

Scaffolding Human-LLM Qualitative Coding Through Inference Evaluation and Construct Boundaries

Zhanlan Wei¹[0009-0002-3931-6398], Jiayi Zhang²[0000-0002-7334-4256], Ryan S. Baker³[0000-0002-3051-3232], Conrad Borchers⁴[0000-0003-3437-8979], Amanda Barany¹[0000-0003-2239-2271], Xiner Liu¹[0009-0004-3796-2251]

¹ University of Pennsylvania, Philadelphia, PA 19104, USA

² Worcester Polytechnic Institute, Worcester, MA 01609, USA

³ Adelaide University, Adelaide, SA 5005, Australia

⁴ Vanderbilt University, Nashville, TN 37235, USA
zhanlanw@upenn.edu

Qualitative coding is key to quantitative ethnography (QE), supporting researchers in interpreting and making sense of rich textual data. However, traditional manual coding can be time-, labor-, and resource-intensive, making it difficult to use on large datasets and limiting its wider application. Recent advances in large language models (LLMs) have led researchers to increasingly explore their use in qualitative coding. Yet prior work reports mixed human-model alignment across constructs: while LLMs align closely with human coders for some constructs, their decisions diverge for others. Aiming to understand what contributes to this coding inconsistency, we examine the *degree of inference*—the extent to which applying a code requires interpretation beyond what is explicitly stated in the text—as a key dimension associated with LLM coding performance. Using student peer-feedback data coded by both human coders and GPT-4o, we investigate how human-model agreement differs across constructs with varying inference demands. Our findings show that GPT-4o achieves stronger alignment with human coders on low-inference constructs, while high-inference constructs remain more challenging. To improve LLM coding performance on constructs with initially low agreement, we further evaluate a scaffolding strategy for LLMs—adding *boundaries* to prompts that specify codes' inclusion and exclusion criteria. The results indicate that adding code boundaries during LLM prompting can meaningfully improve human-model alignment for the high-inference constructs (where GPT initially struggles). Together, these findings offer practical guidance for identifying what makes certain constructs suitable for LLM-based auto-coding, and for designing prompts that better scaffold the model and support human-model collaboration.

Keywords: Qualitative Coding, Large Language Models, Inference, Prompt Engineering, Human-Model Alignment, Construct Boundaries.

1 Introduction and Related Works

Qualitative coding is a foundational step in many qualitative research workflows, through which researchers systematically label, organize, and interpret meaning from data in terms of specific constructs [1]. Because discourse data are inherently dense and time-consuming to interpret, coding helps organize the data and make it more tractable to study patterns [2]. In quantitative ethnography (QE), qualitative coding serves as an important bridge between observed phenomena and meaningful interpretations. In other words, coding is the process of distilling meaning from data into interpretable theoretical claims [3, 4]. It also serves as a prerequisite for identifying interaction patterns before modeling their structural relationships using statistical methods (e.g., epistemic network analysis; ENA [5]). Prior QE work has applied qualitative coding across a range of research contexts, including educational settings (e.g., analyzing knowledge co-construction patterns [6]), media discourse (e.g., media coverage during the COVID-19 pandemic [7]), and clinical environments (e.g., patients' decision-making regarding treatment choices [8]). Across these studies, coding supports the transformation of qualitative data into structured forms that enable the systematic analysis of relationships and serve as the basis upon which more in-depth interpretations are built.

In QE, coding is often conducted manually by human researchers, a process that can be time-consuming, labor-intensive, and costly. These challenges are especially evident when working with large-scale datasets (e.g., over 200 hours of classroom recordings; large corpora of policy documents). As a result, the practical demands of qualitative coding have limited its usefulness within large-scale educational research. In response to these challenges, recent years have seen growing interest in exploring how large language models (LLMs) can assist with qualitative analysis (e.g., [9,10]). For instance, Morgan [11] compared ChatGPT with human researchers in identifying themes within the same dataset, and found that ChatGPT completed the task in just 2 hours, compared to the 23 hours typically required by human coders. Other studies further suggest that, with appropriate prompting, LLMs can meet the same quality standards comparable to those of human researchers in qualitative analysis, including high inter-rater reliability (IRR) with human coders [12], strong alignment with theory [13], and the ability to generate clear and comprehensive constructs [14].

Despite the affordances LLMs bring to qualitative coding, their applicability remains uneven. Prior work has found that LLM-based auto-coding exhibits inconsistent coding performance across constructs; in some cases, the models align closely with human coders, while in others, their performance falls short. For example, Liu et al. [10] assessed GPT-4 Turbo's coding performance across constructs of varying complexity and found stronger alignment with human coders on well-defined, objective constructs, while struggling with more abstract constructs requiring contextual information. These findings demonstrated inconsistencies in how the model recognizes and applies different constructs. To address these coding inconsistencies, researchers have explored whether prompt design can influence coding performance. Borse et al [15], for example, tested multiple prompting strategies (e.g., role-prompting; few-shot; chain-of-thought), and hyperparameter settings to assess their impact on the inter-rater reliability

between human coders and LLMs. Similarly, Liu et al. [9] evaluated multiple prompting techniques (e.g., zero-shot, few-shot, coding-with-context, and embeddings) and found that no single prompting technique consistently outperforms the others, with the effectiveness varying depending on the type of construct being coded.

In line with these early explorations of the factors that influence LLMs' coding success, we propose that the *degree of inference*—the extent to which identifying and applying a construct requires interpretation beyond the explicit text—may also play an important role. LLM inference differs fundamentally from human inference; while LLMs can mimic surface features of human reasoning, they often lack the depth and nuance of human cognitive processes [16]. As a result, constructs requiring greater interpretative judgment or abstraction may pose greater challenges for models to approximate human decisions. To explore this possibility, we examine how constructs that require higher and lower inference levels might impact an LLM's ability to match human judgment. We also assess whether adding *boundaries*—codes' inclusion and exclusion criteria—can support LLMs in reasoning about constructs they initially struggle with. This paper thus investigates the following two research questions (RQs):

1. RQ1: How is the *degree of inference* required by a code associated with differences in human-model coding alignment?
2. RQ2: For codes where an LLM struggles to align with human coders, does adding *construct boundaries* help improve human-LLM coding agreement?

1.1 Conceptualizing Degree of Inference and Boundaries

In this paper, we define the *degree of inference* as the extent to which applying a construct requires the coder to interpret or infer meaning beyond what is explicitly stated in the text. In qualitative coding, the level of reasoning and interpretative effort may vary across constructs. While some codes can be applied based on direct, recognizable textual cues (e.g., a student saying, "Thank you!" may indicate the presence of Gratitude in a peer interaction), others require more nuanced interpretative judgment and contextual reasoning. In these cases, researchers must infer underlying meanings that are not explicitly stated, grounding their coding decisions in subtle implicit cues, broader contextual information, and their own interpretive judgment.

Considerations around differences in inferential demand are not new (cf. discussions of coding fairness; e.g., [4]) but may help shed light on why human-LLM coding agreement varies across constructs. Because LLMs do not fully capture the depth and complexity of human reasoning [16] and often rely on surface-level textual cues [17], their coding decisions may diverge from those of human researchers—performing relatively well on low-inference codes tied to easily identifiable linguistic patterns, but struggling with higher-inference codes that require deeper interpretation. Hence, the *degree of inference* required to identify a code in a dataset may serve as a partial explanation of variability in human-model coding alignment. By exploring which types of constructs LLMs tend to apply more consistently and in closer agreement with human coders, we hope to support more informed decisions about when—and for which kinds of codes—LLM-based auto-coding may be more appropriate.

For codes where the model initially struggles to align with human coders (i.e., the high-inference codes we anticipated), we then investigate whether a specific adjustment to prompts—adding code *boundaries*—can serve as an effective scaffolding strategy to improve human-model agreement. We define *boundaries* as the explicit inclusion and exclusion criteria that specify when a code should or should not be applied. *Boundaries* differ from code definitions, which typically describe what the construct is. *Boundaries* go a step further, specifying when—and just as importantly—when not to assign a code, often by drawing lines around conceptually ambiguous instances or edge cases (a practice that may sometimes emerge implicitly through codebook development, e.g., [18]). The goal of creating boundaries is to help LLMs better understand how human coders conceptualize and apply constructs, thereby reducing interpretive ambiguity and improving alignment with human coding decisions. In this way, *boundaries* operationalize coding decisions and offer practical guidance for both humans and LLMs to better navigate interpretative uncertainty (see Section 2.5 for further discussion).

2 Method

2.1 Data Context

The dataset used in this study was collected from CueThink, a digital learning platform designed to engage middle school students in self-regulated learning and support the development of mathematical communication skills [19]. This platform was deployed in a southwestern region of the United States and made available to middle school students during the regular school year. In CueThink, mathematical problems are organized into scaffolded activities called Thinklets. Thinklets are divided into different phases that guide students through the problem-solving process, prompting them to generate responses and reflect on how they approached the mathematical task. As students work through math problems, they are required to submit both their final written answer and a screencast capturing their screen and verbal explanations as they narrate how they arrived at a solution. Responses are then shared with peers on the platform, where students are encouraged to provide feedback by commenting on whether they agree or disagree with their peers' answers, and what they appreciated or found unclear in their peers' approaches. For this study, we focused specifically on this peer feedback dataset obtained from the authors of Zhang et al. [19], which consists of 229 comments contributed by 63 middle school students and forms the basis of our analysis.

2.2 Codebook Development

Drawing on the collected data, we iteratively developed the codebook through a thematic analysis approach [20]. One researcher first conducted an inductive review of the data to identify emerging themes (codes), then re-examined and refined code definitions and selected representative excerpts to ensure consistency between each code's intended meaning and application. This refinement process was guided by three grounding principles: (1) identified themes need to be broadly applicable across the dataset rather than reflecting rare instances; (2) coding categories that systematically vary in

terms of the *degree of inference* needed (resulting in two low- and two high-inference codes); (3) codes that are conceptually meaningful and would be practically useful if applied in future analyses to address legitimate research questions. At first glance, the third principle might seem irrelevant to this paper, as the primary aim is to examine construct inference levels to better understand human-model coding alignment, rather than to directly apply the codes to data analysis. However, grounding code selection in usefulness (principle 3) helps avoid constructs that are disconnected from the broader analytic goals of the research. Throughout the codebook development process, revisions were discussed with the research team to ensure that each code remained grounded in the data and aligned with our research focus. The final codebook is provided in the section below.

2.3 Degree of Inference Evaluation

After the codebook was finalized, two researchers independently evaluated the *degree of inference* required to apply each code. This evaluation aimed to examine whether variation in inferential demand corresponded to differences in human-model coding alignment. To classify each code as high- or low-inference, we applied the following three criteria, each generating a binary judgment (high vs. low inference):

1. *Are well-trained human coders unlikely to reach consensus on this code without extensive discussion?* If yes, the code is considered a high-inference code; otherwise, it is classified as a low-inference code.
 - Example: "Good morning, everyone!" would likely be coded as an instance of Greetings, as the meaning is explicitly stated and requires little interpretation. Coders are therefore likely to agree with minimal deliberation, indicating that this is a low-inference code.
2. *Does applying this code require the coder to infer internal states, intentions, or beliefs that are not explicitly stated in the text?* If yes, the code is considered a high-inference code; otherwise, it is classified as a low-inference code.
 - Example: "I've tried this so many times... I don't know what else to do?!!!" may be coded as Frustration. While frustration is not stated explicitly, coders may infer the speaker's underlying emotional state from the wording and context. This need for additional inference suggests this is a high-inference code.
3. *Does applying the code require coders' interpretive judgment due to multiple plausible interpretations of the line?* If yes, the code is considered a high-inference code; otherwise, it is classified as a low-inference code.
 - Example: "Oh sure, you're always the expert on everything." might be coded as an instance of Playful/Humor, depending on the coder's interpretation. Some may view it as playful teasing, while others may view it as sarcasm or subtle disrespect. This variation in coders' interpretive lenses makes it a high-inference code.

When all three criteria yielded consistent judgments, the classification was assigned accordingly. In cases of mixed judgments (e.g., low on one criterion but high on others), we applied a majority-vote rule, assigning the final classification based on the dominant

judgment across the three criteria. This approach is grounded in the premise that high-inference codes involve consistent interpretive demands across multiple dimensions, rather than isolated sources of ambiguity. For instance, if coders struggled to reach agreement (Criterion 1), but the code did not require inference of internal states or involve multiple plausible interpretations (Criteria 2 and 3), this was not considered sufficient evidence of high inference, as the disagreement may reflect ambiguity in the code definition, rather than inherent interpretive complexity. Similarly, codes involving multiple plausible interpretations (Criterion 3) were not automatically classified as high inference if coders still reached immediate consensus and little inferential reasoning was required (Criteria 1 and 2). In these cases, interpretive disagreements may indicate the need to carefully differentiate among conceptually related codes (i.e., multiple codes may plausibly apply to the same data segment). In other words, high-inference codes were characterized by consistent interpretive demands across multiple dimensions, rather than isolated sources of ambiguity alone.

It is important to note, however, that although these evaluation criteria focus primarily on the codes, assessing inference level also requires careful review of the data to which each code is applied. For example, even a low-inference code (e.g., Greetings) may become more difficult to apply when instances in the data are ambiguous or expressed in diverse or culturally specific ways. Conversely, a higher-inference code such as Confusion may require less inference when consistently expressed through fixed verbal indicators. Thus, a code's inferential demand may depend not only on the code itself, but also on how clearly and consistently it is manifested in the analyzed data.

Table 1. High- and low-inference codes, with inference levels identified through the evaluation described above. These codes are not mutually exclusive.

Code & Degree of Inference	Definition	Example
Disagreement (Low)	When students challenge or disagree with their peers' answers, processes, or methods.	<i>"I respectfully disagree with you your answer I think you might have messed up a step in the subtraction part."</i>
Compliments (Low)	When students offer positive comments, praise, or encouragement in their feedback to peers.	<i>"nice job explaining it in detail!"</i>
Reflection (High)	When students reflect on their problem-solving approach in relation to their peers', noting similarities, differences, or insights gained from the comparison.	<i>"I hadn't thought of using the direction arrows so I like how you added that."</i>
Reasoning (High)	When students provide an explanation or reason to justify their beliefs or opinions. The emphasis is on making the thinking behind a judgment explicit.	<i>"My strategy is like yours because I also made -12 the same way. I also think its great you put the 20-12 on the side and explained that wasnt the actual problem."</i>

To support a clear comparison in our analysis, we selected codes that were consistently characterized as either high- or low-inference across the three criteria described above, as well as in the context of the analyzed dataset. This excluded codes that showed inconsistent inference levels, which allowed us to establish a clearer and more focused contrast between high- and low-inference codes. Table 1 (see above) presents the final set of codes chosen for this analysis.

For low-inference codes (Disagreement, Compliments; see Table 1), the application tends to be more straightforward. For instance, the line *"I like it how you're organized and your voice is loud and clear. 10/10 Great job,"* exemplifies all three criteria characterizing Compliments as a low-inference code. Two human coders are likely to reach consensus with minimal discussion, as the praise is present in a highly direct form. Applying this code also requires little inference beyond the surface-level text, leaving little ambiguity about the speaker's intent or room for divergent coding decisions.

In contrast, applying high-inference codes (Reflection, Reasoning) typically demands more interpretative judgment. Consider the line: *"I think that Hermione [substituted name] had the best estimate because it's about how many people get the average amount of sleep not about how many hours is good for you, because that does not have anything to do with the problem".* At first glance, a coding decision is not obvious. Though the student contrasts two measures (population average vs. personal ideal) and explains why one is more relevant to the problem, the informal and subjective phrasing may obscure this reasoning. Applying this code thus requires inference beyond the literal text, as coders must determine whether the statement reflects structured reasoning or simply a personal opinion. This ambiguity increases the likelihood of different coding decisions, as coders may interpret the underlying cognition differently, increasing the inferential demands and the need for collaborative discussion to reach agreement.

2.4 Human Coding

Two human researchers independently coded the full dataset (~200 lines) using the developed codebook (see Table 1). A binary coding scheme was employed, in which '1' indicates the presence of a given code in the data, and '0' indicates its absence. After both researchers completed independent coding, Cohen's kappa was calculated to assess IRR. An acceptable agreement of 0.70 was reached for all codes (Disagreement: $\kappa=0.89$; Compliments: $\kappa=0.93$; Reflection: $\kappa=0.95$; Reasoning: $\kappa=0.77$). Researchers then engaged in a social moderation process [21], where they met to review coding discrepancies and discussed their interpretations. Through discussion, all coding disagreements were resolved. The resulting consensus-coded data served as the established human baseline for evaluating the model's coding performance.

2.5 Code Boundary Development

Boundary creation occurred as part of the social moderation process [21], where the two human coders discussed differences in how codes were interpreted and operationalized, articulated their rationales, and reconciled different understandings. As coders worked through coding disagreements, they examined what made a code applicable or

inapplicable in each instance and identified aspects most susceptible to divergent interpretation. The goal was not simply to "*correct coding errors*," but to surface ambiguities that could lead to inconsistent code application. This process also drew attention to edge cases—instances where code assignment was particularly challenging; discussing these cases helped surface the interpretive logic underlying code application and formalize it into explicit inclusion and exclusion criteria. For instance, the line "*I agree with you adding but for the positives but for the negatives your answer should have been negative*" was flagged by human coders as an edge case for Disagreement. While the student did not explicitly say "*I disagree*", the corrective response implicitly indicated that an alternative answer was more accurate. Initially, the coders were unsure whether Disagreement should only include direct opposition or also indirect corrections. After discussion, they reached a consensus to expand the inclusion criteria to encompass both explicit and implicit disagreement.

Boundaries differ from code definitions and examples. While definitions specify what a construct means and examples illustrate representative instances, boundaries emerge as coders operationalize the construct. Though a code definition may appear conceptually coherent, applying it to actual data can expose underspecified aspects that were not evident in the original definition. Boundaries are thus developed in response to these points of uncertainty or disagreement, serving as an operational refinement that clarifies how the code should be interpreted and applied in practice.

Building on the process of identifying, discussing, and resolving application-level ambiguity, inclusion and exclusion criteria (i.e., *boundaries*) were developed for each code (see Table 2), with refinements made based on coder dialogue and emerging edge cases. These finalized *boundaries* were then provided to GPT-4o as additional guidance during the second round of model-based coding. Although more advanced models (e.g., GPT-5.1, GPT-5.2) have since become available, GPT-4o was the model available at the onset of this study and was therefore used for all analyses to maintain consistency.

Table 2. Developed boundaries (i.e., inclusion and exclusion criteria) and reported base rates for both low- and high-inference codes.

Code	Boundaries
Disagreement (5%)	Inclusion Criteria: Includes both explicit disagreement (e.g., "<i>I disagree...</i>"), as well as implicit disagreement expressed through corrective language (e.g., "<i>That is how you find the median, not the mean...</i>"). Disagreement can reflect either an objective difference or a subjective difference between people. Exclusion Criteria: Statements where the student primarily offers positive feedback, clarification, or mixed evaluation rather than genuine disagreement.
Compliment (65%)	Inclusion Criteria: Students highlight what the peer did well (e.g., "<i>Good job</i>"), or express appreciation of their work or explanation ("<i>I like how you...</i>"). Exclusion Criteria: General comments that lack encouragement or positive affirmation, even if they describe the peer's work.

Reflection (24%)	Inclusion Criteria: When a student: (1) Uses first-person perspective language to describe how their strategy relates to or differs from a peer's strategy, (2) Reflects on their own approach after examining the peer's work, (3) Identifies something new they realized or learned from the peer's method. Exclusion Criteria: Feedback that only describes the peer's method without connecting it back to the student's own thinking, agreeing or disagreeing without reflecting on one's own work.
Reasoning (23%)	Inclusion Criteria: Explanation of why a person thinks in a particular way, often using language such as "<i>because</i>," "<i>since</i>," or "<i>the reason is</i>." This includes statements that justify, compare, or clarify the thinking behind an answer, process, or choice, with a focus on the student's underlying rationale. Exclusion Criteria: Student only agrees, disagrees, asks a question, or when the student evaluates correctness without explaining the logic.

2.6 Coding with GPT-4o

To evaluate the LLM's coding performance, we conducted two rounds of coding using GPT-4o. The first round focused on examining whether human-model agreement varied based on the *degree of inference* required by each code. We provided GPT-4o with the same dataset and codebook used in human coding and tested two prompting approaches: (1) zero-shot, where GPT-4o received only the code names and definitions; and (2) few-shot, in which GPT-4o was provided with code names, definitions, and illustrative examples. Given the model's stochastic nature, we set the temperature to 0 to minimize variability across runs. Further, for each code and prompting condition, we ran the same set of data and model three times under identical settings and applied a majority-vote rule when outputs differed. After GPT-4o completed all runs for each code, we calculated Cohen's kappa to assess human-model agreement. The prompt we used to instruct GPT-4o was adapted from Liu et al. [9].

The second round of coding aimed to address our second research question: For codes where an LLM struggles to align with human coders, does adding construct boundaries help improve human-LLM coding agreement? In this round, we used the same zero-shot and few-shot prompting structures as the first round and added the human-developed *boundaries* (i.e., inclusion and exclusion criteria) for each code. We again calculated Cohen's kappa for human-model coding agreement and compared these results with those from the first round, where boundaries were not incorporated.

3 Results

3.1 RQ1: Human-Model Alignment for Low- vs. High-Inference Codes

To answer our first research question (RQ1: how is the degree of inference required by a code associated with differences in human-model coding alignment?), we compared

the coding agreement between human coders and GPT-4o across high- and low-inference codes. As shown in Table 3 (see below), GPT-4o failed to align closely with human coding ($\kappa < 0.60$) for all codes under the zero-shot condition, except for one low-inference code (i.e., Compliments, $\kappa = 0.93$). When using the few-shot prompting approach, GPT-4o reached acceptable agreement on both low-inference codes (Disagreement, $\kappa = 0.80$; Compliments, $\kappa = 0.96$) but continued to struggle on the high-inference codes (Reflection and Reasoning, both $\kappa < 0.50$).

Table 3. Human-GPT coding agreement (Cohen's κ) for high- and low-inference codes, performance without adding boundaries.

Code	Degree of Inference	Zero-Shot κ	Few-Shot κ
Disagreement	Low	0.52	0.80
Compliment	Low	0.93	0.96
Reflection	High	0.56	0.34
Reasoning	High	0.54	0.45

These results suggest that the *degree of inference* required by a code plays a critical role in shaping how well GPT-4o aligns with human coders. In general, the model showed stronger agreement on low-inference codes, where coding decisions can be made more directly from surface-level textual cues, without needing more in-depth inference. However, some variation emerged across prompting conditions. Under zero-shot prompting, one low-inference code fell below a κ of 0.60 (Disagreement, $\kappa = 0.52$). This pattern reflects that even low-inference codes may require examples demonstrating how the code is applied in practice, as definitions alone may not fully capture how human coders interpret and apply the code. Conversely, under the few-shot condition—where GPT-4o was given illustrative examples alongside code definitions—the model reached higher agreement for both low-inference codes (Disagreement, $\kappa = 0.80$; Compliment, $\kappa = 0.96$). In these cases, examples appear to serve as effective scaffolds, helping the model understand how the codes are operationalized in practice and supporting more consistent mapping between relevant excerpts and code application.

For high-inference codes (Reflection and Reasoning), none of the prompting approaches achieved agreement above $\kappa = 0.60$. This result may be because, unlike low-inference codes, these coding decisions cannot be made from surface-level text alone. Applying high-inference codes requires the model to infer the speaker's underlying logic, intentions, or perspective—in other words, not just what the student said, but the meaning-making process underlying the utterance. For example, coding the line "*When you did the 3×2 and you got - 6 it could confuse people because the 3 and the 2 are both positives so how did you get a negative*" as Reasoning, depends on recognizing that the student is not only identifying an error, but also justifying why the answer doesn't make sense based on their understanding of multiplication rules. In such cases, successful coding relies on interpreting how meaning is constructed, a form of inferential judgment that may be more challenging for the model than for human coders.

3.2 RQ2: Effects of Scaffolding with Code Boundaries

To answer our second research question (RQ2: for codes where an LLM struggles to align with human coders, does adding construct boundaries help improve human-LLM coding agreement?), we provided GPT-4o with the same zero-shot and few-shot prompt structures as before, but additionally included the *boundaries* developed through the human coding process to examine whether supplying the model with inclusion and exclusion criteria could improve its coding performance compared to the original prompts.

Table 4. Human-GPT coding agreement (Cohen's κ) for high- and low-inference codes, including performance with and without boundaries, for comparison.

Code	Degree of Inference	Zero-Shot κ	Zero-Shot + Boundaries κ	Few-Shot κ	Few-Shot + Boundaries κ
Disagreement	Low	0.52	0.66	0.80	0.74
Compliment	Low	0.93	0.96	0.96	0.97
Reflection	High	0.56	0.75	0.34	0.78
Reasoning	High	0.54	0.67	0.45	0.81

The results show that adding boundaries improved human-model agreement across all codes under the zero-shot condition, regardless of their inference levels (See Table 4 above). Notably, the addition of boundaries substantially improved agreement for one high-inference code (Reflection), with κ increasing from 0.56 to 0.75. When boundaries were added under the few-shot condition, improvements were clear for the two high-inference codes. Human-model agreement increased substantially for both categories: Reflection: κ increased from 0.34 to 0.78; Reasoning: κ increased from 0.45 to 0.81.

The addition of boundaries may have improved performance by helping the model distinguish between similar yet meaningfully distinct codes, particularly for high-inference codes. For example, GPT-4o initially coded *"I like the way you spread out each problem, I also like the ending when u said 'boom bam bye bye'"* as Reflection, differing from the human coding. This divergence likely reflected the model's over-interpretation of praise as evidence of reflection. After boundaries were added to specify that Reflection required students to relate the peer's response to their own approach or articulate new insights gained from the interaction, the model changed its decision. This shift shows how boundaries can serve as useful interpretative scaffolds, helping the model navigate ambiguity and align more closely with human judgment.

For the two low-inference codes, adding *boundaries* did not consistently enhance human-model alignment, however. For Compliments, where GPT-4o had already achieved very high agreement under the few-shot condition ($\kappa=0.96$), adding boundaries offered limited improvements (κ increased to 0.97). For Disagreement, introducing *boundaries* slightly reduced agreement with human coders (κ dropped from 0.80 to 0.74). In this case, adding *boundaries* may restrict the model's interpretative flexibility, potentially over-filtering otherwise valid instances. For instance, GPT-4o initially coded the line, *"I like the way you showed your work step by step, but the question asks*

what the total amount of money he lost or gained by the end of the day", as an implicit form of disagreement, consistent with human coders. As GPT-4o explained, "*The text exemplifies Disagreement because the student is pointing out that the other student's response does not fully address the question, even though they acknowledge the effort in showing the work step by step. This indicates a disagreement with the completeness of the answer.*" Yet, after boundaries were added, GPT-4o reversed its decision and no longer coded it as Disagreement. This change was also accompanied by a noticeable shift in the model's rationale: "*The text primarily offers positive feedback ('I like the way you showed your work step by step') and provides clarification about the question ('but the question asks what was the total amount of money he lost or gained by the end of the day'). It does not explicitly or implicitly express disagreement with another student's answer, process, or method.*" This shift highlights that added boundaries, while intended to guide, can sometimes also constrain the model's interpretation.

These results suggest that providing models with *boundaries*, in addition to prompts, can be an effective strategy for improving the model's ability to make more informed judgments about when to apply a code. This strategy may be particularly helpful for codes that the model initially struggled with—especially those requiring a high *degree of inference* for interpretation and application. By offering explicit inclusion and exclusion criteria, *boundaries* help reduce interpretative ambiguity and may better scaffold the model to approximate human judgment. However, adding *boundaries* is not always beneficial. For low-inference codes where the model already performs well with prompts alone, *boundaries* may introduce redundant information or unnecessary noise that hinders performance. These cases reflect a potential downside of excessive guidance: highly specific instructions may lead the model to rely too heavily on decision rules, examples, or dataset-specific patterns rather than a code's broader conceptual meaning. This pattern is reminiscent of overfitting [22], whereby more examples intended to improve performance may reduce the model's ability to generalize to novel instances. Hence, while boundaries can support human-model coding alignment, they should be used judiciously, with careful consideration of each specific code's demands and the model's baseline performance.

4 Discussion and Limitations

This study aims to address two key research questions. For RQ1: How is the degree of inference required by a code associated with differences in human-model coding alignment? Our findings show that GPT-4o achieved higher agreement overall with human coders on low-inference codes, especially for few-shot prompting ($\kappa \geq 0.8$), in which both code definitions and illustrative examples were provided. In contrast, alignment was lower for high-inference codes ($\kappa \leq 0.6$), regardless of prompt approach (zero-shot or few-shot). This suggests that, in its current capacity, LLMs like GPT-4o may be better suited for coding tasks involving constructs with lower inferential demand.

For RQ2: For codes where an LLM struggles to align with human coders, does adding construct boundaries help improve human-LLM coding agreement? We provided GPT-4o with human-developed boundaries in addition to the original code definitions

and examples. We observed that for codes GPT-4o initially struggled with—particularly high-inference codes—adding boundaries substantially improved human-model agreement (e.g., κ increased from 0.45 to 0.81 for Reasoning). This finding indicates that boundaries can scaffold model decisions by reducing ambiguity and guiding the model toward human sense-making. However, for low-inference codes, adding boundaries sometimes slightly reduced agreement, possibly because the additional constraints introduced unnecessary complexity into the model's decisions and restricted the interpretation of otherwise straightforward constructs. Such over-specification may narrow the model's decision space and increase reliance on prompt-specific distinctions, making coding more sensitive to superficial features and less robust across different expressions of the same construct. Researchers should thus avoid treating boundaries as a universally useful strategy, and instead carefully consider each model's baseline performance before deciding whether additional scaffolding is beneficial or necessary.

One possible explanation for why *boundaries* help improve human-model alignment for high-inference codes is that LLM-based qualitative coding typically relies on human-developed codebooks to guide the model's coding decisions. Yet, models may still diverge from human interpretations, particularly given that their reasoning processes differ from those of humans [16]. Incorporating boundaries may offer the model a more structured basis for code assignment, thereby facilitating closer alignment with human judgment. Beyond improving human-model alignment, defining a code's boundaries may also improve consistency among human coders by surfacing implicit interpretive differences. This is important because even well-written code definitions can leave interpretative gray areas, especially for high-inference codes, leading coders to apply the same code differently despite using a shared codebook. By discussing which aspects of a construct are most vulnerable to divergent interpretation, researchers can make differences in reasoning explicit and negotiate a code's conceptual limits—that is, what constitutes an appropriate instance and what falls outside its scope. These boundaries can then scaffold coders to navigate gray areas with greater clarity and consistency.

In considering these findings, several limitations should be noted. First, the analysis focused on a small set of constructs (four), a relatively small dataset (just over 200 lines of student data), and a single LLM (GPT-4o). Future work should test whether these patterns hold across larger datasets, broader codebooks, and additional LLMs to better assess the robustness and generalizability of our findings. Our study also treats human-coded data as the standard for evaluating GPT-4o's performance and positions coding agreement as the primary goal. We acknowledge that in some cases, disagreement (between humans or between humans and models) is also analytically valuable (e.g., [23]). Examining such divergence can prompt researchers to revisit assumptions, surface alternative interpretations, and, in the case of human-model disagreement, obtain rationales that may further expand human understanding. Additionally, while this study introduced criteria for evaluating codes' inference levels, future work should further refine how they are assessed. As inference demands may play a key role in shaping codes' suitability for LLM-based auto-coding, clearer and more concrete guidelines for evaluating inference levels are an important next step. This may also open opportunities to move beyond a binary high-low distinction by considering intermediate levels of inference. Finally, boundaries in this study were developed through social moderation of

human coder disagreements and applied in pairs. However, our ongoing work suggests that human–model divergence may also offer useful insights for identifying and refining code boundaries, and that boundaries may also be useful when applied independently. Future research should thus explore how different forms of disagreement and configurations of boundary criteria, can be systematically integrated into qualitative coding workflows to support more reliable code application.

5 Conclusion

This study contributes to ongoing discussions on how LLMs can be more effectively integrated into qualitative coding. We explored factors that may explain why LLMs align better with humans for some constructs than others, identifying the *degree of inference* as a potential factor shaping human-model coding agreement. We further propose the use of *boundaries*—codes' inclusion and exclusion criteria—as a strategy to improve LLM performance, particularly for high-inference codes that models initially struggle with. Our findings suggest that the codes' inference level is associated with human-model coding alignment, and that incorporating *boundaries* can improve LLM performance on such constructs. Together, these results offer practical insights for more thoughtfully integrating LLMs into qualitative coding, especially for large-scale textual analysis. More broadly, this study highlights the importance of accounting for the interpretive demands of qualitative constructs when considering approaches to automated coding by LLMs. It also points to the need to develop targeted strategies (such as the use of *boundaries*) to scaffold models when coding constructs with higher inference demands, where achieving alignment with human coders may be more challenging. By taking these steps, researchers may be better positioned to use LLMs in qualitative coding, while preserving the interpretive reasoning central to qualitative inquiry.

Acknowledgments. This study was supported by NSF # DRL-2300829.

Disclosure of Interests. The authors declare no competing interests relevant to this work.

References

1. Miles, M. B., Huberman, A. M., Saldaña, J.: Qualitative data analysis: A methods source-book. 3rd ed. Sage, USA (2014)
2. Elliott, V.: Thinking about the coding process in qualitative data analysis. *Qualitative Report* **23**(11), 2850-2861 (2018)
3. Arastoopour Irgens, G., & Eagan, B.: The foundations and fundamentals of quantitative ethnography. In: *Int. Conf. on Quantitative Ethnography*, pp. 3-16. Springer, Denmark (2022)
4. Shaffer, D. W., & Ruis, A. R.: How we code. In: Ruis, A.R., Lee, S.B. (eds). *Int. Conf. on Quantitative Ethnography*, pp. 62-77. Springer, Online (2021)
5. Shaffer, D. W., Collier, W., & Ruis, A. R.: A tutorial on epistemic network analysis: Analyzing the structure of connections in cognitive, social, and interaction data. *Journal of learning analytics*, **3**(3), 9-45 (2016)

6. Liang, Y., Wei, Z., Liu, X., Zhu, X., Gao, Y., Chen, B.: Exploring role-based knowledge co-construction in social annotation with epistemic network analysis. In: *Int. Conf. on Quantitative Ethnography*, pp. 226-241. Springer, Mexico (2025)
7. Donegan, S. R., Porter, C., Fogel, A., Seo, J., Choi, S., & Eagan, B.: US media coverage during COVID-19: An epistemic network analysis of bias, topic, and trajectory. In: *Int. Conf. on Quantitative Ethnography*, pp. 23-26. Springer, Online (2021)
8. Zörgő, S., Peters, G. Y., Csajbók-Veres, K., Geröly, A., Jeney, A., & Ruis, A. R.: An epistemic network analysis of patient decision-making regarding choice of therapy. *Quality & Quantity*, 57, 3105–3132 (2023)
9. Liu, X., Zambrano, A. F., Baker, R. S., Barany, A., ..., Wei, Z.: Qualitative coding with GPT-4: Where it works better. *Journal of Learning Analytics* 12(1), 169-185 (2025)
10. Liu, X., Zhang, J., Barany, A., Pankiewicz, M., Baker, R.S.: Assessing the potential and limits of large language models in qualitative coding. In: *Int. Conf. on Quantitative Ethnography*, pp. 89-103. Springer, USA (2024)
11. Morgan, D. L.: Exploring the use of artificial intelligence for qualitative data analysis: The case of ChatGPT. *International Journal of Qualitative Methods* 22, 1-10 (2023)
12. Dunivin, Z. O.: Scaling hermeneutics: A guide to qualitative coding with LLMs for reflexive content analysis. *EPJ Data Science* 14(1), 28 (2025)
13. Zambrano, A. F., Wei, Z., Zhang, J., Baker, R. S., Ocumpaugh, J., Barany, A., ..., Borchers, C.: Data plus theory equals codebook: Leveraging LLMs for human-AI codebook development. *Journal of Educational Data Mining* 18(1), 25-65 (2026)
14. Barany, A., Nasiar, N., Porter, C., Zambrano, A. F., Andres, A. L., Bright, D., ... & Baker, R. S.: ChatGPT for education research: Exploring the potential of large language models for qualitative codebook development. In: *Int. Conf. on artificial intelligence in education*, pp. 134-149. Springer, Brazil (2024)
15. Borse, N. S., Subramaniam, R. C., Rebello, N. S.: Investigation of the inter-rater reliability between large language models and human raters in qualitative analysis. *arXiv preprint arXiv:2508.14764* (2025)
16. Amirizani, M., Martin, E., Sivachenko, M., Mashhadi, A., Shah, C.: Can llms reason like humans? Assessing theory of mind reasoning in llms for open-ended questions. In: *Int. Conf. on Information and Knowledge Management*, pp. 34-44. ACM, USA (2024)
17. Loru, E., Nudo, J., Di Marco, N., Santirocchi, A., Atzeni, R., Cinelli, M., ... & Quattrocioni, W.: The simulation of judgment in LLMs. *Proceedings of the National Academy of Sciences*, 122(42), e2518443122 (2025)
18. Fonteyn, M. E., Vettese, M., Lancaster, D. R., & Bauer-Wu, S.: Developing a codebook to guide content analysis of expressive writing transcripts. *Applied Nursing Research*, 21(3), 165-168 (2008)
19. Zhang, J., Baker, R. S., Andres, J. M., Hutt, S., Sethuraman, S.: Automated multi-dimensional analysis of peer feedback in middle school mathematics. In: *Int. Conf. on Computer-Supported Collaborative Learning - CSCL*, pp. 221-224. CSCL Proceedings, Canada (2023)
20. Braun, V., & Clarke, V.: Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2), 77-101 (2006)
21. Herrenkohl, L. R., Cornelius, L.: Investigating elementary students' scientific and historical argumentation. *Journal of the Learning Sciences* 22(3), 413-461 (2013)
22. Tang, Y., Tuncel, D., Koerner, C., Runkler, T.: The few-shot dilemma: Over-prompting large language models. *arXiv preprint arXiv:2509.13196* (2025)
23. Garcia, N.M., Onyewuenyi, A.C., Shaffer, D.W.: Critical quantitative ethnography (CritQE): A pathway toward convergence research and equity. In: *Int. Conf. on Quantitative Ethnography*, pp. 52-66. Springer, Mexico (2025)