JOURNAL OF LEARNING ANALYTICS

Volume 12(1), 169–185. https://doi.org/10.18608/jla.2025.8575

Qualitative Coding with GPT-4: Where it Works Better

Xiner Liu[1], Andres Felipe Zambrano[2], Ryan S. Baker[3], Amanda Barany[4], Jaclyn Ocumpaugh[5], Jiayi Zhang[6], Maciej Pankiewicz[7], Nidhi Nasiar[8] and Zhanlan Wei[9]

Abstract

This study explores the potential of the large language model GPT-4 as an automated tool for qualitative data analysis by educational researchers, exploring which techniques are most successful for different types of constructs. Specifically, we assess three different prompt engineering strategies — Zero-shot, Few-shot, and Few-shot with contextual information — as well as the use of embeddings. We do so in the context of qualitatively coding three distinct educational datasets: Algebra I semi-personalized tutoring session transcripts, student observations in a game-based learning environment, and debugging behaviours in an introductory programming course. We evaluated the performance of each approach based on its inter-rater agreement with human coders and explored how different methods vary in effectiveness depending on a construct's degree of clarity, concreteness, objectivity, granularity, and specificity. Our findings suggest that while GPT-4 can code a broad range of constructs, no single method consistently outperforms the others, and the selection of a particular method should be tailored to the specific properties of the construct and context being analyzed. We also found that GPT-4 has the most difficulty with the same constructs than human coders find more difficult to reach inter-rater reliability on.

Notes for Practice

- GPT-4 can be used to code qualitative data for educationally relevant constructs.
- Using embeddings and examples can improve agreement with humans. Examples are more useful for constructs that are more difficult to define.
- Constructs that human beings find difficult to agree on are also difficult for GPT-4.

Keywords: Qualitative coding, GPT-4, large language model, quantitative ethnography, automated coding

Submitted: 27/07/2024 — **Accepted:** 11/02/2025 — **Published:** 17/03/2025

Corresponding author [1]Email: xiner@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0009-0004-3796-2251
[2]Email: azamb13@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0003-0692-1209
[3]Email: ryanshaunbaker@gmail.com Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0002-3051-3232
[4]Email: abarany@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0003-2239-2271
[5]Email: ojaclyn@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0002-9667-8523
[6]Email: joycez@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0002-7334-4256
[7]Email: mpank@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0000-0002-6945-0523
[8]Email: nasiar@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0009-0006-7063-5433
[9]Email: zhanlanw@upenn.edu Address: Graduate School of Education, University of Pennsylvania, Philadelphia, PA 19104, USA. ORCID iD: https://orcid.org/0009-0002-3931-6398

1. Introduction and Literature Review

Qualitative coding is a vital component of educational research. This process involves systematically labelling, categorizing, and organizing data into themes, constructs, concepts, or patterns to identify recurring ideas or concepts within the data (Saldaña, 2016). The qualitative coding pipeline (e.g., developing coding categories, coding data according to them, and validating codes) is often time-consuming and labour-intensive (Shaffer & Ruis, 2021), especially in research involving large datasets and complex or numerous constructs. This issue is exacerbated when understanding the patterns in a dataset requires

contextual knowledge of a culture or domain, or when specific types of semiotic and semantic literacy are needed for reliable interpretation (e.g., programming data, interaction log data).

Automated coding has been proposed for decades as a response to this issue (Weber, 1984; Shapiro, 1997), within a variety of data contexts and approaches, including content analysis for literature review in ecology research (Nunez-Mir et al., 2016), classification of opinions in government transcripts (Hopkins & King, 2010), latent semantic analyses of student interviews around science concept developments (Sherin, 2012), and student cognition (Kovanović et al., 2016). Approaches for doing so have ranged from the use of count data (Kovanović et al., 2016) to regular expression authoring (Cai et al., 2019; Crowston et al., 2010).

With the advent of large language models (LLM), a rapidly increasing number of studies both within and outside of the learning analytics community are exploring the use of these models for qualitative coding (e.g., Tai et al., 2024; Zambrano et al., 2023; Kirsten et al., 2024; Chew et al., 2023; Xiao et al., 2023; Hutt et al., 2024). Qualitative data coding using LLMs potentially offers a more cost-effective and time-efficient way of analyzing text than fully human coding, particularly for large datasets. There is also demonstrated potential for LLMs to serve as "co-researchers" that can support human refinement of qualitative codebooks or improve coding accuracy (Chew et al., 2023; Zambrano et al., 2023). OpenAI's Generative Pre-trained Transformers (GPT) LLM has been increasingly used for this purpose and has obtained promising results for coding a range of constructs (e.g., Morgan, 2023; Zambrano et al., 2023; Kirsten et al., 2024; Chew et al., 2023).

In these studies, prompts are given to ChatGPT (OpenAI, 2022) to tell it how to code, along with definitions and, in some cases examples. For instance, in Zambrano et al. (2023), ChatGPT (GPT-4) was instructed to code the topic and valence of press releases. Similarly, Chew et al. (2023) used GPT-3.5 to qualitatively code a range of categories in reports, news articles, blog posts, and social media. In educational domains, Xiao et al. (2023) used ChatGPT (GPT-3) to code different types of student help-seeking behaviours, and Hutt et al. (2024) used ChatGPT (GPT-4) to rate the quality of peer feedback. Results were promising, with GPT generally achieving good agreement with human-coded labels, but coding performance varied across constructs/coding categories and GPT in some cases performed worse than more traditional NLP approaches such as regular expression authoring (Zambrano et al., 2023).

Although GPT can accurately code some types of qualitative data (Chew et al., 2023), it is unclear which constructs GPT handles best. Many constructs are hard for humans to code (Gao et al., 2023; Kirsten et al., 2024) — will the same constructs be hard to code for GPT, or different ones? Furthermore, although there are now many examples of qualitative coding with GPT, it is unclear which approach is best, and whether the answer may differ across constructs and datasets.

Several methods for using GPT for qualitative coding have been investigated. For example, Zero-shot prompting involves giving instructions for a task without any labelled examples, whereas one-shot or Few-shot prompting involves using labelled data that provide examples for the model to learn from in addition to the instructions. When studying student help-seeking behaviours, Xiao et al. (2023) found that GPT achieved higher rates of inter-rater reliability with human experts when given prompts that included examples (One-shot and Few-shot) than when given Zero-shot prompts. Comparable results have been found in other domains (Brown et al., 2020; Prabhumoye et al., 2021). Liu et al. (2023) found that choices in example selection and ordering can also impact model performance. Within this paper, we will more systematically investigate the trade-offs associated with these choices of how to use an LLM for qualitative coding.

In addition to the use of examples, this study also investigates the details of prompt engineering, the process of designing inputs to guide a language model's behaviour and responses (Giray, 2023) in qualitative coding. Previous research (in domains other than qualitative coding) has shown how the structure of prompts (White et al., 2023), the phrasing and specificity of instruction (Ekin, 2023), the inclusion of guiding keywords or phrases (Spasić & Janković, 2023), and the formulation of tasks (e.g., requiring direct results or applying chain-of-thought reasoning for step-by-step problem-solving; Lo, 2023b) may all influence the results of prompting. A growing body of literature has also highlighted the role of contextual information in determining what output an LLM produces in response to differences in prompts. For instance, Hou et al. (2024) demonstrate that explicitly defining the model's role or persona within a task helps align its responses with role-specific expectations and requirements. Lo (2023a) highlights that including contextual elements, such as the objectives of the tasks, increases precision and reduces ambiguity in the output. Femepid et al. (2024) show that adding domain-specific information improves both the relevance and accuracy of the model's responses by grounding them in established norms and knowledge within the field. Building on these insights, we also integrated contextual elements related to the research purpose and data into the prompts used in this study to assess their impact on qualitative coding.

We also investigate the potential use of embeddings in qualitative coding. Embeddings are numerical representations of data points in a multi-dimensional space that transform qualitative data into a format suitable for computational analysis (Alvarez & Bast, 2017). While they have been extensively used in areas such as clustering, classification, and information retrieval (see review by Asudani et al., 2023) and play an essential role in retrieval-augmented generation within various applications of large language models (Zhao et al., 2024), their application in qualitative coding remains largely unexplored. Their primary use thus far has been to support clustering of text for the discovery of qualitative categories (Katz et al., 2024),

but they also have the potential to support the coding process by providing a way to quantify semantic similarities (Alvarez & Bast, 2017) between different pieces of text.

With these areas of potential enhancement in mind (e.g., prompt types, the use of examples, embeddings), this paper investigates five approaches for automated coding with GPT-4: 1) Zero-shot prompts, 2) Few-shot prompts with only positive examples, 3) Few-shot prompts with positive and negative examples, 4) Few-shot prompts with contextual information (e.g., related background information, the purpose of the conversation, or the surrounding text), and 5) the use of Embeddings. We test subsets of these variations on three different studies/datasets drawn from different educational tasks and domains. In Study 1, we examine transcripts from semi-personalized virtual tutoring sessions, specifically assessing how different prompt engineering strategies affect coding accuracy for constructs varying in clarity, concreteness, objectivity, granularity, and specificity. In Study 2, we evaluate the same approaches, plus embeddings, in learners studying astronomy within Minecraft. Finally, Study 3 extends this analysis to programming code from novice computer science students, testing GPT's ability for qualitative coding outside the context of natural language. For each study, we use slightly different methodologies (see discussion below) to account for the unique characteristics of each dataset. By combining insights from the three complementary studies, we aim to advance the field's understanding of how to utilize GPT-4 and similar LLMs most effectively for qualitative coding, identifying which coding methods work best for which types of constructs.

2. Study 1: Virtual Tutoring Session Transcripts

2.1. Dataset

The Study 1 dataset was obtained from the Saga Education platform, where trained tutors provided personalized mathematics support to students attending high-poverty schools in the United States. The dataset consists of de-identified transcripts, including timestamps of lines spoken and speaker type (instructor and student), from four 60-minute virtual tutoring Algebra I sessions with six 9th-grade students (two sessions involving two students, two sessions involving one student).

2.2. Codebook Development

This dataset has previously been used to explore the potential of LLMs (specifically GPT-4) to support codebook development for investigating teaching methodologies from transcripts (Barany et al., 2024). The prior study compared four codebooks inductively developed with different approaches: 1) a fully manual method using only human analysis, 2) a fully automated method using only ChatGPT, 3) a hybrid approach where GPT refined a codebook initially proposed by a human, and 4) another hybrid method where GPT proposed an initial codebook that was subsequently refined by a human. For our analysis, we selected the third codebook (initially crafted by humans and then refined by GPT) because its constructs encompass the broadest range of thematic meanings among the developed approaches, enabling a more comprehensive evaluation of GPT's effectiveness in coding constructs with various levels of complexity. This codebook, originally proposed in Barany et al. (2024), is presented in Table 1.

Table 1. Codebook for Study 1

Construct	Definitions & Examples
Greetings	Lines unrelated to learning, useful for rapport. Lines during the start or mid-session as an engagement check. **Example**: "What's good, [Redacted]?"
Direct Instruction	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. **Example**: "We got twelve equals one over x minus five."
Guided Practice	Engaging students in problem-solving with support. Instructions include explanations, illustrations, reminders, and invites understanding. **Example**: "Do that and then I want to see if you can solve from there."
Questioning	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. **Example**: "Twelve times x gives you what?"
Connecting to Prior Knowledge	Linking current topics to previously learned concepts for cohesive understanding. **Example**: "What kind of math is a fraction?"
Clarification	Reiterating or paraphrasing for clearer understanding, helping move from abstract to concrete thinking. **Example**: "Anytime we multiply, we always multiply what's in the denominator."

Feedback	Offering constructive comments on student 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 an error, with or without explicitly giving the correction. • Yes, and: Acknowledging student understanding and extending it. **Example**: "The first one is right."
Engagement Checks	Actively seeking signs of student attention and participation. • Direct Check: Directly asking or observing student involvement. • Engagement Probes: Using strategies to pull students back into the lesson. **Example**: "You working or you phased out?"
Software/ Tool Use	Reference to or assistance with using the tutoring software itself. **Example**: "Touch the screen; you can pinch and move it around."
Session Logistics	Addressing or organizing the structural aspects of the session. Indicating goals and tasks. Could be instances at the start, during, and end of the session. **Example**: "Try out number nine."

Source: Barany et al. (2024).

2.3. Automated Coding Process

In our study, two new researchers independently coded the transcript using the codebook's construct definitions (see Table 1). Their initial Cohen's Kappa (κ) values varied significantly, ranging from 0.24 to 0.87 (see Table 2), in line with the kappa values reported in Barany et al.'s (2024) study using the same codebook. Given the insufficient agreement obtained, researchers in our study resolved discrepancies through social moderation (Herrenkohl & Cornelius, 2013), aiming to achieve consensus and establish a single, accurate categorization for each transcript line. This coded data serves as the ground truth for training and evaluating the coding performance of GPT.

We then utilized GPT-4 (gpt-4-turbo-2024-04-09, the most recent version at the time of the research) for coding the data, accessed via Open AI's application programming interface (API). We employed the default hyperparameter settings, except for setting the temperature to 0 to ensure consistent output.

We used a binary (prompt-engineered) classifier for coding each construct to reduce the complexity of the coding task, as proposed by Zambrano et al. (2023), an approach that aligns with common practice in qualitative coding, particularly within the learning analytics community. When coding the data, GPT was specifically asked to assign binary labels, either 0 or 1. However, in rare instances where transcriptions were poor (e.g., lines that were transcribed as "Huh?" or [?: equals.]", which indicate that audio quality was so low that transcription was impossible,) 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. This was the only study where we observed such a case; this issue did not occur in the other two studies below.

Across all three studies, the strategy for developing effective prompts involved an iterative process of refinement to align GPT's responses with the coding task requirements. The goal was to craft clear, precise, structured prompts that reduced ambiguity, minimized variability, and maximized reliability across repeated outputs. Each prompt was evaluated using validation data (randomly drawn from the dataset but separate from the testing data) across multiple sessions, accounts, and computers. Insights from these iterations guided adjustments to improve prompt clarity and accuracy in representing the constructs being coded. Key adjustments included rephrasing instructions and specifying the expected format of responses. A prompt was finalized only when it consistently produced reliable binary outputs with minimal inconsistency across different attempts.

In this particular study, we compared three different prompt engineering approaches for coding the data: Zero-shot, Few-shot, and Few-shot with context (defined below). 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 student-spoken lines from the model evaluation process. This approach yielded a dataset of 990 lines. However, for the third method, where context is crucial for coding, we included student lines as reference material. Although presented, these lines were not coded by GPT; instead, they were used solely for reference to enhance the contextual understanding of the instructor's lines.

2.3.1. Method 1: Coding with Zero-Shot Prompting

For Zero-shot prompting, we first provided the GPT-4 model with the definition of each construct. Then, we prompted the model to code each line in the entire dataset using the following specific 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 endpoint, followed by the specific line of data that GPT should code, sent as a user message.

2.3.2. Method 2: Coding with Few-shot Prompting

This method extends the Zero-shot technique by including annotated examples as well as the line to be coded, including explanations of how the constructs should be interpreted and applied. Providing annotated examples — rather than only example texts without explanations — aims to enhance accuracy in identifying and classifying relevant content and addressing edge cases. Below are the 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.*

2.3.3. Method 3: Few-Shot with Context

In this dataset, some utterances might span multiple lines due to the transcription process, and some constructs in the codebook specify when they are likely to occur during the 60-minute tutoring session (e.g., *Greetings* typically occur at the start, whereas *Engagement Checks* occur later). Given this structure, we incorporated context into the coding prompt in addition to the construct definition and annotated examples used in the second method. Contextual information consisted of three parts: 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."

2.4. Results

2.4.1. Coding with Zero-Shot Prompting

The performance of Zero-shot prompting varied considerably (Table 2) from excellent (*Questioning* κ=-0.91, *Greetings* κ=0.79) to poor (κ<0.2 for *Direct Instruction*, *Session Logistics*, *Guided Practice*, *Connect Prior Knowledge*). GPT often struggled in cases where contextual understanding is required. For example, GPT (Zero-shot) coded the line "How you doing over there, [Redacted]?" as 1 for *Greetings*, but this line occurred in the middle of a class session, where it represents an *Engagement Check*. Similarly, GPT coded every instance of the word "Perfect" as *Feedback*, even in cases where the instructor appeared to be using "Perfect" as a filler word without offering actual feedback or encouragement. GPT also did not perform as well for 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 1. This indicates that for coding highly conversational data or constructs that require understanding context across multiple lines, the context-free Zero-shot approach may not be ideal.

We also observed that this approach tends to expect direct matches to the definitions in the codebook. For example, for the construct Direct Instruction, GPT correctly identified "So we got X minus three equals six," but did not identify "You want to get another six." The latter case may have been harder for GPT to correctly identify because the instruction is implied and conversational. Inter-rater agreement metrics for each approach (in Table 2) suggest that the Zero-shot approach was most successful in only two cases, but that Zero-shot often overlooks relevant instances that are less explicitly stated. This tendency is reflected in the higher precision than recall for 8 out of the 10 constructs, suggesting that clear and comprehensive definitions in a qualitative codebook were essential for the Zero-shot approach.

Table 2. Performance Metrics for Automated Coding

Construct	Freq. in Data	Hum–Hum κ	Method	Hum–GPT κ	Hum–GPT Prec.	Hum–GPT Recall
Greetings	2%	0.70	**Zero-shot**	**0.79**	**0.69**	**0.96**
			Few-shot	0.50	0.37	0.85
			Few-shot with context	0.74	0.82	0.69
Direct Instruction	14%	0.24	Zero-shot	0.11	1.00	0.06
			Few-shot	**0.79**	**0.75**	**0.71**
			Few-shot with context	0.62	0.56	0.89
Guided Practice	26%	0.35	Zero-shot	0.16	1.00	0.11
			Few-shot	0.55	0.56	0.83
			Few-shot with context	**0.86**	**0.91**	**0.87**
Questioning	18%	0.87	**Zero-shot**	**0.91**	**0.91**	**0.93**
			Few-shot	0.89	0.85	0.97
			Few-shot with context	0.60	0.53	0.98
Connect Prior Knowledge	12%	0.45	Zero-shot	0.18	1.00	0.10
			Few-shot	0.38	0.27	0.94
			Few-shot with context	**0.78**	**0.78**	**0.83**
Clarification	7%	0.72	Zero-shot	0.30	0.70	0.20
			Few-shot	0.56	0.46	0.90
			Few-shot with context	0.30	0.23	0.87
Feedback	5%	0.66	Zero-shot	0.27	0.40	0.24
			Few-shot	0.26	0.25	0.35
			Few-shot with context	0.15	0.13	0.67
Engagement Checks	6%	0.48	Zero-shot	0.45	0.66	0.37
			Few-shot	**0.82**	**0.80**	**0.86**
			Few-shot with context	0.53	0.48	0.68
Software	1%	0.45	Zero-shot	0.25	0.67	0.15
			Few-shot	**0.71**	**0.67**	**0.77**
			Few-shot with context	0.60	0.43	1.00
Session Logistics	4%	0.33	Zero-shot	0.15	1.00	0.09
			Few-shot	**0.85**	**0.80**	**0.91**
			Few-shot with context	0.41	0.28	0.97

Note: For each construct, the best coding method is highlighted/in bold if it also obtains a minimum of κ≥0.70. Constructs with κ≤0.70 are still included in the subsequent correlation analysis.

2.4.2. Coding with Few-Shot Prompting

The Few-shot prompting approach generally obtained better results than the Zero-shot prompting (Table 2). Several of the constructs had substantial improvements in inter-rater reliability, including *Direct Instruction* (κ=0.79 with Few-shot vs. κ=0.11 with Zero-shot), *Guided Practice* (κ=0.55 with Few-shot vs. κ=0.16 with Zero-shot), and *Session Logistics* (κ=0.85 with Few-shot vs. κ=0.15 with Zero-shot). In contrast, the performance for coding *Greetings* decreased by κ=0.29 compared to the Zero-shot approach. The Few-shot approach improved recall across all constructs but tended to overgeneralize based on the provided examples, resulting in a lower precision for all but one construct. 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?" Both examples highlight the importance of carefully selecting examples that minimize the risk of overgeneralization, and reviewing results in detail to identify unanticipated cases where it occurs.

2.4.3. Few-Shot with Context

Finally, as Table 2 shows, the Few-shot with context approach was most effective for constructs that typically involve repetition or continuation of a construct across consecutive lines, such as *Guided Practice* (κ=0.86) and *Connect Prior*

Knowledge (κ=0.78). However, for other constructs that typically occur in a single line (e.g., *Clarification* and *Session Logistics*), GPT sometimes coded the construct in the context lines rather than in the target line.

2.5. Evaluating Construct Complexity

Next, we explored the relationship between the characteristics of a construct and GPT-4's coding performance. We evaluated each construct in terms of five dimensions: 1) clarity, 2) concreteness, 3) objectivity, 4) granularity, and 5) specificity. We also evaluated the usefulness of the three examples in improving understanding of the construct. These evaluation criteria were developed based on our prior experience with qualitative coding, where we observed that the difficulty of coding and agreement between coders can be affected by the way such characteristics of a construct are presented. We distributed the rubric in Table 3 and descriptions of the ten constructs to a group of eleven researchers experienced in qualitative coding and analysis. For each construct, we provided its definition and asked the researchers to rate each dimension of the rubric on a scale from 1 to 5, with 1 being the lowest and 5 being the highest.

Table 3. The Dimensions Used to Evaluate Constructs

Dimension	Definition
Clarity	Well-defined and easily comprehensible; without ambiguity or confusion (antonym ambiguity).
Concreteness	Specific, tangible, and perceptible by the senses (antonym abstractness).
Objectivity	Verifiable based on facts and evidence; not based on feelings, opinions, or emotions (antonym subjectivity).
Granularity	Involving finer, detailed elements (antonym coarseness).
Specificity	Distinct and clearly distinguishable from other related concepts; not conflated or overlapping with other constructs (antonym generality).
Example	Were the examples useful in improving understanding of the construct.

For each construct, we calculated the average ratings for the dimensions (Table 4). To assess the reliability of the survey responses, we used Cronbach's Alpha, a statistical measure of internal consistency that evaluates how well a group of questions collectively measure the same concept. Alpha values range from 0 to 1, with higher scores reflecting greater consistency. Standard interpretation guidelines suggest that $\alpha \geq 0.9$ indicates excellent reliability, $0.8 \leq \alpha < 0.9$ signifies good reliability, $0.7 \leq \alpha < 0.8$ represents acceptable reliability, and $\alpha < 0.7$ points to low reliability. For each construct, we grouped six related questions (five addressing specific dimensions and one evaluating the usefulness of examples) and calculated Cronbach's Alpha to assess whether those questions collectively measure the corresponding construct. The Alpha values ranged from 0.75 to 0.93, with an average of 0.87, which indicates that the constructs demonstrate acceptable to excellent internal consistency overall.

Table 4. Average Scores of Evaluated Dimensions for Each Construct

Construct	Best Method	Hum–Hum κ	Hum–GPT κ	Clarity	Concrete.	Obj.	Gran.	Spec.	Ex.
Questioning	Zero-shot	0.87	0.91	4.82	4.45	4.45	4.27	4.27	3.64
Guided Practice	Few-shot w/context	0.35	0.86	3.64	3.55	2.82	2.91	3.27	3.64
Session Logistics	Few-shot	0.33	0.85	3.36	3.45	4.00	3.55	3.73	3.91
Engagement Checks	Few-shot	0.48	0.82	3.64	3.18	2.73	3.36	3.18	4.55
Greetings	Zero-shot	0.70	0.79	4.64	4.18	3.45	3.82	3.45	3.64
Direct Instruction	Few-shot	0.24	0.79	3.73	3.64	2.82	3.73	3.09	4.45
Connect Prior Knowledge	Few-shot w/context	0.45	0.78	3.91	3.27	3.36	2.82	3.27	3.64
Software	Few-shot	0.45	0.71	3.45	3.91	4.45	3.55	3.73	4.64
Clarification	NA*	0.72	0.56	3.55	2.73	3.00	3.00	3.18	3.73
Feedback	NA*	0.66	0.27	3.82	2.82	3.00	3.55	3.18	3.64

*Notes: * All approaches failed to reach κ >0.70. None of the methods obtained κ over 0.70 for Clarification and Feedback.*

Interestingly, the human–GPT κ values were more closely aligned with the clarity ratings than the human–human κ values, which indicates that GPT's performance may be more sensitive to well-defined constructs than that of human raters. In other

words, GPT may rely more on explicit definitions and structure when making coding decisions, whereas human raters might bring in additional context or subjective interpretation, even when constructs are less clear.

Due to non-normality in the data, we used Spearman correlations to investigate the relationship between each pair of dimensions. Correlations were moderate (see Figure 1), with an average Spearman correlation coefficient of 0.36 (SD = 0.49). Notably, there was a remarkably high correlation (0.87) between objectivity and specificity.

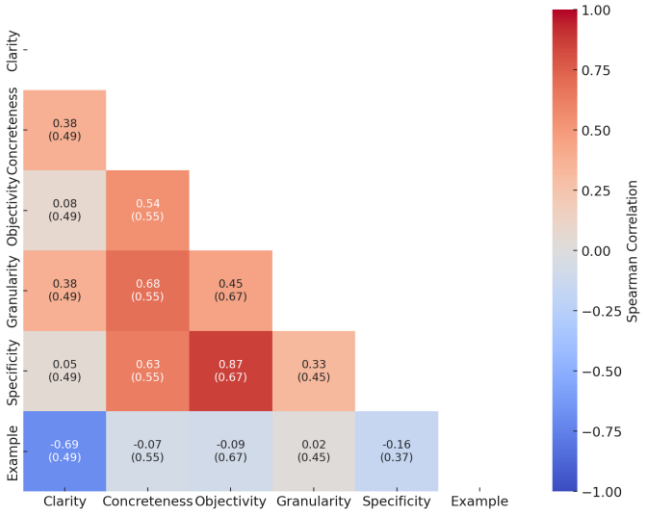

Figure 1. Spearman correlation coefficients across dimensions.
Standard deviations are provided in parentheses below each correlation coefficient.

We next calculate Spearman correlations between the average values for the dimensions and the κ values achieved by GPT-4 in coding tasks for each of the three methods (Table 5). The positive relationship between performance in Zero-shot coding and construct clarity (Spearman coefficient of 0.50) demonstrates the value of clear and unambiguous definitions when employing this approach. Interestingly, this correlation decreases for Few-shot with context (0.40) and becomes negative for the Few-shot approach (-0.24), which speaks to the ability of GPT to identify patterns that humans find harder to define but can find examples of. There were positive correlations between concreteness and performance for Few-shot (0.39) and Few-shot with context (0.55), possibly indicating that Concreteness is best leveraged by GPT when concrete examples or context are available. 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 therefore only 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 Few-shot with context (-0.33). It is possible that the additional context could be overwhelming GPT, causing it to rely less on the examples and more on the surrounding information.

Table 5. 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
Few-shot with context	0.40	0.55	-0.05	-0.12	0.17	-0.33

3. Study 2: Middle-School Students In-Game Astronomy Observations

3.1. Dataset

The Study 2 dataset consists of scientific observations made by students while exploring educational worlds in Minecraft. These observations were obtained from What-if Hypothetical Implementations in Minecraft (WHIMC; Lane et al., 2022), where learners explore scenarios (e.g., “What if Earth had no moon?” or “What if the sun was cooler?”) during informal settings like summer camps. The WHIMC server includes a NASA-inspired launch site, a lunar base, a space station, a Mars map with real Martian terrain data, various known exoplanets, and phenomena such as black holes and quasars. Learners are assisted by automated pedagogical agents and human facilitators to use scientific tools to measure critical habitability factors — temperature, air pressure, radiation, gravity, and atmospheric composition — and write descriptive, comparative, and inferential observations that assess the habitability of each world. Students post their observations in the game space and the observations are visible in real time to other players in their cohort.

The dataset comprises data from 76 learners (49 male, 20 female, and 7 who reported another category or preferred not to answer) collected in 2022. Learners used the system in five locations across three states, with participants drawn from populations living in rural, suburban, and urban areas. They also represented a wide range of racial backgrounds (12 Black/African American, 3 American Indian, 2 Asian/Pacific Islander, 10 Hispanic/Latino, 22 White/Caucasian, 1 who selected multiple categories, 6 other, and 19 who preferred not to answer). Socioeconomic backgrounds also varied considerably between locations, with nearly half of students coming from high-income counties and the rest from areas with mixed or lower-income groups.

3.2. Codebook Development

Prior qualitative research on this learning environment categorized student observations into four categories: *Noun*, *Measure/Descriptive*, *Comparison*, and *Hypothesis* (Yi et al., 2020). These categories were specifically developed to identify and study observations that align with the learning objectives of WHIMC. In our study, we extended this classification framework by applying an inductive thematic analysis to identify additional themes within the data (Thornberg & Charmaz, 2014). Specifically, we introduce six new codes to capture additional aspects of game-related interactions and social communication that were not fully captured within the original coding scheme. The final version of the codebook is shown in Table 6. Since these constructs are not mutually exclusive, some observations may be categorized under multiple labels.

Table 6. Inductively Developed Themes/Constructs

Code Name	Definition/Example
Noun	Definition: Stating nouns without any elaboration. (Previously labelled as "factual" in [Yi et al., 2020].) **Example:** "I see trees."
Measure/ Descriptive	Definition: Related to measures of physical attributes that learners are encouraged to take in each of the different planets and moons they visit, including colour, temperature, quantity, weight or size, radiation, temperature, airflow, pressure, altitude, etc. **Example:** "The temp is -20.6 C, -5.1 F, 252,5 K."
Comparison	Definition: Observations that compare or contrast conditions either (a) among in-game worlds (e.g., two different planets they've been asked to explore) or (b) their real-life experiences on Earth to the in-game worlds. Also includes examples that suggest that their expectations were violated. **Example:** "The grass is greener in the habitable strip."
Hypothesis	Definition: Making hypotheses or guesses, showing speculative thinking, forming conjectures, or making predictions or explanations. **Example:** "This world is probably closer to the sun."
Questioning	Definition: Asking questions about game mechanics or world elements; Seeking to understand the game better, showing curiosity. **Example:** "Why is there no grass?"
Exclamations	Definition: Pure exclamations without any accompanying explanation of observations, including exclamatory grammatical markers or words. **Example:** "Wow!"
Continuing Discussion	Definition: The same user's observations represent the continuation of discussion around a specific topic. **Example:** "Can't find [NAME]." "[NAME] where are you?"
Non-game: True Nonsense	Definition: A sequence of characters, emojis, or symbols repeated excessively, including Random numbers or letters without associated explanations or observations. **Example:** "AAAAAAAAAAAAAAAAAAAAAAA."
Non-game: Unrelated Phrases	Definition: Sentences or phrases unrelated to the purpose of making observations during Minecraft gameplay. **Example:** "This will expire in a week."
Non-game: Out-of-Context Ref.	Definition: References to movies, books, celebrities, etc., without relevance to the game. **Example:** "Subscribe to MrBeast Gaming."

Source: The table includes four Constructs from Yi et al. (2020).

3.3. Automated Coding Process

Two researchers independently coded 200 observations to determine the presence or absence of each construct using predefined definitions. After coding about 100 observations each, they checked inter-rater reliability (IRR). Constructs for which human coders had low agreement were discussed before coding the remaining data. Upon completing the 200 observations, IRR was checked again. The two human coders resolved any discrepancies through social moderation (Herrenkohl & Cornelius, 2013) before evaluating the performance of GPT for coding each construct, mirroring the approach used in Study 1.

For coding this dataset, we used gpt-4-turbo-2024-04-09 (default hyperparameters and temperature=0). We coded the entire dataset three times and calculated the average performance metrics. We used the same Zero-shot and Few-shot prompt approaches as in Study 1. In this case, we generally did not leverage the Few-shot with context approach because consecutive observations posted in WHIMC are not necessarily linked, and human coders noted that the construct can be coded independently of the surrounding context.

However, an exception was made for the construct *Continuing Discussion*, which identifies the occurrence of two consecutive thematically related observations that occur near each other. For coding this construct, we adopted the coding with context approach by 1) providing context (basic information about the game and the preceding observation made by the same student) when coding the current observation, and 2) adding three paired (previous and current) examples along with the context.

Given the potential of embeddings for calculating semantic similarities (Alvarez & Bast, 2017), we also explored the use of OpenAI's text embedding model (text-embedding-3-small) in coding instances for *Continuing Discussion*. Embedding is a process that converts words, phrases, or larger texts into numerical vectors that can be compared. Each observation was first converted into embeddings using OpenAI's text embedding model. Then, we computed the cosine similarity in the spatial domain between the current observation's embedding and the embedding of the previous line, treating similarity scores greater than 0.6 as positive examples. This cutoff was selected as it yielded the best performance based on initial experiments conducted on a subset of the data (50%; 100 lines). The use of embeddings was limited to this construct because *Continuing Discussion* specifically required identifying the relationship between two consecutive student posts. Embeddings are well suited for capturing semantic relationships beyond surface-level matching of words. In other words, even if students use slightly different phrasing across observations, embeddings can detect the underlying thematic connection based on meaning rather than exact wording.

3.4. Results

Table 7 presents the Kappa scores between two human coders prior to resolving disagreements through social moderation, as well as the performance metrics — Kappa, Precision, and Recall — for each coding category, comparing GPT's coding to human coding. Kappa scores between GPT and human coders ranged from 0.72 to 0.95 — consistently higher than in Study 1. The Zero-shot approach yielded high inter-rater reliability (IRR) for constructs where the human coders also had high initial agreement before discrepancies were addressed, such as *Questioning*, *Exclamation*, and *True Nonsense*. Conversely, for nuanced constructs that elicited more coder disagreements (e.g., *Comparison*, *Measure/Descriptive*, and *Unrelated Phrases*) the Few-shot approach was more effective. For the construct *Continuing Discussion*, where it was necessary to assess multiple lines at once, using a text embedding model led to better performance than the Context Only or Few-shot with context approaches. These findings align with those in Study 1: examples were likely to improve coding outcomes for complex constructs without a sufficiently clear definition, and additional context was required only for the construct that needed more information beyond the target line to be accurately coded.

Table 7. Performance Metrics for Each Automated Model

Construct	Freq.	Hum–Hum κ	Method	Hum–GPT κ	Hum–GPT Prec.	Hum–GPT Recall
Noun	17%	0.85	**Zero-shot**	**0.84**	**0.88**	**0.83**
			Few-shot	0.77	0.71	0.92
Measure/ Descriptive	36%	0.80	Zero-shot	0.74	0.88	0.77
			Few-shot	**0.78**	**0.88**	**0.83**
Comparison	14%	0.73	Zero-shot	0.69	0.84	0.64
			Few-shot	**0.74**	**0.74**	**0.79**
Questioning	9%	0.96	**Zero-shot**	**0.95**	**0.90**	**1.00**
			Few-shot	0.93	0.88	1.00
Hypotheses	6%	0.73	**Zero-shot**	**0.77**	**0.81**	**0.76**
			Few-shot	0.69	0.69	0.77
Exclamation	6%	0.95	**Zero-shot**	**0.86**	**0.81**	**0.96**
			Few-shot	0.76	0.70	0.88
Continuing Discussion	13%	0.88	Context only	0.85	0.91	0.83
			Few-shot with context	0.88	0.88	0.91
			Embedding	**0.93**	**0.97**	**0.95**
Non-game: True Nonsense	4%	0.97	**Zero-shot**	**0.95**	**0.94**	**0.97**
			Few-shot	0.85	0.82	0.90
Non-game: Unrelated Phrases	7%	0.75	Zero-shot	0.77	0.82	0.75
			Few-shot	**0.77**	**0.82**	**0.91**
Non-game: Out-of-Context Reference	3%	0.88	Zero-shot	0.82	0.88	0.78
			Few-shot	**0.86**	**0.93**	**0.81**

Note: The best coding method (highlighted/in bold) for each construct is selected if it has the highest Kappa among all coding methods and a minimum κ≥0.70.

4. Study 3: Computer Science Students Programming Code

4.1. Dataset

The dataset for Study 3 consisted of practice assignments submitted through the automated assessment platform RunCode (Pankiewicz & Furmańczyk, 2020). Students in this study were learning C# in an introductory computer science course in Poland during the fall 2022 semester. Using RunCode was optional and did not affect the final course grade. In this semester, 169 students actively used this platform, submitting code 44,448 times. Each submission was immediately evaluated, and students received feedback on compiler errors or failed unit tests, aiding them in refining their submissions until they achieved completely correct answers. The submissions spanned across 146 tasks covering four fundamental programming topics: types and variables (33 tasks), conditional statements (25 tasks), recursion (28 tasks), and arrays and loops (60 tasks).

4.2. Codebook Development

Previous research has developed qualitative codebooks to explore the debugging behaviours of programming learners, comparing two consecutive submissions of code for the same task (Pinto et al., 2023; Zambrano et al., 2024). Inspired by this prior research and the codebooks they proposed, we investigated GPT's capabilities for qualitatively coding not only natural language but also programming code. We specifically focused on submissions made on conditional statement tasks by students who self-reported having little-to-no programming experience prior to joining the course. Conditional statements are the first module in the semester that requires a basic understanding of both syntax and programming logic and has been previously identified as one of the earliest topics where differences between high- and low-performing students become apparent (Zambrano et al., 2024; Izu et al., 2022). Based on this specific subset of programming code submissions, we refined the codebooks presented in Pinto et al. (2023) and Zambrano et al. (2024) to consider specific constructs associated with conditional statement tasks. This inductively developed codebook, built based on earlier inductively developed codebooks, is presented in Table 8.

Table 8. Codebook with Definitions

Construct	Definitions & Examples
If Header	Modifications to the if condition/header.
If Body	Modifications to the lines enabled by the if condition/header.
Function Return	Modifications inside the return statement.
Function Body	Modifications inside the body of the function. These modifications include adding more conditional statements, auxiliary variables, and others.
Comment	A new commented line or a deletion or modification of an already existing comment.
Testing	Modifications inside the Main function (section of the code used for testing), such as adding a line to print results in the console and testing the correct functioning of their code.
Added Lines	Contains at least one completely new code line.
Removed Lines	Student removed code lines in the submission.
Variable Usage & Assignment	Student submission adds a new variable or deletes or modifies the value assignment of an already existing variable.
Variable-type Change	A modification of the type of variable on its initial declaration.
Variable-type Conversion Change	Modification in the conversion of the type of variable after its initial declaration or a conversion in the type of variable obtained after using an already existing method.
Value Change	Modification of any value. It can be in the if header/condition, in the coefficient in an equation, or in the assignment of a variable.
Operator	Modification of an operator, such as changing the "greater than" operator to "equal to" in a conditional statement.
Syntax Change	A modification in the syntax of a code line to correct a compiler error.

Source: Refined from work by Pinto et al. (2023) and Zambrano et al. (2024).

4.3. Automated Coding Process

For the automated coding of the submissions, we used GPT-4 (gpt-4-turbo-2024-04-09) to develop binary classifiers for each construct, again using default hyperparameters and a temperature of 0. We used a Zero-shot prompt as in the previous two studies. We also used a Few-shot prompt, but in this study our Few-shot prompt both included a positive example that aligns with the target construct and a negative example that does not. This pair of examples were added because, in most cases,

providing only a single positive example caused GPT to overgeneralize the construct and confuse it with other constructs that might seem similar (e.g., two "if" statements where one has a changed condition and the other has a changed return, both still having a significant overlap in their code). We crafted the positive and negative examples to be similar to each other, to clarify for GPT how to distinguish between these lines that, despite their similarity, correspond to different constructs. As with the use of these two methods in Studies 1 and 2, we provided GPT-4 only with the lines showing differences between two consecutive submissions rather than the full submissions, which often spanned more than 50 lines. The Few-shot with context approach was not used, as key context could appear far from the line of interest, meaning that the entire submission would include irrelevant context and could make it difficult for GPT to focus on the specific line where the change happened.

4.4. Results

Table 9 presents the level of agreement between human coders and GPT's performance for each construct examined in this study. Although experienced programmers should find most of these constructs straightforward to identify in programming data, categorizing the intentions behind these changes can be challenging since novices are more likely to introduce syntax errors or make changes in unexpected sections of the code. Our findings indicate that GPT's performance was highly related to human-to-human inter-rater reliability. When inter-rater reliability between the two human coders was low ($\kappa<0.60$; *If Body*, *Variable Usage & Assignment*, *Variable-type Change*, *Syntax Change*), GPT also had difficulty. Although recall was around 0.8 for three of these four constructs, precision was low, suggesting that GPT may be overgeneralizing.

Table 9. Performance Metrics for Each Automated Model

Construct	Freq.	Hum–Hum κ	Method	Hum–GPT κ	Hum–GPT Prec.	Hum–GPT Recall
If Header	36%	0.96	Zero-shot	0.78	0.93	0.78
			Few-shot	**0.80**	**0.90**	**0.85**
If Body	6%	0.48	Zero-shot	0.12	0.14	0.39
			Few-shot	0.12	0.14	0.33
Function Return	22%	0.77	**Zero-shot**	**0.76**	**0.73**	**0.95**
			Few-shot	0.66	0.81	0.67
Function Body	20%	0.79	Zero-shot	0.10	0.25	0.66
			Few-shot	0.08	0.24	0.64
Comment	2%	1.00	**Zero-shot**	**0.80**	**0.67**	**1.00**
			Few-shot	0.66	0.50	1.00
Testing	28%	0.94	Zero-shot	0.54	0.90	0.48
			Few-shot	0.31	0.88	0.25
Added Lines	11%	0.85	**Zero-shot**	**0.93**	**0.88**	**1.00**
			Few-shot	**0.93**	**0.88**	**1.00**
Removed Lines	9%	0.94	Zero-shot	0.71	0.67	0.84
			Few-shot	**0.76**	**0.72**	**0.84**
Variable Usage & Assignment	16%	0.49	Zero-shot	0.30	0.33	0.84
			Few-shot	0.25	0.29	0.82
Variable-type Change	3%	0.32	Zero-shot	0.45	0.40	0.80
			Few-shot	0.29	0.22	0.80
Variable-type Conversion	9%	0.64	Zero-shot	0.57	0.47	0.96
			Few-shot	0.55	0.47	0.88
Value Change	10%	0.63	Zero-shot	0.40	0.38	0.62
			Few-shot	0.34	0.30	0.85
Operator	30%	0.86	**Zero-shot**	**0.73**	**0.75**	**0.90**
			Few-shot	0.62	0.73	0.73
Syntax Change	29%	0.55	Zero-shot	0.49	0.58	0.79
			Few-shot	0.41	0.51	0.80

Note: The coding method (highlighted/in bold) with the highest Kappa is selected if it meets a minimum threshold of $\kappa \geq 0.70$.

On the other hand, for constructs where both human coders achieved higher levels of agreement ($\kappa \geq 0.70$), GPT also performed better ($\kappa \geq 0.70$). GPT was successful at identifying many constructs related to specific (less ambiguous) locations

within student code (i.e., *If Header*, *Function Return*, and *Comment*). GPT also accurately identified other constructs that depend solely on the specific line being modified, such as *Adding Lines*, *Removing Lines*, and *Operator*. However, when location-related constructs required additional context (i.e., the entire function or method where the change is embedded), GPT performed more poorly (i.e., location-related constructs like *Function Body*, *If Body*, or *Testing* (Main body).

In most cases where GPT was able to successfully code the constructs ($\kappa \geq 0.70$), the Zero-shot approach outperformed the Few-shot approach. The poorer performance for the Few-shot approach can possibly be attributed to the diverse range of modifications possible in programming code, which cannot be fully captured by just a few examples. This limitation of the Few-shot approach becomes even more pronounced with programming novices, who might introduce changes that are entirely unexpected or that do not conform to the standard syntax or logic of the intended code or, indeed, the programming language at all. However, for some constructs less subject to variability or interpretation, such as *Added Lines* and modifying the *If Header*, adding examples appeared to slightly improve GPT's performance.

5. Discussion and Conclusion

This article investigated the use of GPT-4 for automated qualitative coding across three educational datasets: Algebra I tutoring session transcripts (from Study 1), scientific observations made by students in the WHIMC Minecraft environment (from Study 2), and debugging behaviours in introductory programming code submissions (from Study 3). In each of these approaches, we took a codebook that was initially inductively developed and refined. Then we applied these codes deductively across the entire dataset, using them to identify instances of each theme (binary presence/absence). This approach allowed us to apply our inductively derived insights consistently to the rest of the data.

Across these datasets, we tested four prompt engineering approaches: 1) Zero-shot coding, which presents only the construct definition to GPT and prompts it to code, 2) Few-shot coding (annotated examples along with the construct definition) with only positive examples, 3) Few-shot coding with both positive and negative examples, 4) Few-shot with context, which provides GPT with some context of the study and the preceding lines to aid in coding the current line, and — for just one construct in one data set — 5) Embeddings, where we used OpenAI's tool for converting text into numerical vectors and then compared the current student observation to their previous observation.

Across all three studies, the GPT-4 API achieved good agreement with human coders ($\kappa \geq 0.70$ for 25 out of the 34 constructs) for at least one of these prompt engineering approaches. This finding indicates GPT-4'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. Specifically, across these different contexts and data sources, we observed that Zero-shot prompting can achieve high performance for well-defined constructs — like *Greetings* in Study 1, *Noun* in Study 2, and *Comment* in Study 3 — with straightforward and easily comprehensible definitions. However, Zero-shot coding tends to miss many cases, achieving lower recall than other methods. Moreover, similar to findings in Amarasinghe et al. (2023) and Theelen et al. (2024), the absence of contextual understanding and reliance on strict definitions limit the Zero-shot approach's effectiveness for nuanced or context-dependent constructs. For example, while constructs like *Greetings* were coded reliably in Study 1, those requiring contextual understanding, such as *Direct Instruction*, were not. This finding highlights the need for qualitative codebook development to prioritize clarity and concreteness, if Zero-shot coding will be used.

Incorporating annotated examples (Few-shot prompting) improved performance for some of the more complex constructs, such as *Software* in Study 1, *Out-of-Context References* in Study 2, and *Removed Lines* in Study 3. However, the use of examples also led to overgeneralization in some cases. For instance, in Study 3, novice programmers made a range of choices, many of them unexpected, leading to overgeneralization when using few-shot approaches. In these cases, more straightforward Zero-shot prompting often performed better. Misclassification issues also arose when examples were not carefully selected, which demonstrates the need for examples to be representative of the span of cases, and also the value of selecting examples that precisely differentiate the category of interest from other categories. We also found that incorporating explicit non-examples when coding with GPT improved coding precision in some cases, mitigating overgeneralization.

Perhaps not surprisingly, Few-shot with context is more accurate for constructs that require an understanding of the surrounding context or when lines of data have temporal relationships. However, the approach also led to issues when the context lines involved different constructs than the current line being coded (i.e., when lines of data have subject changes or are not connected). Thus, while context can enhance understanding, it must be selected thoughtfully to avoid introducing noise.

Additionally, all methods struggled with constructs that have a lower level of concreteness, such as *Clarification* in Study 1 and *Syntax Change* in Study 3. These constructs were generally not the most difficult for humans to code (κ=0.72 and κ=0.55, respectively), suggesting that human reasoning is able to identify fewer concrete constructs, which remains difficult for GPT-4. This represents a deviation from the more general overall trend, where the hardest constructs for humans to code were also the hardest for GPT-4.

Our research provides evidence regarding the advantages and limitations of different approaches when using GPT-4 for coding tasks. Each method offers distinct benefits that can be leveraged depending on the nature of the constructs being coded. Researchers should therefore consider the nature of their constructs when choosing a prompting method for automated coding. Careful selection of an approach could maximize the benefits of GPT in qualitative research by producing more accurate data coding.

One limitation of using the GPT-4 model through the OpenAI API — as in our work — relative to using ChatGPT is that the API is not as effective as ChatGPT at providing explanations for its decisions, identifying ambiguity in construct definitions, or discussing inconsistencies in human coding as ChatGPT (e.g., Zambrano et al., 2023; Barany et al., 2024), which is specifically designed for interaction, conversation, and iteration. 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.

There is still much to do. Each method explored in this paper could be investigated in finer-grained detail. For example, it may be relevant for some datasets to separate out task context from discourse context, and perhaps use one or the other but not both. Another potential area for future work is the exploration of coding multiple constructs at once using a single prompt. While this study focused on binary classification to reduce complexity, it is also possible to allow the model to select between a range of mutually exclusive codes all at once, or to assign multiple labels where constructs intersect or overlap within the same line. Approaches of this nature could streamline coding workflows and eliminate the need to code the same dataset multiple times for different constructs, though it is unclear whether the greater complexity could confuse an LLM or lead to it focus on the first or last code in the set. Future research should also investigate further strategies to improve performance for complex, ambiguous, or subtle constructs while also refining the coding process to increase adaptability to different research needs. More broadly, future work will need to investigate how different LLMs, such as Claude or LLaMA (as well as future versions of OpenAI's offerings) can be optimally used for different forms of qualitative coding. Finally, future research should explore the applicability of these methods to more fully deductive coding processes, using LLMs to develop coding schemes directly from theoretical models and frameworks.

By leveraging the strengths of GPT-4, and LLMs in general, educational researchers can streamline the coding process, enabling more efficient and comprehensive analysis of qualitative data. Ultimately, informed selection and tailoring the approach to the data context and code type has potential to improve the accuracy and reliability of LLMs, better positioning tools such as GPT to serve as reliable "co-researchers" that can strengthen the trustworthiness of findings in qualitative data analysis. As the capabilities of large language models continue to advance, so too will their applications in automated qualitative coding, making it possible to conduct these methods faster and ultimately better.

Data Availability

To promote transparency, reproducibility, and further exploration, the raw, de-identified data from Study 3, along with their corresponding coded versions, are available for sharing with the research community through this link.

Declaration of Conflicting Interest

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

We would like to acknowledge funding support for Study 1 (LEVI Institute), Study 2 (NSF # DRL2301173), and Study 3 (Penn Center for Learning Analytics).

References

Alvarez, J. E., & Bast, H. (2017). *A review of word embedding and document similarity algorithms applied to academic text* [Unpublished bachelor's thesis]. University of Freiburg.

Amarasinghe, I., Marques, F., Ortiz-Beltrán, A., & Hernández-Leo, D. (2023). Generative pre-trained transformers for coding text data? An analysis with classroom orchestration data. In O. Viberg, I. Jivet, P. J, Munoz-Merino, M. Perifanou, & T. Papathoma (Eds.), *Responsive and sustainable educational futures: 18th European Conference on Technology Enhanced Learning, EC-TEL 2023, Aveiro, Portugal, September 4–8, 2023, proceedings* (pp. 32–43). Springer Cham. http://dx.doi.org/10.1007/978-3-031-42682-7_3

Asudani, D. S., Nagwani, N. K., & Singh, P. (2023). Impact of word embedding models on text analytics in deep learning environment: A review. *Artificial Intelligence Review, 56*(9), 10345–10425. http://dx.doi.org/10.1007/s10462-023-10419-1

Barany, A., Nasiar, N., Porter, C., Zambrano, A. F., Andres, A. L., Bright, D., Shah, M., Liu, X., Gao, S., Zhang, J., Mehta, S., Choi, J., Giordano, C., & Baker, R. S. (2024). ChatGPT for education research: Exploring the potential of large language models for qualitative codebook development. In A. M. Olney, I.-A. Chouta, Z. Liu, O. C. Santos, & I. I. Bittencourt (Eds.), *Artificial intelligence in education: 25th international conference, AIED 2024, Recife, Brazil, July 8–12, 2024, proceedings, part II* (pp. 134–149). Springer Cham. http://dx.doi.org/10.1007/978-3-031-64299-9_10

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language models are few-shot learners*. arXiv. https://doi.org/10.48550/arXiv.2005.14165

Cai, Z., Siebert-Evenstone, A., Eagan, B., Shaffer, D. W., Hu, X., & Graesser, A. C. (2019). nCoder+: A semantic tool for improving recall of nCoder coding. In B. Eagan, M. Misfeldt, & A. Siebert-Evenstone (Eds.), *Advances in quantitative ethnography: First international conference, ICQE 2019, Madison, WI, USA, October 20–22, 2019, proceedings* (pp. 41–54). Springer. http://dx.doi.org/10.1007/978-3-030-33232-7_4

Chew, R., Bollenbacher, J., Wenger, M., Speer, J., & Kim, A. (2023). *LLM-assisted content analysis: Using large language models to support deductive coding*. arXiv. https://doi.org/10.48550/arXiv.2306.14924

Crowston, K., Liu, X., & Allen, E. E. (2010). Machine learning and rule-based automated coding of qualitative data. In *Proceedings of the American Society for Information Science and Technology, 47*(1), 1–2. http://dx.doi.org/10.1002/meet.14504701328

Ekin, S. (2023). *Prompt engineering for ChatGPT: A quick guide to techniques, tips, and best practices.* TechRxiv. https://doi.org/10.36227/techrxiv.22683919.v2

Femepid, S., Hatherleigh, L., & Kensington, W. (2024). *Gradual improvement of contextual understanding in large language models via reverse prompt engineering*. Authorea. https://doi.org/10.22541/au.172376001.14254079/v1

Gao, J., Choo, K. T. W., Cao, J., Lee, R. K.-W., & Perrault, S. (2023). CoAIcoder: Examining the effectiveness of AI-assisted human-to-human collaboration in qualitative analysis. *ACM Transactions on Computer–Human Interaction, 31*(1), Article 6. http://dx.doi.org/10.1145/3617362

Giray, L. (2023). Prompt engineering with ChatGPT: A guide for academic writers. *Annals of Biomedical Engineering, 51*(12), 2629–2633. http://dx.doi.org/10.1007/s10439-023-03272-4

Herrenkohl, L. R., & Cornelius, L. (2013). Investigating elementary students' scientific and historical argumentation. *Journal of the Learning Sciences, 22*(3), 413–461. http://dx.doi.org/10.1080/10508406.2013.799475

Hopkins, D. J., & King, G. (2010). A method of automated nonparametric content analysis for social science. *American Journal of Political Science, 54*(1), 229–247. http://dx.doi.org/10.1111/j.1540-5907.2009.00428.x

Hou, C., Zhu, G., Zheng, J., Zhang, L., Huang, X., Zhong, T., Li, S., Du, H., & Ker, C. L. (2024). Prompt-based and fine-tuned GPT models for context-dependent and -independent deductive coding in social annotation. In B. Flanagan, B. Wasson, & D. Gašević (Eds.), *LAK '24: Proceedings of the 14th Learning Analytics & Knowledge Conference* (pp. 518–528). ACM Press. http://dx.doi.org/10.1145/3636555.3636910

Hutt, S., DePiro, A., Wang, J., Rhodes, S., Baker, R. S., Hieb, G., Sethuraman, S., Ocumpaugh, J., & Mills, C. (2024). Feedback on feedback: Comparing classic natural language processing and generative AI to evaluate peer feedback. In B. Flanagan, B. Wasson, & D. Gašević (Eds.), *LAK '24: Proceedings of the 14th Learning Analytics & Knowledge Conference* (pp. 55–65). ACM Press. http://dx.doi.org/10.1145/3636555.3636850

Izu, C., Denny, P., & Roy, S. (2022). A resource to support novices refactoring conditional statements. In B. A. Becker, K. Quille, M.-J. Laakso, E. Barendsen, & Simon (Eds.), *ITiCSE '22: Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education* (Vol. 1, pp. 344–350). ACM Press. http://dx.doi.org/10.1145/3502718.3524810

Katz, A., Fleming, G. C., & Main, J. (2024). *Thematic analysis with open-source generative AI and machine learning: A new method for inductive qualitative codebook development.* arXiv. https://doi.org/10.48550/arXiv.2410.03721

Kirsten, E., Buckmann, A., Mhaidli, A., & Becker, S. (2024). *Decoding complexity: Exploring human–AI concordance in qualitative coding*. arXiv. https://doi.org/10.48550/arXiv.2403.06607

Kovanović, V., Joksimović, S., Waters, Z., Gašević, D., Kitto, K., Hatala, M., & Siemens, G. (2016). Towards automated content analysis of discussion transcripts: A cognitive presence case. In D. Gašević, G. Lynch, S. Dawson, H. Drachsler, & C. Penstein Rosé (Eds.), *LAK '16: Proceedings of the 6th International Conference on Learning Analytics & Knowledge* (pp. 15–24). ACM Press. http://dx.doi.org/10.1145/2883851.2883950

Lane, H. C., Gadbury, M., Ginger, J., Yi, S., Comins, N., Henhapl, J., & Rivera-Rogers, A. (2022). Triggering STEM interest with Minecraft in a hybrid summer camp. *Technology, Mind, and Behavior, 3*(4). http://dx.doi.org/10.1037/tmb0000077

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys, 55*(9), Article 195. http://dx.doi.org/10.1145/3560815

Lo, L. S. (2023a). The art and science of prompt engineering: A new literacy in the information age. *Internet Reference Services Quarterly, 27*(4), 203–210. http://dx.doi.org/10.1080/10875301.2023.2227621

Lo, L. S. (2023b). The CLEAR path: A framework for enhancing information literacy through prompt engineering. *The Journal of Academic Librarianship, 49*(4), Article 102720. http://dx.doi.org/10.1016/j.acalib.2023.102720

Morgan, D. L. (2023). Exploring the use of artificial intelligence for qualitative data analysis: The case of ChatGPT. *International Journal of Qualitative Methods, 22*. http://dx.doi.org/10.1177/16094069231211248

Nunez-Mir, G. C., Iannone, B. V., III., Pijanowski, B. C., Kong, N., & Fei, S. (2016). Automated content analysis: Addressing the big literature challenge in ecology and evolution. *Methods in Ecology and Evolution, 7*(11), 1262–1272. http://dx.doi.org/10.1111/2041-210X.12602

OpenAI. (2022). *ChatGPT* [Large language model]. https://openai.com/chatgpt

Pankiewicz, M., & Furmańczyk, K. (2020). From zero to hero: Automated formative assessment for supporting student engagement and performance in a gamified online programming course. *EdMedia + Innovate Learning, 2020*(1), 1252–1261.

Pinto, J. D., Liu, Q., Paquette, L., Zhang, Y., & Fan, A. X. (2023). Investigating the relationship between programming experience and debugging behaviors in an introductory computer science course. In G. A. Irgens & S. Knight (Eds.), *Advances in quantitative ethnography: 5th international conference, ICQE 2023, Melbourne, VIC, Australia, October 8–12, 2023, proceedings* (pp. 125–139). Springer Cham. http://dx.doi.org/10.1007/978-3-031-47014-1_9

Prabhumoye, S., Kocielnik, R., Shoeybi, M., Anandkumar, A., & Catanzaro, B. (2021). *Few-shot instruction prompts for pretrained language models to detect social biases.* arXiv. https://doi.org/10.48550/arXiv.2112.07868

Saldaña, J. (2016). *The coding manual for qualitative* researchers (3rd ed.). SAGE Publications.

Shaffer, D. W., & Ruis, A. R. (2021). How we code. In A. R. Ruis & S. B. Lee (Eds.), *Advances in quantitative ethnography: Second international conference, ICQE 2020, Malibu, CA, USA, February 1–3, 2021, proceedings* (pp. 62–77). Springer Cham. http://dx.doi.org/10.1007/978-3-030-67788-6_5

Shapiro, G. (1997). The future of coders: Human judgments in a world of sophisticated software. In C. W. Roberts (Ed.), *Text analysis for the social sciences: Methods for drawing statistical inferences from texts and transcripts* (pp. 225–238). Routledge. http://dx.doi.org/10.4324/9781003064060-16

Sherin, B. (2012). Using computational methods to discover student science conceptions in interview data. In S. Dawson, C. Haythornthwaite, S. Buckingham Shum, D. Gašević, & R. Ferguson (Eds.), *LAK '12: Proceedings of the 2nd International Conference on Learning Analytics & Knowledge* (pp. 188–197). ACM Press. http://dx.doi.org/10.1145/2330601.2330649

Spasić, A. J., & Janković, D. S. (2023). Using ChatGPT standard prompt engineering techniques in lesson preparation: Role, instructions and seed-word prompts. In N. Dončov, Z. Ž. Stanković, & B. P. Stošić (Eds.), *2023 58th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST): Proceedings of Papers* (pp. 47–50). IEEE. http://dx.doi.org/10.1109/ICEST58410.2023.10187269

Tai, R. H., Bentley, L. R., Xia, X., Sitt, J. M., Fankhauser, S. C., Chicas-Mosier, A. M., & Monteith, B. G. (2024). An examination of the use of large language models to aid analysis of textual data. *International Journal of Qualitative Methods, 23*. http://dx.doi.org/10.1177/16094069241231168

Theelen, H., Vreuls, J., & Rutten, J. (2024). Doing research with help from ChatGPT: Promising examples for coding and inter-rater reliability. *International Journal of Technology in Education, 7*(1), 1–18. http://dx.doi.org/10.46328/ijte.537

Thornberg, R., & Charmaz, K. (2014). Grounded theory and theoretical coding. In U. Flick (Ed.), *The SAGE handbook of qualitative data analysis*, 153–169. http://dx.doi.org/10.4135/9781446282243.n11

Weber, R. P. (1984). Computer-aided content analysis: A short primer. *Qualitative Sociology, 7*(1), 126–147. https://doi.org/10.1007/BF00987112

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). *A prompt pattern catalog to enhance prompt engineering with ChatGPT.* arXiv. https://doi.org/10.48550/arXiv.2302.11382

Xiao, Z., Yuan, X., Liao, Q. V., Abdelghani, R., & Oudeyer, P.-Y. (2023). Supporting qualitative analysis with large language models: Combining codebook with GPT-3 for deductive coding. In F. Chen, M. Billinghurst, M. Zhou, & S. Berkovsky (Eds.), *IUI '23 companion: Companion proceedings of the 28th International Conference on Intelligent User Interfaces* (pp. 75–78). ACM Press. https://doi.org/10.1145/3581754.3584136

Yi, S., Gadbury, M., & Lane, H. C. (2020). Coding and analyzing scientific observations from middle school students in Minecraft. In M. Gresalfi & I. S. Horn (Eds.), *The interdisciplinarity of the learning sciences: 14th International Conference of the Learning Sciences (ICLS) 2020* (Vol. 3, pp. 1787–1788). International Society of the Learning Sciences. https://repository.isls.org//handle/1/6443

Zambrano, A. F., Liu, X., Barany, A., Baker, R. S., Kim, J., & Nasiar, N. (2023). From nCoder to ChatGPT: From automated coding to refining human coding. In G. A. Irgens & S. Knight (Eds.), *Advances in quantitative ethnography: 5th international conference, ICQE 2023, Melbourne, VIC, Australia, October 8–12, 2023, proceedings* (pp. 470–485). Springer Cham. https://doi.org/10.1007/978-3-031-47014-1_32

Zambrano, A. F., Pankiewicz, M., Barany, A., & Baker, R. S. (2024). Ordered network analysis in CS education: Unveiling patterns of success and struggle in automated programming assessment. In M. Monga, V. Lonati, E. Barendsen, J. Sheard, J. Patterson, & L. Barker (Eds.), *ITiCSE 2024: Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education* (Vol. 1, pp. 443–449). ACM Press. http://dx.doi.org/10.1145/3649217.3653613

Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W., Jiang, J., & Cui, B. (2024). *Retrieval-augmented generation for AI-generated content: A survey.* arXiv. https://doi.org/10.48550/arXiv.2402.19473