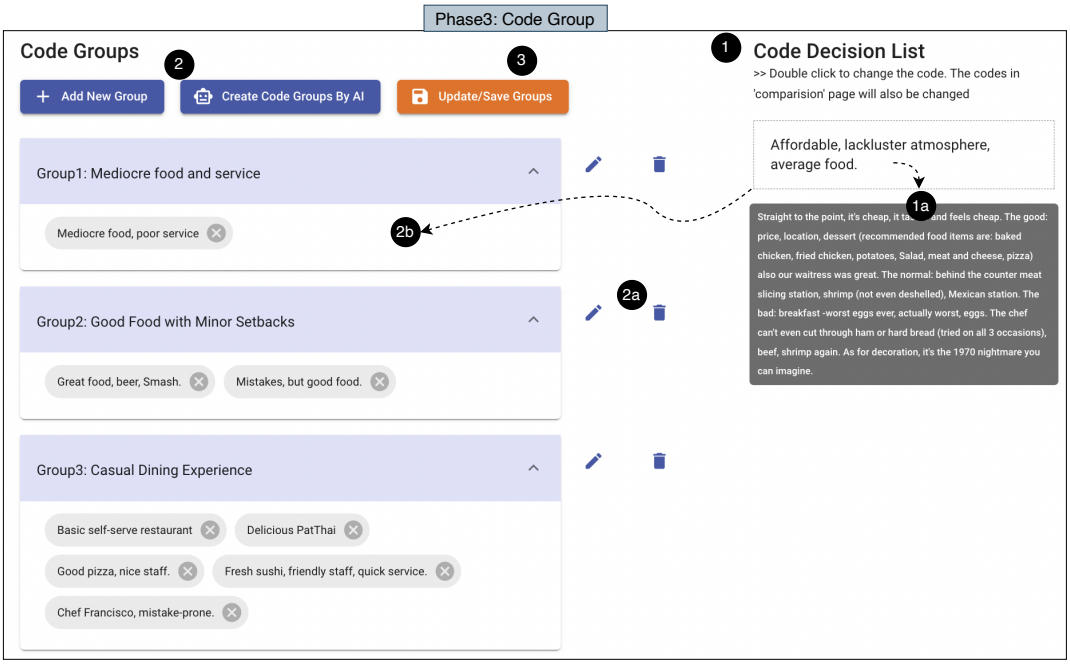

Figure 7: Code Group Interface. It enables users to manage their code decisions in a few steps: 1) the code decisions are automatically compiled into a list of unique codes that users can edit by double-clicking and accessing the original data by hovering over the code. 2) Users can group their code decisions by using either "Add New Group" or "Create Code Groups By AI" options. They can then 2a) name or delete a code group or use AI-generated themes, and 2b) drag the code decisions into code groups. 3) Finally, users can save and update the code groups.

- **RQ2.** How does DiffCoder compare to currently available tools like Atlas.ti Web?
- **RQ3.** How can the design of DiffCoder be improved?

5.1 Participants and Ethics

We enlisted 16 participants via public channels and university email lists. Out of these participants, 14/16 indicated that English was their first language, while 2 reported having a proficient level of English. 9/16 participants had at least 1-3 instances of prior qualitative experience, while 7/16 had no previous qualitative experience. Two participants identified as experts in QA, while three considered themselves intermediate, and four as beginners (see Appendix Table 4). Participants, who had diverse levels of QA expertise, were randomly matched, leading to the formation of 8 pairs (see Appendix Table 5). Each participant received compensation of approximately $22.3 USD for their participation, based on the total duration. The study protocol was approved by our local IRB, and the financial compensation was based on the duration at the hourly rate also approved by our local IRB.

5.2 Datasets

We established two criteria to select the datasets used for participants: 1) the datasets should not require domain-specific knowledge for coding, and 2) coders should be able to derive a theme tree and provide insights iteratively. Accordingly, we selected two datasets containing book reviews on "Business" and "History" topics from the `Books_v1_00` category of `amazon_us_reviews` dataset[14]. For each of these datasets, we filtered 15 reviews to include only those with a character count between 400 and 700 and removed odd symbols such as \ and
, and these were provided to the coders for coding. The workload was pilot-tested and determined by our research team.

5.3 Conditions

- **Atlas.ti Web**: a powerful platform for qualitative analysis that enables users to invite other coders to collaborate by adding, editing, and deleting codes. It also allows for merging codes and generating code groups manually.
- **DiffCoder platform**: the final version of our full-featured platform.

The order of presentation of both platforms was fully counterbalanced across participants, and the data set was also randomized to make sure that the pairs would not code the same data set with the same platform (see Appendix Table 5).

5.4 Procedure

Each study was conducted virtually via Zoom and lasted around 2 to 3 hours. It consisted of a pre-study questionnaire, training for novice

[14]https://huggingface.co/datasets/amazon_us_reviews/viewer/Books_v1_00/train

participants, two qualitative coding sessions with different conditional systems, a post-study questionnaire, and a semi-structured interview.

5.4.1 Introduction to the Task. After obtaining consent, we introduced the task to the pairs of participants, which involved analyzing reviews and coding them to obtain meaningful insights. We introduced research questions they should take into account when coding, such as recurring themes or topics, common positive and negative comments or opinions. We provided guidelines to ensure that the coding was consistent across all participants. Participants were permitted to use codes that were under 10 words in length, include multiple codes for each data unit, and add both descriptive and in-vivo codes.

5.4.2 Specific Process. Following the introduction, we provided a video tutorial on how to use the platform for qualitative coding. Participants first did independent coding, and then discussed the codes they had found and made final decisions for each unit, ultimately forming thematic groups. We urge them to engage in extensive discussions and to present code groups that accurately reflect the valuable insights they have acquired, emphasizing the importance of quality. To ensure they understand the study purpose better, participants were shown sample code groups as a reference for the type of insights they should aim to obtain from their coding. After completing the coding for all sessions, participants were asked to complete a survey, which included a 5-level Likert Scale to rate the effectiveness of two platforms, and self-reported feelings about the platforms.

5.4.3 Data Recording. During the process, we asked participants to share their screens and obtained their consent to record the meeting video for the entire experiment. Once the coding sessions were completed, participants were invited to participate in a post-study semi-structured interview.

6 RESULTS

6.1 Quantitative Results

6.1.1 Post-study questionnaire. We gathered the subjective preferences from our participants. To do so, we gave them 12 statements like "I find it effective to..." and "I feel confident/prefer..." pertaining to the effectiveness and self-perception. We then asked them to rate their agreement with each sentence on a 5-point Likert scale for each platform. The details of the 12 statements are shown in Figure 8.

Overall, pairwise t-tests showed that participants rated DiffCoder significantly (all $p < .05$) better than Atlas.ti Web for effectiveness in 1) coming up with codes, 2) producing final code groups, 3) identifying disagreements, 4) resolving disagreements and making decisions, 5) understanding the current level of agreement, and 6) understanding others' thoughts. The results also indicated that participants believed DiffCoder could be learned for use rapidly ($t(15) = -3.05, p < .01$). For other dimensions, the confidence in the final quality, perceived level of preference, level of control, level of understanding, and ease of use, while our results show a general trend where DiffCoder achieves higher scores, we found no significant differences. Additionally, we observed that one expert user (P6) exhibited a highly negative attitude towards implementing AI in qualitative coding, as he selected "strongly disagree" for nearly all assessment criteria.

6.1.2 Log Data Analysis. A two-tailed pairwise t-test on Discussion Time revealed a significant difference ($t(15) = -3.22, p = .017$) between DiffCoder ($M = 24:10 mins, SD = 7:12 mins$) and Atlas.ti ($M = 10:48 mins, SD = 5:24 mins$). These results indicate that the DiffCoder condition led to significantly longer discussions compared to the Atlas.ti Web condition. When examining the IRR, it was found that the IRRs in the Atlas.ti Web condition were overall significantly ($t(7) = -6.69, p < .001$) lower ($M = 0.06, SD = 0.40$), compared to the DiffCoder condition ($M \approx 1$). In this case, participants thoroughly examined all codes, resolved conflicts, merged similar codes, and reached a final decision for each data unit. Conversely, Atlas.ti Web posed challenges in comparing individual data units side-by-side, leading to minimal code discussions overall (averaging 4 codes discussed) compared to the DiffCoder option (averaging 15 codes discussed). Hidden disagreements within Atlas.ti Web would thus necessitate further discussion rounds to attain a higher IRR level.

6.2 Qualitative Results

6.2.1 Data analysis. We analyzed interview transcripts and observation notes using thematic analysis as described in Braun and Clarke's methodology [6]. After familiarizing ourselves with data and generating initial codes, we grouped the transcripts into common themes based on the content. Next, we discussed, interpreted, and resolved discrepancies or conflicts during the grouping process. Finally, we reviewed the transcripts and audio recordings to extract specific quotes relevant to each theme. We summarized the following key findings.

6.2.2 Key Findings (KF).

KF1: GPT support is valued for reducing cognitive burden during independent coding, yet there remains room for improvement. DiffCoder makes it easier for beginner users to apply codes and edit them compared to Atlas.ti Web. 7/16 participants appreciated that GPT's additional assistance (P7, P15), which gave them reference (P1) and decreased thinking (P9). Most of them are beginners (except an intermediate user, P9) or have no prior QA experience. As P13 said, *"I think the DiffCoder one is definitely more intuitive in a sense, because it provides some suggestion, you might not use it, but at least some basic suggestion, whereas the Atlas one, you have to take from scratch and it takes more mental load."* (P13).

Some participants showed displeasure towards GPT, largely stems from its content summarization level, which users cannot regulate. P1 (beginner) found that **in certain instances, DiffCoder generated highly detailed summaries**, such summaries might not be well-suited to their requirements, leading them to prefer crafting their own summaries: *"One is that its summary will be very detailed, and in this case, I might not use its result, but I would try to summarize [the summary] myself."* This caused them to question AI's precision and appropriateness for high-level analysis, especially in the context of oral interviews or focus groups.

In addition, when adding codes, our participants indicated that they preferred **reading the raw data first before looking at the**

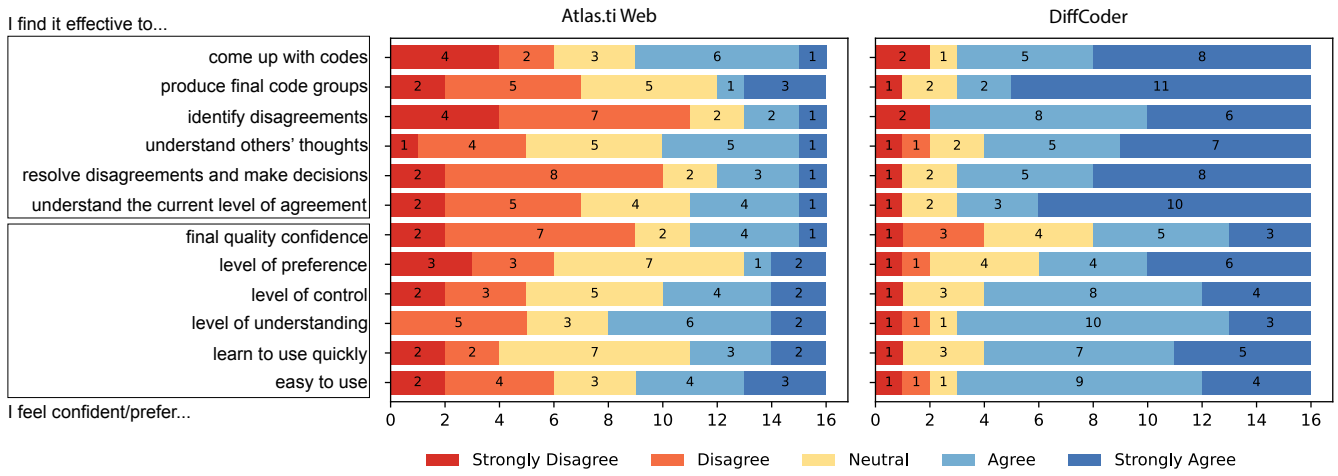

Figure 8: Post-study Questionnaires Responses from Our Participants on Different Dimensions on A 5-point Likert Scale, where 1 denotes "Strongly Disagree", 5 denotes "Strongly Agree".

suggestions from the AI Agent, as they believed that reading the suggestions first could influence their thinking process (P1, P3, P4, P14) and introduce bias into their coding: *"So I read the text at first. it makes more sense, because like, if you were to solely base your coding on [the AI agent], sometimes its suggestions and my interpretation are different. So it might be a bit off, whereas if you were to read the text, you get the full idea as to what the review is actually talking about. The suggestion functions as a confirmation of my understanding."* (P4)

KF2: DiffCoder can enhance mutual understanding, identify disagreements, and initiate discussions. Regarding collaboration, participants appreciated Diffcoder's keyword addition functionality, as it aided them in capturing finer details (P9) and facilitated a better understanding of the coding process: *"It presents a clearer view about that paragraph. And then it helps us to [I think] get a better idea of what the actual correct code should be. But since the other one [Atlas.ti Web] is [...] a little bit more like superficial, because it's based solely on two descriptive words."* (P14)

This understanding was seen as particularly helpful during discussions, as it mitigated the potential for forgetting or requiring additional time to remember the decision-making process, which might otherwise hinder conversation. Additionally, users discovered that having **a pre-defined unit of analysis enabled them to more easily align their understanding**, thus eliminating the extra step of examining for quotations: *"I am able to see your quotations, and the definition. Basically what they coded is just the entire unit. But you see if they were to code the reviews based on sentence, I wouldn't actually do the hard work based on like which sentence did he highlighted. But for DiffCoder, I am able to see at a glance, the exact quotations that they did. So it gives me a better sense of how their codes came about."* (P3) Moreover, users emphasized the importance of not only having the quotation but also its context, as they often preferred to refer back to the original text. This is because understanding the context is crucial for accurate data interpretation and discussion: *"I guess, is because like we're used to reading a full text and we know like the context rather than if we were to read like short extracts from the text. the context is not fully there from just one or two line [quotations]."* (P9)

In addition, users find DiffCoder to be better as it supports side-by-side comparison of scores and data, which makes it **easier to understand (P2), more straightforward (P7) and beginner-friendly (P4)** than Atlas.ti Web, while P4 noted that DiffCoder had a **lower learning curve**.

In terms of statistics during the collaboration, the similarity calculation and ranking features enable users to **quickly identify similarities and disagreements (P2, P3, P7, P10, P14) to ensure they focus more (P4).** As P14 said, *"I think it's definitely a good thing [to calculate similarity]. From there, I think we can decide whether it's really a disagreement on whether it's actually two different information captured in the two different codes. About if it's a disagreement, then it helps to use the rating to like achieve higher IRR."* Moreover, the ranking of similarity is reported to **pave the way for discussion (P1, P8)**: *"So I think in that sense, it just opens up the door for the discussion compared to Atlas.ti...because honestly i'm so tired right now. And P7 didn't say anything. I think in that sense... I guess, better in that idea generation stands and opening up the door for discussion."* (P8) In contrast, Atlas.ti necessitated more discussion initiation on the part of users.

Nevertheless, ranking similarity might have a negative effect, as it may restrict its original purpose which is to provide a more comprehensive interpretation for the data, making coders focus more on improving their agreements: *"I think pros and cons. because you will feel like there's a need to get high similarity on every code, but it might just be different codes. So there might be a misinterpretation."* (P7)

The progress bar feature in DiffCoder was seen as helpful when collaborating with others. It allowed them to **manage their time better and track the progress of each coder.** *"I actually like the progress bar because like that I know where my collaborators are."* (P8) Additionally, it acted as a **tracker to notify the user if they**

missed out on a part, which can help to avoid errors and improve the quality of coding. *"So if say, for example, I missed out one of the codes then or say his percentage is at 95% or something like that, then we will know that we missed out some parts"* (P3)

The participants had mixed opinions regarding the usefulness of IRR in the coding process. P9 found Cohen's kappa useful for their report as they do not need to calculate manually: *"I think it's good to have the Cohen's Kappa, because we don't have to manually calculate it, and it is very important for our report. "*. However, P6 did not consider the statistics to be crucial in their personal research as they usually do coding for interview transcripts. *"Honestly, it doesn't really matter to me because in my own personal research, we don't really calculate. Even if we have disagreements, we just solve it out. So I can't comment on whether the statistics are relevant, right from my own personal experience."* (P6)

KF3: Users can conduct a top-down approach to understand the current code trees, decreasing the cognitive burden of creating groups. Participants expressed a preference for the automatic grouping function of DiffCoder, as it was **more efficient (P1, P2, P8, P14) and less labor-intensive (P3),** compared to the more manual approach in Atlas.ti Web. P1 also highlighted this is particularly helpful when dealing with large amounts of codes, as manually grouping them one-by-one becomes nearly unfeasible. P14 characterized the primary distinction between the two platforms as Atlas.ti Web adopting a "bottom-up approach," while **DiffCoder employs a "top-down approach.**" This difference impacts the mental effort required to create categories and organize codes, as it creates "overall categories" first, allowing users to edit and shift things starting from a foundation instead of from scratch. *"I think it's different also because for Atlas.ti is more like a bottom-top approach. So we need to see through the primary codes to create the larger categories which might be a bit more tedious. because usually they are the primary codes. So it's very hard to see an overview of everything at once. So it takes a lot of mental effort, but for DiffCoder, it is like a top down approach. So they create the overall categories. And then from there you can edit and then like will shift things around which helps a lot. So i also prefer DiffCoder."* (P14)

Attitudes towards other functions like labeling certainty, relevant codes suggestions, and individual codebook. Most participants expressed concerns about the clarity, usefulness, and importance of the certainty function in DiffCoder. The self-reported nature of the function, the potential for inconsistencies, and minimal usage among users suggest that the **certainty function may not be as helpful as intended**. P12 found the certainty function "not really helpful," and P13 admitted to forgetting about it due to the numerous other tasks involved. P3 also reported limited use of the function, mainly assigning low certainty scores when not understanding the raw data. However, P14 recognized that the certainty function could be helpful in larger teams, as it might help flag quotes that require more discussion.

The perceived usefulness of the function of the relevant code in DiffCoder depends on the dataset and users' preferences. Some participants found it **less relevant than the AI agent's summary function**, which they considered more accurate and relevant. *"Maybe not that useful, but I think it depends on your dataset. Say whether they are many similar data points or whether they are different data points. So I think in terms of these cases they are all very different, have a lot of different contents. So it's not very relevant, but definitely, I think, in datasets which might be more relevant, could be useful."* (P2)

As for the individual codebook function, although users acknowledged its potential usefulness in tracking progress and handling large datasets, most users "did not pay much attention to it during this coding process" (P2, P3, P4). P3 mentioned finding it helpful for tracking progress but did not pay attention to it during this entire process. P4 acknowledged that the function could be useful in the long run, particularly when dealing with a large amount of data. Further exploration of these functions and their potential benefits may be necessary to improve users' experiences with DiffCoder.

7 DISCUSSION AND DESIGN IMPLICATIONS

The results from our study highlight how collaboration is a key aspect of qualitative analysis, and that AI has the potential to significantly improve the efficiency of this process. In this discussion, we will specifically explore the role of LLMs' assistance (e.g., GPT) in various stages of CQA, namely, (1) the independent open coding phase, (2) the discussion phase, and (3) the final code group creation. We also discuss the importance of thoughtful interface design considerations for fostering more efficient discussions on CQA platforms.

7.1 LLMs as Effective "Suggestion Provider" in Open Coding: Helper, not Replacement.

During the open coding phase, GPT can effectively function as a suggestion provider. Our study demonstrated that both novices and experts valued GPT's assistance, as our participants used GPT's suggestions either as code or as a basis to create codes 76.67% of the time on average.

7.1.1 Utilizing LLMs to Reduce Cognitive Burden. Independent open coding is a highly cognitively demanding task, as it requires understanding the text, identifying the main idea, creating a summary based on research questions, and formulating a suitable phrase to convey the summary [31, 47]. Additionally, there is the need to refer to and reuse previously created codes. In this context, GPT's text comprehension and generation capabilities can assist in this mentally challenging process by serving as a suggestion provider. In comparison, traditional platforms without AI support are predominantly manual, offering limited advantages over tools like Google Docs or Microsoft Word.

7.1.2 Improving LLMs' Suggestions Quality. However, a key consideration according to KF1 is how GPT can provide better quality suggestions that align with the needs of users. For DiffCoder, we only provided essential prompts such as "summary" and "relevant codes". However, a crucial aspect of qualitative coding is that coders should always consider their research questions while coding and work towards a specific direction. For instance, are they analyzing the main sentiment of the raw data or the primary content or opinion? These factors can significantly impact the coding approach (e.g., descriptive or in-vivo coding) and what should be coded (e.g., sentiment or opinions). Therefore, the system should support mechanisms for users to inform GPT of the user's intent or direction. One

possible solution is to include the research question or intended direction in the prompt sent to GPT alongside the data to be coded. Alternatively, users could configure a customized prompt for guidance, directing GPT's behavior through the interface [26]. This adaptability accommodates individual preferences and improves the overall user experience.

7.1.3 LLMs should Remain a Helper. Another key consideration is how GPT can stay a reliable suggestion provider without taking over from the coder. One expert user (P6) held a negative attitude towards employing LLMs in open coding, assigning the lowest score to nearly all measures (see Figure 8). This user expressed concerns about the role of AI in this context, suggesting that qualitative researchers might feel forced to use AI-generated codes, which could introduce potential biases. Picking up the nuances from the text is considered "fun" for qualitative researchers (P6), and suggestions should not give the impression that "the code is done for them and they just have to apply it" (P6) or lead them to "doubt their own ideas" (P5). On the other side, it is important not to overlook the risk of over-reliance on GPT. While we want GPT to provide assistance, we do not intend for it to fully replace humans in the process. Our observations revealed that although participants claimed they would read the raw data first and then check GPT's suggestions, some beginners tended to rely on GPT for forming their suggestions, and experts would unconsciously accept GPT's suggestions if unsure about the meaning of the raw data, in order to save time. Therefore, preserving the enjoyment of qualitative research and designing for appropriate reliance [32] to avoid misuse [16] or over-trust can be a complex challenge [48]. To this end, mixed-initiative systems [1, 25] can be designed to allow for different levels of automation. For example, GPT-generated suggestions could be provided only for especially difficult cases upon request, rather than being easily accessible for every unit, even when including a pre-defined time delay.

7.2 LLMs as "Mediator" for Different Group Dynamics in the Discussion Phase

During the experiment, we observed various intriguing collaboration dynamics, such as "follower-leader" (P1×P2, P5×P6), "amicable cooperation" (P3×P4, P7×P8, P9×P10, P13×P14, P15×P16), and "swift but less cautious" (P11×P12). We now discuss the use of LLMs as a group recommendation system [17] to facilitate decision-making during group discussions. Our aim is to help groups make decisions that are cost-effective, appropriate, justifiable, and fair [8].

7.2.1 Mediation under an "Amicable Cooperation" Dynamic. For "amicable cooperation", the coders respected each other's opinions. When they make a decision, they firstly identify the common keywords between their codes, and then check the suggestions with similar keywords to decide whether to use suggestions or propose their own. They often took turns applying the final code. For example, for the first data unit, the coder might say, *"hey, mine seems better, let's use mine as the final decision,"* and for the second one, they might say, *"hey, I like yours, we should choose yours [as the final decision]"* (P3×P4). In some cases, such as P13×P14, both coders generally reach a consensus, displaying no strong dominance and showing respect for each other's opinions, sometimes it can be difficult to reach a final decision. To address this, the coders used an LLMs agent as a mediator to find a more suitable expression that takes into account both viewpoints.

7.2.2 Mediation under a "Follower-Leader" Dynamic. The follower-leader pattern typically occurred when one coder was a novice, while the other had more expertise. Often, the inexperienced coder contributed fewer ideas or only offered support during the coding process: when using Atlas.ti Web, we noticed that since their coding tasks could not be precisely quantified, those "lead" coders tended to take on more coding tasks than the other. Even though both of them were told to do code all the data, it would end up in a situation where one coder primarily handled the work and the other merely followed with minimal input. This pattern could also appear if the coders worked at different paces (P1×P2, P3×P4). As a result, the more efficient coders expressed more ideas. With DiffCoder, the workload was more equitably distributed, and both coders participated in creating codes for every data unit. DiffCoder also allowed for a clear expression and documentation of the decision-making process. This approach ensures that the coder with fewer opportunities to express their ideas can still utilize quantitative metrics to indirectly express their ideas and be compared with their collaborator, as highlighted by KF2. Subsequently, GPT will generate suggestions based on both codes and raw data, taking into account the input from both collaborators.

7.2.3 Mediation under a "Swift but Less Cautious" Dynamic. The swift but less cautious collaboration was a less desirable pattern we noticed: For P11×P12, during the merging process, they would heavily rely on GPT-generated decisions in order to finish the task quickly. This scenario highlights the concerns regarding excessive reliance on GPT and insufficient deep thinking, which can negatively impact the final quality even when GPT is used as a mediator after the codes have been produced, as defined as our initial objective. Under this pattern, the pair sadly used GPT for "another round of coding" rather than as a neutral third-party decision advice provider.

To sum up, GPT can act as a mediator between coding teams when a consensus cannot be reached or serve as a third party to balance the division of labor and expression of opinions. However, to enhance our system and ensure that humans remain the ultimate decision-makers while using GPT as an aid in clarifying specific differences between coders, we recommend that suggestions be displayed only upon request during the discussion phase. Alternatively, after a coder proposes a final decision, GPT could refine the final expression or generate a definitive description for the code to aid in future reflection. This approach ensures that humans remain central in the decision-making process while benefiting from GPT's capabilities for potentially improved understanding and communication.

7.3 LLMs as "Facilitator" in Streamlining Primary Code Grouping

7.3.1 Participants's Appreciation for Automatic Grouping. As per KF3, Our participants offered insightful feedback about using GPT

to generate primary code groups. They found the top-down approach, where GPT first generates primary groups and users subsequently refine and revise them, more efficient and less cognitively demanding compared to the traditional bottom-up method. In the bottom-up method, users must begin by examining all primary codes, merging them, and then manually grouping them into categories, which can be mentally taxing. Users can then focus on making adjustments and revisions to the generated groups, ensuring a more accurate and relevant categorization. Through this, researchers can more effectively and easily manage large volumes of data and potentially enhance the quality of their text analysis.

7.3.2 Challenges and Solutions of Automatic Grouping. However, it is crucial to exercise caution when applying this method. We observed that when time constraints exist, coders may skip discussions, with only one of two coders combining and categorizing the codes into code groups (P7×P8). Additionally, P14 mentioned that GPT appears to dominate the code grouping process, resulting in a single approach to grouping. For instance, while the participants might create code groups based on sentiment analysis during their own coding process, they could be tempted to focus on content analysis under GPT's guidance.

To overcome these challenges, we envision a system where coders would create their own groupings first and only request LLMs' suggestions afterward. Alternatively, LLMs' assistance could be limited to situations where the data volume is substantial. Another approach could be prompting LLMs to generate code groups based on the research questions rather than solely on the (superficial) codes. This would ensure a more contextually relevant and research-driven code grouping process.

7.4 Facilitating Discussion by Building Common Ground During Discussion Phase

7.4.1 On the Importance of Documenting the Decision-Making Process. KF2 suggests that our DiffCoder prototype facilitated enhanced mutual understanding among its users through the creation of a shared space where the coding decision-making process could be shared and all necessary information could be organized on a single page. This information included original data, codes and certainty from two coders, and supported keywords and quotations. The observation made aligns with the principles of grounding in communication theory and transparent communication. According to the principles of grounded theory [11], establishing a common ground through open and honest communication promotes mutual understanding and trust, leading to better collaboration and decision-making. Similarly, transparent communication, as described by Yue et al. [49], emphasizes the importance of open and clear communication that enables individuals to share information, ideas, and feelings without fear of reprisal, leading to more effective collaboration and positive outcomes.

7.4.2 Identifying Agreements through Similarity Scores to kick-off Discussion. In addition, by quantitatively identifying agreements and disagreements through the similarity metric, users could filter codes requiring more discussion from those that did not. Similar codes with a high agreement level of were seen a more accessible entry point: by merging agreements first, then gradually addressing disagreements, which necessitated more debate, critical thinking, and reflection on the original codes. The provided statistics also offer a better understanding of the current level of agreement between coders, proving especially beneficial during discussions.

7.4.3 Discussion Quality across Platforms. In contrast, the current platform requires users to click back and forth to search for information needed for codes, meaning, and context under descriptive codes, and to identify agreements and disagreements, which demands significant human effort and time. This approach is prone to errors and may cause further disagreements if a code is refined for a given unit, but still attached to other units. Moreover, team members need to be available simultaneously and may struggle to recall the reasoning behind their code choices, especially when working with lengthy and complex data. As a result, they may only discuss a limited number of codes, conduct superficial discussions (averaging around 4.5 discussed codes in Atlas.ti Web session vs. all 15 codes discussed in DiffCoder sessions), or even abandon discussions if time doesn't allow (as P7×P8 did). Although our data suggests that discussion time was overall lower for Atlas.ti Web (≈ 0.18 hours in Atlas.ti Web vs. ≈ 0.4 hours in DiffCoder), coders using DiffCoder discussed more codes and reached a higher IRRs (≈ 0.06 for Atlas.ti Web vs. virtually 1 for DiffCoder). As such, users would require more rounds of discussion on Atlas.ti to reach a better final consensus.

8 LIMITATIONS AND FUTURE WORK

This study has limitations. Firstly, we only used pre-defined unit data and did not consider splitting complex data into units (e.g., interview data). Future work could explore utilizing GPT to support the segmentation of interview data into semantic units and automating the import process. Secondly, we did not investigate the specific process by which users select and edit a GPT suggestion. Instead, we used a generic term, "Derived from GPT" to represent both the user-selected and edited codes. Future research could delve deeper into how users incorporate these suggestions to generate a final idea. Moreover, for a tool that could be used by the same coder on multiple large datasets, it would also be beneficial to have GPT generate suggestions based on users' coding patterns rather than directly providing suggestions. Finally, the expert interviews highlighted that addressing the issue of multiple codes for a single unit is crucial during the collaborative coding process, as it frequently initiates discussions. Future research should consider tackling this problem as well.

9 CONCLUSION

This paper introduces DiffCoder, a system that can support users through multiple phases of qualitative coding through the use of LLMs. Our evaluation with 16 participants indicated a preference for DiffCoder over existing platforms like Atlas.ti Web due to its user-friendly design and AI assistance tailored for collaboration. We also demonstrated the system's capability to streamline consensus-building, facilitate discussions, and create codebooks using a top-down approach. Our research has demonstrated the potential of LLMs in enhancing the effectiveness of consensus-building and codebook creation. By examining both human-AI and

human-human interactions within the context of qualitative analysis, we have uncovered key challenges and insights that can guide future research and development in this area.

REFERENCES

[1] James E Allen, Curry I Guinn, and Eric Horvtz. 1999. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications* 14, 5 (1999), 14–23.

[2] Ross C Anderson, Meg Guerreiro, and Joanna Smith. 2016. Are all biases bad? Collaborative grounded theory in developmental evaluation of education policy. *Journal of Multidisciplinary Evaluation* 12, 27 (2016), 44–57.

[3] David Armstrong, Ann Gosling, John Weinman, and Theresa Marteau. 1997. The place of inter-rater reliability in qualitative research: An empirical study. *Sociology* 31, 3 (1997), 597–606.

[4] Steve Benford, John Bowers, Lennart E Fahlén, and Chris Greenhalgh. 1994. Managing mutual awareness in collaborative virtual environments. In *Virtual Reality Software and Technology*. World Scientific, 223–236.

[5] Pernille Bjørn, Morten Esbensen, Rasmus Eskild Jensen, and Stina Matthiesen. 2014. Does distance still matter? Revisiting the CSCW fundamentals on distributed collaboration. *ACM Transactions on Computer-Human Interaction (TOCHI)* 21, 5 (2014), 1–26.

[6] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[7] Scott Brave, Hiroshi Ishii, and Andrew Dahley. 1998. Tangible interfaces for remote collaboration and communication. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. 169–178.

[8] Li Chen, Marco De Gemmis, Alexander Felfernig, Pasquale Lops, Francesco Ricci, and Giovanni Semeraro. 2013. Human decision making and recommender systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 3, 3 (2013), 1–7.

[9] Nan-Chen Chen, Margaret Drouhard, Rafal Kocielnik, Jina Suh, and Cecilia R Aragon. 2018. Using machine learning to support qualitative coding in social science: Shifting the focus to ambiguity. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 8, 2 (2018), 1–20.

[10] Bonnie Chinh, Himanshu Zade, Abbas Ganji, and Cecilia Aragon. 2019. Ways of qualitative coding: A case study of four strategies for resolving disagreements. In *Extended abstracts of the 2019 CHI conference on human factors in computing systems*. 1–6.

[11] Herbert H Clark and Susan E Brennan. 1991. Grounding in communication. (1991).

[12] Juliet M Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21.

[13] Flora Cornish, Alex Gillespie, and Tania Zittoun. 2013. Collaborative analysis of qualitative data. *The SAGE handbook of qualitative data analysis* 79 (2013), 93.

[14] Jessica Díaz, Jorge Pérez, Carolina Gallardo, and Ángel González-Prieto. 2023. Applying Inter-Rater Reliability and Agreement in collaborative Grounded Theory studies in software engineering. *Journal of Systems and Software* 195 (2023), 111520.

[15] Margaret Drouhard, Nan-Chen Chen, Jina Suh, Rafal Kocielnik, Vanessa Pena-Araya, Keting Cen, Xiangyi Zheng, and Cecilia R Aragon. 2017. Aeonium: Visual analytics to support collaborative qualitative coding. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 220–229.

[16] Mary T Dzindolet, Scott A Peterson, Regina A Pomranky, Linda G Pierce, and Hall P Beck. 2003. The role of trust in automation reliance. *International journal of human-computer studies* 58, 6 (2003), 697–718.

[17] Hanif Emamgholizadeh. 2022. Supporting group decision-making processes based on group dynamics. In *Proceedings of the 30th ACM Conference on User Modeling, Adaptation and Personalization*. 346–350.

[18] Jessica L Feuston and Jed R Brubaker. 2021. Putting Tools in Their Place: The Role of Time and Perspective in Human-AI Collaboration for Qualitative Analysis. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–25.

[19] Abbas Ganji, Mania Orand, and David W McDonald. 2018. Ease on Down the Code: Complex Collaborative Qualitative Coding Simplified with'Code Wizard'. *Proceedings of the ACM on human-computer interaction* 2, CSCW (2018), 1–24.

[20] Simret Araya Gebreegziabher, Zheng Zhang, Xiaohang Tang, Yihao Meng, Elena Glassman, and Toby Jia-Jun Li. 2023. PaTAT: Human-AI Collaborative Qalitative Coding with Explainable Interactive Rule Synthesis. *corpora* 7, 27 (2023), 52.

[21] Linda S. Gilbert, Kristi Jackson, and Silvana di Gregorio. 2014. *Tools for Analyzing Qualitative Data: The History and Relevance of Qualitative Data Analysis Software*. Springer New York, New York, NY, 221–236. https://doi.org/10.1007/978-1-4614-3185-5_18

[22] Barney G Glaser and Anselm L Strauss. 2017. *The discovery of grounded theory: Strategies for qualitative research*. Routledge.

[23] Julian Hocker, Taryn Bipat, Mark Zachry, and David W McDonald. 2020. Sharing your coding schemas: Developing a platform to fit within the qualitative research workflow. In *Proceedings of the 16th International Symposium on Open Collaboration*. 1–10.

[24] Matt-Heun Hong, Lauren A Marsh, Jessica L Feuston, Janet Ruppert, Jed R Brubaker, and Danielle Albers Szafir. 2022. Scholastic: Graphical Human-AI Collaboration for Inductive and Interpretive Text Analysis. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–12.

[25] Eric Horvitz. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 159–166.

[26] Daphne Ippolito, Ann Yuan, Andy Coenen, and Sehmon Burnam. 2022. Creative Writing with an AI-Powered Writing Assistant: Perspectives from Professional Writers. *arXiv preprint arXiv:2211.05030* (2022).

[27] Anthony Jameson, Stephan Baldes, and Thomas Kleinbauer. 2003. Enhancing mutual awareness in group recommender systems. In *Proceedings of the IJCAI*, Vol. 10.

[28] Jialun Aaron Jiang, Kandrea Wade, Casey Fiesler, and Jed R Brubaker. 2021. Supporting serendipity: Opportunities and challenges for Human-AI Collaboration in qualitative analysis. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW1 (2021), 1–23.

[29] Klaus Krippendorff. 2019. Content Analysis: An Introduction to Its Methodology. https://doi.org/10.4135/9781071878781

[30] Karen S Kurasaki. 2000. Intercoder reliability for validating conclusions drawn from open-ended interview data. *Field methods* 12, 3 (2000), 179–194.

[31] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. *Research methods in human-computer interaction*. Morgan Kaufmann.

[32] John D Lee and Katrina A See. 2004. Trust in automation: Designing for appropriate reliance. *Human factors* 46, 1 (2004), 50–80.

[33] Robert P Lennon, Robbie Fraleigh, Lauren J Van Scoy, Aparna Keshaviah, Xindi C Hu, Bethany L Snyder, Erin L Miller, William A Calo, Aleksandra E Zgierska, and Christopher Griffin. 2021. Developing and testing an automated qualitative assistant (AQUA) to support qualitative analysis. *Family Medicine and Community Health* 9, Suppl 1 (2021). https://doi.org/10.1136/fmch-2021-001287

[34] Megh Marathe and Kentaro Toyama. 2018. Semi-Automated Coding for Qualitative Research: A User-Centered Inquiry and Initial Prototypes. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173922

[35] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and inter-rater reliability in qualitative research: Norms and guidelines for CSCW and HCI practice. *Proceedings of the ACM on human-computer interaction* 3, CSCW (2019), 1–23.

[36] Slava Mikhaylov, Michael Laver, and Kenneth R. Benoit. 2012. Coder Reliability and Misclassification in the Human Coding of Party Manifestos. *Political Analysis* 20, 1 (2012), 78–91. https://doi.org/10.1093/pan/mpr047

[37] Michael Muller, Shion Guha, Eric P.S. Baumer, David Mimno, and N. Sadat Shami. 2016. Machine Learning and Grounded Theory Method: Convergence, Divergence, and Combination. In *Proceedings of the 2016 ACM International Conference on Supporting Group Work* (Sanibel Island, Florida, USA) *(GROUP '16)*. Association for Computing Machinery, New York, NY, USA, 3–8. https://doi.org/10.1145/2957276.2957280

[38] Alireza Nili, Mary Tate, Alistair Barros, and David Johnstone. 2020. An approach for selecting and using a method of inter-coder reliability in information management research. *International Journal of Information Management* 54 (2020), 102154.

[39] Gary M Olson and Judith S Olson. 2000. Distance matters. *Human–computer interaction* 15, 2-3 (2000), 139–178.

[40] Cliodhna O'Connor and Helene Joffe. 2020. Intercoder reliability in qualitative research: debates and practical guidelines. *International journal of qualitative methods* 19 (2020), 1609406919899220.

[41] Pablo Paredes, Ana Rufino Ferreira, Cory Schillaci, Gene Yoo, Pierre Karashchuk, Dennis Xing, Coye Cheshire, and John Canny. 2017. Inquire: Large-Scale Early Insight Discovery for Qualitative Research. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing* (Portland, Oregon, USA) *(CSCW '17)*. Association for Computing Machinery, New York, NY, USA, 1562–1575. https://doi.org/10.1145/2998181.2998363

[42] Harshada Patel, Michael Pettitt, and John R Wilson. 2012. Factors of collaborative working: A framework for a collaboration model. *Applied ergonomics* 43, 1 (2012), 1–26.

[43] Elin Rønby Pedersen and Tomas Sokoler. 1997. AROMA: abstract representation of presence supporting mutual awareness. In *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. 51–58.

[44] K Andrew R Richards and Michael A Hemphill. 2018. A practical guide to collaborative qualitative data analysis. *Journal of Teaching in Physical education* 37, 2 (2018), 225–231.

[45] Tim Rietz and Alexander Maedche. 2021. Cody: An AI-Based System to Semi-Automate Coding for Qualitative Research. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 394, 14 pages. https://doi.org/10.1145/3411764.3445591

[46] Johnny Saldaña. 2021. The coding manual for qualitative researchers. *The coding manual for qualitative researchers* (2021), 1–440.

[47] Robert W Service. 2009. Book Review: Corbin, J., & Strauss, A.(2008). Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory . Thousand Oaks, CA: Sage. *Organizational Research Methods* 12, 3 (2009), 614–617.

[48] Ziang Xiao, Xingdi Yuan, Q Vera Liao, Rania Abdelghani, and Pierre-Yves Oudeyer. 2023. Supporting Qualitative Analysis with Large Language Models: Combining Codebook with GPT-3 for Deductive Coding. In *Companion Proceedings of the 28th International Conference on Intelligent User Interfaces*. 75–78.

[49] Cen April Yue, Linjuan Rita Men, and Mary Ann Ferguson. 2019. Bridging transformational leadership, transparent communication, and employee openness to change: The mediating role of trust. *Public relations review* 45, 3 (2019), 101779.

[50] Himanshu Zade, Margaret Drouhard, Bonnie Chinh, Lu Gan, and Cecilia Aragon. 2018. Conceptualizing disagreement in qualitative coding. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–11.

[51] Tania Zittoun, Aleksandar Baucal, Flora Cornish, and Alex Gillespie. 2007. Collaborative research, knowledge and emergence. *Integrative Psychological and Behavioral Science* 41 (2007), 208–217.

Table 1: Participant Demographics in Exploration Interview

No.	Fields of Study	Current Position	QA Software	Years of QA
P1	HCI, Ubicomp	Postdoc Researcher	Atlas.ti	4.5
P2	HCI, NLP	PhD student	Excel	1.5
P3	HCI, NLP	PhD student	Google Sheet/ Whiteboard	4
P4	HCI, Health	PhD student	Google Sheet	4
P5	Software Engineering	PhD student	Google Sheet	1

Table 2: Different CQA Software. Note: This list is not exhaustive and based on public online resources

Application	**Atlas.ti Desktop**	**Atlas.ti Web**	**nVivo Desktop**	**Google docs**	**MaxQDA**
Collaboration ways	Coding separately and then export the project bundles to other coders	Coding on one web page	share and coding independently and then merge	Collaborative online and simultaneous	Provide master project that includes docments and primary codes, and then send copies to others, allowing them to merge
Coding phase	Deductive coding for IRR	All phases	Dedcutive coding	All phases	Deductive coding
Independence	independent	not independent	Inpedendent	not independent	Inpedendent
Synchrony	Asynchronous	Synchronous	Asynchronous	Synchronous	Asynchronous
Unit of analysis	Select any text (unit of analysis is every character, one can select any length of characters)	Select any text	Select any text, but calculation can be on character, sentence, paragraph	Select any text	Select any text
IRR	Percent Agreement; Holsti Index; Krippendorff's family of Alpha	NA	Percentage agreement; Kappa coefficient	NA	percentage agreement; kappa coefficient
Calculation of IRR	only start distribute projects and do calculation after code system is stable and all codes are defined	Cannot	distribute projects, finish independent coding and merge them, then calculate intercoder reliability	NA	distribute projects, finish independent coding and merge them, then calculate intercoder reliability
Multi-valued coding	support adding multiple codes	support adding multiple codes	support adding multiple codes	support adding multiple codes	support adding multiple codes
Uncertainty	NA	NA	quickly see where there is agreement or disagreement in the source or node using the green, yellow, and blue markers on the scroll bar.	NA	0-50 or memo types for Open question / ambiguity (discuss in team)

Table 3: The prompts utilized in DiffCoder for each feature when communicating with the ChatGPT API to produce code suggestions for text.

Phases	Features	Prompts
Phase1	Seek code suggestions for units	Create three general summaries for [text] (within six-word)
	Seek most relevant codes from coding history	Identify the top three codes relevant to this [text] from the following code list: 1. [Code] 2. [Code] ... Here is the example format of results: 1. code content 2. code content 3. code content
Phase2	Make code decisions	Create three concise, non-repetitive, and general six-word code combinations for the [text] using Code1 ([Code]) and Code2 ([Code]): Here is the format of results: Version1: Version2: Version3: Requirements: 6 words or fewer; No duplicate words; Be general; Three distinct versions
Phase3	Generate code groups	Organize the following codes into 5 thematic groups without altering the original codes, and name each group: 1. [Code] 2. [Code] ... Here is the format of the results: Group1: [theme] 1.[code] 2.[code] 3.[code]

Table 4: Demographics of Participants in User Evaluation. Note: QA expertise is not solely determined by the number of QA experiences, but also by the level of QA knowledge. This is why some participants with 1-3 instances of prior experience may still regard themselves as having intermediate expertise.

Pairs		English	Job	Education	Field of expertise	Self-reported QA expertise	QA Times	Software for QA
Pair1	P1	Proficient	Student	Master	Basic understanding of qualitative research method	No Experience	None	None
	P2	First language	Automation QA Engineer	Undergraduate	Automation	No Experience	None	None
Pair2	P3	First language	Phd Student	PhD and above	HCI	Expert	7 times above	Atlas.ti Desktop
	P4	First language	Undergraduate	Undergraduate	Business analytics with Python and R	No Experience	None	None
Pair3	P5	Proficient	Student	Undergraduate	Coding with Python	Beginner	1-3 times	None
	P6	First language	Research Assistant	Master	Asian studies	Expert	7 times above (mainly interview data)	Word, Excel, Dedoose
Pair4	P7	First language	Data Analyst	Undergraduate	Data Visualisation	No Experience	None	None
	P8	First language	Student	Undergraduate	R, HTML/CSS, Market research	Beginner	1-3 times	R
Pair5	P9	First language	Researc assistant	Undergraduate	Learning science, Grounded theory	Intermediate	4-6 times	nVivo
	P10	First language	Data science intern	Undergraduate	CV using Python	No Experience	None	None
Pair6	P11	First language	Behavioral Scientist	Undergraduate	Psychology, Behavioral Science, Thematic analysis	Intermediate	1-3 times	Word
	P12	First language	Student	Undergraduate	Accounting & Python, SQL	No experience	None	None
Pair7	P13	First language	Research Assistant	Undergraduate	SPSS, Python, basic qualitative analysis understanding, topic modeling	Beginner	1-3 times	None
	P14	First language	Research Assistant	Undergraduate	Have research experience using QA for interview transcription	Intermediate	7 times above	nVivo, Excel
Pair8	P15	First language	Researcher	Master	Thematic analysis for interview, literature review	Beginner	1-3 times	fQCA
	P16	First language	Student	Master	Social science	No experience	None	None

Table 5: Overview of the final coding results. "Diff." denotes DiffCoder, "Atlas." denotes Atlas.ti Web, "Total." denotes the total number of codes generated while "Discussed" denotes the total number of codes that were discussed by the coders during the Discussion phase.

Pairs	Self-reported QA expertise	Conditions	Collaboration Observation	Total Codes		Discussed Codes		IRR (-1 to 1)		Code Groups		Discussion Time (mins:secs)		Decision Derived from GPT (Percentage)		Self-propose (Percentage)	Decision Derived from GPT (Percentage)
				Diff.	Atlas.	Diff.	Atlas.	Diff.[b]	Atlas.	Diff.	Atlas.	Diff.	Atlas.	GPT	Rele.		
P1	Beginner	A (Business),	Following-Leading	15	24	15	6	NA	-0.07	6	3	19:41	07:39	100	0	0	5
P2	No Experience	D (History)												70	5	25	
P3	Expert	D (Business),	Respectful collaboration	15	10	15	10	NA	1	5	4	35:24	21:32	90	0	10	40
P4	No Experience	A (History)												90	0	10	
P5	No Experience	A (History),	Following-Leading	15	11	15	2	NA	-0.02	5	2	17:55	06:16	73	7	20	100
P6	Expert	D (Business)												100	0	0	
P7	No Experience	D (History)	Respectful collaboration	15	22	15	2	NA	-0.33	7	6	29:08	No discussion[b]	7	0	93	80
P8	No Experience	A (Business)												13	7	80	
P9	Intermediate	A (Business),	Respectful collaboration	15	17	15	5	NA	0.04	5	2	15:11	14:38	73	13	13	80
P10	No Experience	D (History)												53	40	7	
P11	Intermediate	D (Business),	Quick and not careful	15	61	15	2	NA	-0.07	3	3	19:23	14:15	100	0	0	100
P12	No experience	A (History)												100	0	0	
P13	Beginner	A (History),	Respectful collaboration	15	30	15	5	NA	-0.08	8	2	29:19	08:43	87	7	7	100
P14	Intermediate	D (Business)												93	0	7	
P15	Beginner	D (History)	Respectful collaboration	15	8	15	4	NA	0.04	4	2	29:09	08:52	100	0	0	43
P16	No experience	A (Business)												73	20	7	
Mean				15	22.88	15	4.5	NA	0.06	5.38	3	24:00	10:48	76.46	6.15	17.4	68.5
SD				0	17.19	0	2.73	NA	0.40	1.60	1.41	07:12	05:24	29.43	10.74	28.11	35.3

[a] P5 and P6 gave up discussion for the Atlas.ti session due to spending too much time in the DiffCoder session.

[b] Following the discussion session in DiffCoder, the original codes have been restructured and finalized as a single code decision, resulting in an IRR of 1. Consequently, IRR calculations are not applicable for DiffCoder session.