Assessing the Potential and Limits of Large Language Models in Qualitative Coding

Xiner Liu[1][0009-0004-3796-2251], Jiayi Zhang[1] [0000-0002-7334-4256], Amanda Barany[1][0000-0003-2239-2271], Maciej Pankiewicz[1][0000-0002-6945-0523], and Ryan S. Baker[1][0000-0002-3051-3232]

[1] University of Pennsylvania, Philadelphia, PA 19104, USA
xiner@upenn.edu

Abstract. This paper examines the advantages and limitations of conducting automated coding of virtual tutoring session transcripts using the GPT-4 Turbo model via the OpenAI API. We compare three coding methods: (1) zero-shot, which relies solely on construct definitions; (2) few-shot, which includes annotated examples; and (3) coding with context, which provides GPT-4 with surrounding dialogue and study context. We used these approaches to code ten constructs from an existing codebook. We then had a set of experienced qualitative researchers rate the set of constructs across multiple dimensions. The results show that while zero-shot coding is effective for constructs with clear definitions, it tends to miss cases and struggles with constructs requiring contextual understanding. Few-shot coding works well for constructs that are seen as objective by experts, and those for which experts feel examples are needed to fully understand. However, it tends to overgeneralize based on the examples included in the prompt. Coding with context is particularly effective for constructs that often appear as part of sequences, but can also lead to the model coding more based on the context rather than the current line. This investigation highlights the potential of GPT-4 Turbo for efficient auto-coding of large datasets but emphasizes that specific prompting decisions impact quality and that the optimal decisions vary based on the characteristics of what is being coded.

Keywords: Large Language Model, GPT, Qualitative Coding, Automated Coding

1 Introduction

With the recent large language model (LLM) advancements in technology for natural language processing, an increasing number of studies are exploring the use of these models for qualitative coding (e.g., [20, 23]). Qualitative data coding using LLMs potentially offers a more cost-effective and time-efficient way of analyzing text. For example, GPT (Generative pre-trained transformers), a LLM that can be interacted with through prompts to ChatGPT, has been increasingly employed for this purpose, demonstrating promising results in coding various constructs (e.g., [15]). In these cases, prompts are given to GPT on how and what to code, with definitions and examples.

However, questions have been raised as to how accurate and reliable natural language processing models can be for coding qualitatively [10]. Even given the evidence that GPT can code accurately [5], it still remains unclear what constructs it is best able to code. Does its ability to code vary across constructs, and how does the complexity of a construct influence its ability? To provide some evidence on these questions, we compared GPT-4's ability to code ten constructs using three types of prompts: zero-shot prompting, few-shot prompting, and prompts with context. To understand why the ability to code might vary, we identified five dimensions (i.e., clarity, concreteness, objectivity, granularity, and specificity) to evaluate the attributes of a construct. We asked experts to rate the ten constructs using the five dimensions and used the ratings, along with a measure that indicates the usefulness of examples, to examine how these dimensions of a coding category might explain the variability in ChatGPT's ability to code using the three prompting methods.

2 Related Work

2.1 Qualitative Coding

Qualitative coding, as a critical step in qualitative research, involves systematically labeling, categorizing, and organizing data into themes, concepts, or patterns to identify recurring ideas or concepts within the data [18]. Researchers assign labels or "codes" to segments of data that represent meaningful units of information. By analyzing and drawing connections among the codes, researchers are able to explore and interpret the underlying meanings and patterns [18].

Codes can be generated inductively, where researchers create codes based on the data itself without preconceived ideas or theoretical frameworks, or deductively (top-down), where researchers rely on existing theories to define and operationalize codes [14]. The two methods are often used iteratively to ensure that the codes are grounded in both the data and theories [19]. In deductive coding, a predefined codebook is often used by raters, deciding on the presence or absence of a code within a segment of text.

As noted in [19], qualitative coding can be a time-consuming and labor-intensive process, as researchers need to manually examine each text segment and label the codes. For years, researchers have attempted to find ways to qualitatively code automatically or partially-automatically [4, 8]. More recently, an increasing number of studies have explored the use of ChatGPT, to facilitate the coding process. In these studies, prompts are given to ChatGPT to tell it how to code, along with definitions and, in some cases, examples. For example, in [24], ChatGPT (GPT-4) was instructed to code the topic and valence of press releases (e.g., economic positive, economic negative, medical positive). Similarly, Chew et al. [5] used GPT-3.5 to apply codes to four publicly available datasets collected from reports, news articles, blog posts, and social media. In the field of education, [23] instructed ChatGPT (GPT-3) to code the types of students' help-seeking and [12] used ChatGPT to rate the quality of peer feedback. These studies found GPT promising for conducting coding and achieving good agreement to human-coded labels. However, some of these studies have found that LLMs' success in coding varies across constructs/coding categories and that LLMs perform worse for some constructs compared to more traditional NLP models.

2.2 GPT and Prompt Engineering

Large language models (LLMs) are advanced artificial intelligence systems designed to generate human-like text based on patterns and structures learned from large amounts of training data [3]. GPT as a generative AI built based on large language models has demonstrated considerable ability in processing and generating natural language.

When using GPT, a prompt, which often includes a set of instructions, is used to describe a task, provide context, define guidelines, and/or specify a desired output [16]. Prompt engineering - the process of refining and improving prompts - may be employed to optimize the results. Several prompting methods have been experimented with and used to instruct models to improve outcomes, differentiated based on the specificity of instruction and the use of examples. For example, zero-shot prompting involves giving instructions for a task without any labeled examples, whereas one-shot or few-shot prompting involves using labeled data that provide examples for the model to learn from in addition to instructions. In [23], when GPT was instructed to code types of help-seeking, prompts that included examples (one-shot and few-shot) had higher agreement with experts' coding than prompts with just a code and description (zero-shot). Similar results were found in [3] and [17]. The selection of examples (e.g., including both positive and negative examples) and the ordering of the examples have also been examined in relation to model performance [13]. In addition to the use of examples, prompt engineering research has also explored how the structure of prompts [22], the inclusion of context [11] such as describing GPT's role in performing the task, and requiring GPT to justify its coding decisions [9] influence model performance.

3 Method & Results

3.1 Context

The data used in our study consists of deidentified transcripts from four 60-minute virtual tutoring sessions with 9th-grade students from high-poverty schools in the United States during the 2022-2023 academic year, focusing on Algebra I, obtained from [2]. The high dosage tutoring sessions were facilitated by the Saga Education platform. In these sessions, trained tutors offered personalized, small-group support for mathematics learning, a type of learning support that has been shown to be beneficial for students' math learning, achievement, and grades (e.g., [6, 7]). Each line of the transcript is annotated with the speaker and timestamp.

3.2 Codebook Development

To study GPT's coding capabilities, we obtained a prior codebook developed by [2] for this data set. This prior study compared four different approaches to codebook development: fully manual (human only), fully automated (ChatGPT only), and two hybrid approaches that integrate ChatGPT at different stages of the development process. By using an existing codebook not developed by the researchers in the current study, we aim to mitigate potential biases, such as being able to better craft the prompts that could arise from familiarity with the codebook's creation process.

For our analysis, we chose the hybrid codebook that was first crafted by humans and then refined using GPT (Human → ChatGPT). This codebook was selected because it offers the widest range of thematic meanings among those developed in the previous study. This broad range of themes allows for a more comprehensive examination of GPT's capabilities in coding various constructs, which is important for understanding both the strengths and limitations of this automated coding tool in qualitative research.

In the study by [2], two researchers who were not involved in developing the codebook were randomly assigned to code the tutoring lesson transcripts using the Human → ChatGPT codebook. However, even after two rounds of coding, only four out of ten codes achieved a kappa of 0.6 or above. Therefore, in our study, two (new) human researchers independently coded the transcript lines based on the definitions of the constructs outlined in the codebook (see Table 1). The kappa values from their first round of coding were inconsistent, ranging from 0.24 to 0.87, which is comparable to the kappa values obtained in the second round of coding in [2]'s study using the same codebook. Since this human-coded data did not achieve sufficient agreement, the two researchers in our study then discussed any coding disagreements to achieve consensus and establish a single categorization for each transcript line [19].

Table 1. Codebook used for GPT coding capabilities assessment

Construct	Definitions and examples
Greetings ($\kappa = 0.70$)	**Def**: Lines unrelated to learning, useful for rapport. Lines during the start or mid-session as "Engagement Checks." **Ex**: "What's good, [Redacted]?"
Direct instruction ($\kappa = 0.24$)	**Def**: Providing information or demonstrating methods without immediate student participation. • Definitions/Explanations: Stating mathematical rules or properties. • Demonstrating Steps: Giving instructions of how to solve a problem. **Ex**: "We got twelve equals one over x minus five."
Guided practice ($\kappa = 0.35$)	**Def**: Engaging students in problem-solving with support. Instructions include explanations, illustrations, reminders, and invites understanding. **Ex**: "Do that and then I want to see if you can solve from there."
Questioning ($\kappa = 0.87$)	**Def**: Prompting students to think, respond, or elaborate. • Recall & Comprehension: Asking students to remember or use something previously learned. • Higher Order Thinking: Questions that push students to analyze, evaluate, or plan next steps. **Ex**: "Twelve times x gives you what?"

Connect prior knowledge (κ = 0.45)	**Def**: Linking current topics to previously learned concepts for cohesive understanding. **Ex**: "What kind of math is a fraction?"
Clarification (κ = 0.72)	**Def**: Reiterating or paraphrasing for clearer understanding, helping move from abstract to concrete thinking. **Ex**: "Anytime we multiply, we always multiply what's in the denominator."
Feedback (κ = 0.66)	**Def**: Offering constructive comments on student's performance or understanding. • Positive Reinforcement: Confirming correct understanding or steps or offering words of encouragement or praise to motivate or acknowledge effort. • Corrective: Pointing out error, with or without explicitly giving correction. • Yes, and: Acknowledging student understanding and extending it. **Ex**: "The first one is right."
Engagement checks (κ = 0.48)	**Def**: Actively seeking signs of students' attention and participation. • Direct Check: Directly asking or observing the student's involvement. • Engagement Probes: Using strategies to pull students back into the lesson. **Ex**: "You working or you phased out?"
Software/ tool use (κ = 0.45)	**Def**: Reference to or assistance with using the tutoring software itself. **Ex**: "Touch screen you can pinch and move it around."
Session logistics (κ = 0.33)	**Def**: Addressing or organizing the structural aspects of the session. Indicating goals and tasks. Could be at the start, during, and end of session. **Ex**: "Try out number nine."

* "κ" measures kappa between two human coders during the first round of coding

3.3 Coding Data with GPT

We then proceeded to code the data using GPT-4 Turbo, which is accessible via an application programming interface (API) provided by OpenAI. For this research, we used the model version gpt-4-Turbo-2024-04-09, the most recent version at the time of writing. This version has an updated knowledge cutoff of April 2023, and a context window of 128,000 tokens. The cost structure is $0.01 per thousand tokens for input, which is one-third the cost of the original GPT-4 model, and $0.03 per thousand tokens for output, which is half the cost for output tokens compared to the original GPT-4 model. We used the default settings for hyperparameters, except for setting the temperature hyperparameter to 0 to ensure the consistency of the output.

When coding the data, we asked GPT to produce binary labels: 0 or 1. However, in rare instances where transcriptions were poor, GPT produced non-binary responses (e.g., "Sure, please provide the line you'd like to code"). We treated any response from GPT that did not provide binary labels as being incorrect, regardless of the ground truth value, since these responses would not be usable by a coder going forward.

Due to the stochastic nature of GPT models, which can result in variable outputs, we ran the coding process three times to enhance the accuracy and thoroughness of our evaluation for each coding approach. We then computed the average values for Kappa (κ), precision, and recall across all three iterations to assess GPT's performance. Given the emphasis on analyzing tutors' teaching methodologies, we excluded lines spoken by students from the model evaluation process. This process resulted in a dataset consisting of 990 lines. For the third method, which also sends prior lines of data as context, we included student lines, but asked GPT to reference them instead of code them.

Method 1: Coding with Zero-shot Prompting. We first coded the data using zero-shot prompting. This method involves taking each construct one at a time, and then presenting the GPT-4 Turbo model with the definition of the current construct, followed by directly asking it to code each line in the full dataset using the following prompt:

> *Please review the provided text and code it based on the construct: {construct}. The definition of this construct is {definition}. After reviewing the text, assign a code of '1' if you believe the text exemplifies {construct}, or a '0' if it does not. Your response should only be '1' or '0'.*

This prompt was sent as a system message to the Chat Completions API endpoint, followed by the specific line of data that GPT should code sent as a user message. After coding all 10 constructs, we calculated the performance metrics for each (See Table 2).

Table 2. Performance metrics for the zero-shot prompting

Construct	κ	Precision	Recall	F1	Shaffer's Rho
Greetings	**0.79**	**0.69**	**0.96**	**0.80**	**<0.01**
Direct instruction	0.11	1.00	0.06	0.12	1.00
Guided practice	0.16	1.00	0.11	0.19	1.00
Questioning	**0.91**	**0.91**	**0.93**	**0.92**	**<0.01**
Connect prior knowledge	0.18	1.00	0.10	0.19	1.00
Clarification	0.30	0.70	0.20	0.31	1.00
Feedback	0.27	0.40	0.24	0.30	1.00
Engagement checks	0.45	0.66	0.37	0.47	1.00
Software	0.25	0.67	0.15	0.25	1.00
Session logistics	0.15	1.00	0.09	0.16	1.00

* Constructs with kappa values greater than 0.7 are bolded

Review of Mis-coded Cases. We then analyzed misclassified cases in order to better understand GPT's decision-making process in coding and its limitations. Upon review, we noticed that GPT's zero-shot coding approach is very conservative and tends to expect direct matches to the definitions in the codebook. For instance, for the construct *Direct Instruction*, GPT coded "So we got X minus three equals six" as 1, but coded "You want to get another six" as 0. This is likely because the instruction in the second case is more implied and conversational. This issue is also reflected in the higher precision scores compared to recall for eight out of the 10 constructs. This suggests that while GPT (zero-shot) is usually correct when it identifies a code, it often misses relevant instances that are less clear-cut. This highlights the importance of providing clear and comprehensive definitions in the codebook for a zero-shot approach, as this is key for GPT to perform accurately in qualitative coding.

Secondly, we recognized that GPT (zero-shot) struggles with constructs requiring contextual understanding. For example, it coded the line "How you doing over there, [Redacted]?" as 1 for *Greetings*. However, this line occurred in the middle of a class session, which makes it an instance of *Engagement Checks* rather than *Greetings*. Similarly, GPT coded every instance of "Perfect" as 1 for *Feedback*, even in cases where the instructor might be using "Perfect" as a filler word without offering actual feedback or encouragement. Additionally, GPT does not perform well at coding constructs that span multiple related lines of the same dialogue. For example, for the construct *Feedback*, human coders identified the consecutive lines "No" and "We're not going to multiply here" as *Feedback* (1). However, GPT only coded the second line as 0. This indicates that for coding highly conversational data or constructs that require understanding context across multiple lines, the context-minimal zero-shot approach may not be ideal.

Method 2: Coding with Few-shot Prompting

We then coded the data using few-shot prompting. This approach uses the same prompt as the zero-shot method but now includes annotated examples with explanations before the line to be coded. We opted for annotated examples (instead of simply providing example text with no explanation) to provide explicit guidance on how the constructs should be interpreted and applied, which might improve the model's accuracy in identifying and classifying relevant content as well as the edge cases. Below are three annotated examples for the construct *Direct Instruction*:

> *(1) "Yeah, so track five on both sides first" because it specifies an action to be taken to solve a problem (2) "We got twelve equals one over x minus five" because it guides the student through a step in the process of solving an equation (3) "Remember, remember we're trying to get x by itself." because it provides guidance on what the focus should be during the task.*

This approach's performance when applied to the coding of the 10 constructs is shown in Table 3.

Table 3. Performance metrics for the few-shot prompting

Construct	κ	Precision	Recall	F1	Shaffer's Rho
Greetings	0.50	0.37	0.85	0.51	1.00
Direct instruction	**0.79**	**0.95**	**0.71**	**0.81**	**<0.01**
Guided practice	0.55	0.56	0.83	0.67	0.32
Questioning	**0.89**	**0.85**	**0.97**	**0.91**	**<0.01**

Connect prior knowledge	0.38	0.27	0.94	0.42	1.00
Clarification	0.56	0.46	0.90	0.60	0.37
Feedback	0.26	0.25	0.35	0.30	1.00
Engagement checks	**0.82**	**0.80**	**0.86**	**0.83**	**<0.01**
Software	**0.71**	**0.67**	**0.77**	**0.71**	**0.07**
Session logistics	**0.85**	**0.80**	**0.91**	**0.85**	**<0.01**

* Constructs with kappa values greater than 0.7 are bolded

Review of Mis-coded Cases. Upon reviewing the misclassified cases, we observed that with more examples, the few-shot approach is generally able to identify when each construct appears with the exception of *Feedback*. However, it tends to overgeneralize based on the examples included in the prompt and incorrectly codes many irrelevant instances as 1. For example, when the interjection "All right, fellas" was included as an example for *Greetings*, GPT overgeneralized part of that phrase. As a result, 23 instances of "All right" were misclassified as *Greetings*, even when it was used in an adverbial/adjectival form (i.e., "All right, let's look at number one."). When "All right, fellas" was removed as an example, misclassification dropped significantly. A similar overgeneralization issue arose with the *Feedback* construct, where GPT incorrectly coded 10 out of 11 instances that contained only the word "No" as containing *Feedback* after being given the following example: "No, not quite one x because you divided the negative three by three but did you divide the x by x?" This issue is also reflected in recall scores, which increased for all constructs compared to the zero-shot approach, while precision decreased for all but one construct. The findings highlight the importance of selecting examples that minimize the risk of overgeneralization, and reviewing results in detail to identify unanticipated cases where they occur.

Method 3: Coding with context

For our third approach, we added context to the coding prompt in addition to the construct definition and annotated examples used in the second method. We implemented this approach for two reasons. First, some sentences in the transcription were split across multiple lines (e.g., when participants interrupted each other), and this approach allows GPT to reference earlier parts of a sentence if needed. Second, some construct definitions in the codebook specify when they are likely to occur during the tutoring session (e.g., *Greetings* often occur at the start, *Engagement Checks* might occur mid-session), and including prior lines (if they are present) might help GPT identify the location of the line. Contextual information consisted of (1) a summary background of the study covering how the data was collected, the subjects taught, and the recording of transcripts; (2) the three lines preceding the current line (if not coding the first three lines), and (3) each line's timestamp and speaker (instructor or student). The decision to include three lines was based on a preliminary analysis of 20 randomly selected lines. For example, when coding the fourth line in the second tutoring session, the model will receive the following contextual information along with the study background:

CONTEXT (3 lines before the text you should code. Use this for context understanding, but do not code this part):
00:07 - [Instructor]: "Okay, so you should remember this from last time."
00:12 - [Instructor]: "We're gonna go ahead and use our grouping method."
00:17 - [Instructor]: "So factor these equations using our grouping method."

The performance metrics for this approach are recorded in Table 4.

Table 4. Performance metrics for coding with context

Construct	κ	Precision	Recall	F1	Shaffer's Rho
Greetings	**0.74**	**0.82**	**0.69**	**0.75**	**0.01**
Direct instruction	0.62	0.56	0.89	0.69	0.10
Guided practice	**0.86**	**0.91**	**0.87**	**0.89**	**<0.01**
Questioning	0.60	0.53	0.98	0.69	0.22
Connect prior knowledge	**0.78**	**0.78**	**0.83**	**0.81**	**<0.01**
Clarification	0.30	0.23	0.87	0.37	1.00
Feedback	0.15	0.13	0.67	0.21	1.00
Engagement checks	0.53	0.48	0.68	0.56	1.00
Software	0.60	0.43	1.00	0.60	0.32
Session logistics	0.41	0.28	0.97	0.43	1.00

* Constructs with kappa values greater than 0.7 are bolded

Review of Mis-coded Cases. The context-based coding approach seems to work best for constructs that typically involve repetition or continuation across consecutive lines, such as *Guided Practice* and *Connect Prior Knowledge*. For these categories, isolated lines (defined as lines coded as 1 with no other 1's in the three rows above) appeared only 29% and 26% of the time, respectively. Despite explicit instructions to GPT to use the previous three lines as reference but not to code them, it still tends to code the current lines based on the previous ones, which has resulted in many consecutive lines coded as 1s for each construct. This creates a problem when the conversation topic changes between prior lines and the coded line, such as in constructs that tend to appear in isolation. For example, the *Clarification* construct has isolated lines 65% of the time, and *Session Logistics* has isolated lines 60% of the time.

3.4 Evaluating Construct Complexity

Based on our previous experience in qualitative coding, we created a rubric of five dimensions that we found useful for evaluating the complexity of constructs: clarity, concreteness, objectivity, granularity, and specificity (See Table 5). These dimensions guided our investigation into the relationship between construct complexity and GPT-4 Turbo's coding ability. The rubric and codebook of 10 constructs were distributed to a group of researchers familiar with qualitative coding and analysis. Eleven researchers reviewed the name and definition of each construct and rated them from 1 (lowest) to 5 (highest) for each dimension. Researchers were then presented with examples and asked to rate their usefulness for improving their understanding of the constructs.

Table 5. The five dimensions used to evaluate construct complexity

Dimension	Definition
Clarity	Clarity (as opposed to ambiguity) measures how well-defined and easily comprehensible a concept is, without ambiguity or confusion.
Concreteness	Concreteness (as opposed to abstractness) refers to the quality of being specific, tangible, and perceptible by the senses.
Objectivity	Objectivity (as opposed to subjectivity) means verifiable information based on facts and evidence, whereas subjective means information or perspectives based on feelings, opinions, or emotions.
Granularity	Granularity (as opposed to coarseness), refers to the extent to which a concept involves finer, detailed elements.
Specificity	Specificity (as opposed to generality) measures the extent to which a construct is distinct and can be clearly distinguished from other related concepts, rather than being conflated with them/overlapping them.

For each construct, we computed the average of ratings on the five dimensions along with the usefulness of examples (See Table 6).

Table 6. Average scores of evaluated dimensions for each construct

Construct	Best method	Clarity	Concreteness	Objectivity	Granularity	Specificity	Example
Greetings	Zero-shot	4.64	4.18	3.45	3.82	3.45	3.64
Direct instruction	Few-shot	3.73	3.64	2.82	3.73	3.09	4.45
Guided practice	Coding with context	3.64	3.55	2.82	2.91	3.27	3.64
Question-ing	Zero-shot	4.82	4.45	4.45	4.27	4.27	3.64
Connect prior knowledge	Coding with context	3.91	3.27	3.36	2.82	3.27	3.64
Clarifica-tion	-	3.55	2.73	3.00	3.00	3.18	3.73
Feedback	-	3.82	2.82	3.00	3.55	3.18	3.64
Engage-ment checks	Few-shot	3.64	3.18	2.73	3.36	3.18	4.55

Software	Few-shot	3.45	3.91	4.45	3.55	3.73	4.64
Session Logistics	Few-shot	3.36	3.45	4.00	3.55	3.73	3.91

* The best coding method for each construct is selected if it has the highest κ among all three methods and $\kappa \geq 0.75$. Constructs with $\kappa < 0.75$ are still included in the correlation analysis.

Due to non-normality in the data, we used Spearman correlations to investigate the relationship between each pair of dimensions to understand how different dimensions relate to each other. Correlations were moderate, with an average Spearman correlation coefficient of 0.36 (SD = 0.49). Notably, there was a very high correlation (0.87) between objectivity and specificity.

3.5 Agreement and Construct Complexity

We next calculate Spearman correlations between the average values for the dimensions and the κ values achieved by GPT-4 Turbo in coding tasks for each of the three methods (See Table 7). We chose this approach because the small sample size limits the reliability and power of more complex statistical models. Additionally, multicollinearity among our dimensions makes it challenging to use regression-based methods without introducing significant bias or instability in the estimates. Spearman correlation coefficients, being non-parametric, do not assume a specific distribution of the data; this is important because many of our measures were non-normal, making them suitable for identifying monotonic relationships between the dimensions and the κ values.

We did not conduct post-hoc corrections for p-values because our significance testing showed that none of the correlations were statistically significant even without post-hoc correction. However, the Spearman correlation coefficients still provide suggestive and interpretable insights that allow us to generate hypotheses about the strengths and weaknesses of GPT as an automated coding tool.

Table 7. Summary of spearman correlation coefficients across different methods

Construct	Clarity	Concreteness	Objectivity	Granularity	Specificity	Example
Zero-shot	0.50	0.15	0.24	0.34	0.28	-
Few-shot	-0.24	0.39	0.20	0.41	0.36	0.48
Coding with context	0.40	0.55	-0.05	-0.12	0.17	-0.33

The positive coefficient of 0.50 for clarity and performance in zero-shot coding indicates a direct and moderate positive relationship between the clarity of the content and the κ score. This suggests that clearer and unambiguous definitions tend to improve agreement between human and GPT in a zero-shot approach. Interestingly, this correlation decreases for coding with context (0.40) and becomes negative for the few-shot approach (-0.24). This suggests that GPT can identify patterns that humans struggle to define but can recognize through examples. The zero-shot and few-shot approaches achieve better performance for more granular constructs; in these cases, extra context may not be useful (as the construct only needs one line due to its high granularity) and serves as a distraction. Examples that humans found useful were associated with better performance for the few-shot approach (0.48), but the reverse seemed to be true for coding with context (-0.33). It is possible that the additional context could overwhelm GPT, causing it to rely less on the examples and more on surrounding information.

4 Discussion and Conclusion

In this paper, we explored the potential of the GPT-4 Turbo model through the OpenAI API for automated coding, using a codebook previously developed in partnership between a human and GPT [2]. The data consisted of transcripts from four 60-minute virtual tutoring sessions with 9th grade students. To assess the complexity of constructs in the codebook, we asked experts in qualitative coding to evaluate the constructs across five identified dimensions. We then calculated the average of these dimensions and correlated these averages to the κ scores from three coding approaches to evaluate the strengths and weaknesses of GPT-4 Turbo for automated coding.

We employed three distinct methods to code the data: 1) zero-shot coding, which presents only the construct definition to GPT and prompts it to code, (2) few-shot coding, which includes annotated examples along with the construct definition before prompting GPT to code, and (3) coding with context, which provides GPT with some context of the study and the preceding lines to aid in coding the current line. For eight out of the 10 constructs, the GPT-4 Turbo achieved good agreement with human coders ($\kappa \geq 0.70$) for at least one of these prompt engineering approaches. This finding indicates GPT-4 Turbo's general capability to accurately code a wide range of constructs. However, each method showed unique strengths and limitations, and not every method was equally effective for all constructs.

We found that zero-shot prompting can achieve high performance for well-defined constructs (e.g., *Greetings* and *Questioning*), which have straightforward and easily comprehensible definitions. However, zero-shot coding tends to miss many cases, resulting in lower recall than other methods. This approach also incorrectly codes some cases due to a lack of contextual cues, similar to findings in [1, 21].

Few-shot coding is effective for constructs with high objectivity, such as *Software* and *Session Logistics*, where the information is based on verifiable information or evidence. It is also effective for constructs with low objectivity but highly useful examples, such as *Engagement Checks* and *Direct Instruction*, where additional explanations help clarify subjective information. However, incorporating annotated examples may lead to overgeneralization in some cases, as the model might apply specific patterns from the examples to unrelated contexts. In these instances, more straightforward zero-shot prompting often performed better.

Coding with context is more accurate for constructs that require understanding of surrounding context or temporal relationships between data lines. This method is useful for constructs where lines that should be coded often appear consecutively, such as

Guided Practice and *Connect Prior Knowledge*, where the preceding lines provide essential context for accurate coding. However, this approach can cause problems when context lines include different constructs than the current line being coded.

On the other hand, we observed that all methods struggled with constructs that have a lower level of concreteness, such as *Clarification* and *Feedback*. These constructs are more abstract and rely heavily on interpreting subtle cues and implicit information that the model struggles to accurately code. Furthermore, all methods, similar to humans, face difficulties with coding edge cases, particularly when additional context is needed to make a decision, as noted in [9].

One limitation of using the GPT-4 Turbo model through the OpenAI API, compared to ChatGPT, is that the API is less effective at explaining its decisions, identifying ambiguities in definitions, and discussing inconsistencies in human coding. This is because ChatGPT is specifically fine-tuned for conversation and interaction, which makes it better suited for these tasks. Examples of this can also be found in our prior work [2, 24]. However, using the API for qualitative coding has significant advantages in terms of efficiency. It is highly automated; once the prompt is defined and the chat completion endpoint is set up, it can automatically code all lines in the dataset. This eliminates the need to copy and paste or send prompts repeatedly to the chat window, making it a much more efficient approach when dealing with large datasets. Additionally, it is much easier to recode the data by API if the prompt needs to be updated and also allows researchers to modify the default hyperparameter settings (such as temperature) to achieve more consistent results.

Overall, this study suggests that GPT-4 Turbo can be useful for qualitative coding in Quantitative Ethnographic research, but that there remains some engineering work in doing so optimally. A current limitation of the work is that we only tested using one dataset and one codebook, so findings may not be generalizable to other research contexts. Future research should test this approach on other constructs and datasets to determine if different patterns or insights emerge. By systematically exploring which types of constructs GPT can code more effectively identifying those that present challenges, we aim to develop more reliable coding methodologies across different qualitative research contexts. Ultimately, this research seeks to harness the full potential of GPT for qualitative coding, turning it into a reliable tool that reduces the time researchers spend on coding without compromising quality.

Acknowledgments. We extend thanks to the Learning Engineering Virtual Institute Engagement Hub project for facilitating access to tutoring datasets from Saga Education.

References

1. Amarasinghe, I., Marques, F., Ortiz-Beltrán, A., Hernández-Leo, D.: Generative pre-trained transformers for coding text data? An analysis with classroom orchestration data. In: European Conference on Technology Enhanced Learning, pp. 32-43 (2023).
2. Barany, A., Nasiar, N., … Baker, R.S.: ChatGPT for education research: Exploring the potential of large language models for qualitative codebook development. In: Proceedings of the 25th International Conference on Artificial Intelligence in Education (in press).

3. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Amodei, D.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020).
4. Cai, Z., Siebert-Evenstone, A., Eagan, B., Shaffer, D. W., Hu, X., Graesser, A. C.: nCoder+: a semantic tool for improving recall of nCoder coding. In: Eagan, B., Misfeldt, M., Siebert-Evenstone, A. (eds.) Advances in Quantitative Ethnography. ICQE 2019. Communications in Computer and Information Science, vol. 1112. Springer (2019).
5. Chew, R., Bollenbacher, J., Wenger, M., Speer, J., Kim, A.: LLM-assisted content analysis: Using large language models to support deductive coding. arXiv:2306.14924 (2023).
6. Cook, P. J.: Not too late: Improving academic outcomes for disadvantaged youth. Northwestern University Institute for Policy Research Working Paper, pp. 15-01 (2015).
7. Cook, P.J., Dodge, K., Farkas, G., Fryer, R.G., Guryan, J., Ludwig, J., Steinberg, L.: The (surprising) efficacy of academic and behavioral intervention with disadvantaged youth: Results from a randomized experiment in Chicago, Working Paper No. 19862. National Bureau of Economic Research (2014).
8. Crowston, K., Liu, X., Allen, E. E.: Machine learning and rule-based automated coding of qualitative data. In: Proceedings of the American Society for Information Science and Technology, 47(1), pp. 1-2 (2010).
9. Dunivin, Z. O.: Scalable qualitative coding with LLMs: Chain-of-thought reasoning matches human performance in some hermeneutic tasks. arXiv preprint arXiv:2401.15170 (2024)
10. Gao, J., Choo, K. T. W., Cao, J., Lee, R. K. W., Perrault, S.: CoAIcoder: Examining the effectiveness of AI-assisted human-to-human collaboration in qualitative analysis. ACM Transactions on Computer-Human Interaction, 31(1), 1-38 (2023).
11. Hou, C., Zhu, G., Zheng, J., Zhang, L., ... Ker, C. L.: Prompt-based and fine-tuned GPT models for context-dependent and-independent deductive coding in social annotation. In Proceedings of the 14th Learning Analytics and Knowledge Conference, pp. 518-528 (2024)
12. Hutt, S., DePiro, A., Wang, J., Rhodes, S., Baker, R. S., Hieb, G., Mills, C.: Feedback on feedback: Comparing classic natural language processing and generative AI to evaluate peer feedback. In: Proceedings of the 14th Learning Analytics and Knowledge Conference, pp. 55-65 (2024).
13. Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., Neubig, G.: Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9), pp. 1-35 (2023).
14. Miles, M. B., Huberman, A. M.: Qualitative data analysis: An expanded sourcebook. Sage, Newcastle upon Tyne (1994).
15. Morgan, D. L.: Exploring the use of artificial intelligence for qualitative data analysis: The case of ChatGPT. International Journal of Qualitative Methods, 22 (2023).
16. OpenAI: ChatGPT: OpenAI's conversational language model (2022).
17. Prabhumoye, S., Kocielnik, R., Shoeybi, M., Anandkumar, A., Catanzaro, B.: Few-shot instruction prompts for pretrained language models to detect social biases. arXiv preprint arXiv:2112.07868 (2021).
18. Saldaña, J.: The coding manual for qualitative researchers, pp. 1–440 (2016).
19. Shaffer, D.W., Ruis, A.R.: How we code. In: Ruis, A. R., Lee, S. B. (eds.) ICQE 2020. CCIS, vol. 1312, pp. 62-77 . Springer, Cham (2021).
20. Tai, R. H., Bentley, L. R., Xia, X., Sitt, J. M., Fankhauser, S. C., Chicas-Mosier, A. M., Monteith, B. G.: An examination of the use of large language models to aid analysis of textual data. bioRxiv, pp. 2023-07 (2023).
21. Theelen, H., Vreuls, J., Rutten, J.: Doing research with help from ChatGPT: Promising examples for coding and inter-rater reliability. International Journal of Technology in Education, 7(1), 1-18 (2024).

22. White, J., Hays, S., Fu, Q., Spencer-Smith, J., Schmidt, D. C.: Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv preprint arXiv:2303.07839 (2023).
23. Xiao, Z., Yuan, X., Liao, Q. V., Abdelghani, R., Oudeyer, P. Y.: 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, pp. 75-78 (2023).
24. Zambrano, A. F., Liu, X., Barany, A., Baker, R. S., Kim, J., Nasiar, N.: From nCoder to ChatGPT: From automated coding to refining human coding. In: Arastoopour Irgens, G., Knight, S. (eds.) ICQE23. CCIS, vol. 1895, pp. 470-485, Springer, Cham (2023).