Agent-as-Peer-Debriefer: A Multi-Agent Framework with Perspective-Based Refinement for Qualitative Analysis

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly used for qualitative data analysis (QDA), yet their outputs often miss the depth and nuance of human analysis. We argue this gap reflects a missing credibility practice from human QDA: *peer debriefing*, in which an analyst seeks feedback from a *disinterested peer* and uses it to refine their coding. To bring this practice into LLM-assisted QDA, we propose `Agent-as-Peer-Debriefer`, a multi-agent QDA framework that builds *peer debriefing* into key coding steps. In our framework, a *Hierarchical Coding Agent* follows the standard QDA process to generate codes, sub-themes, and themes, along with self-explanations and reflection memos. It then shares these outputs with three *Peer-Debriefing Agents*, each applying a distinct analytical perspective (Theory-Driven, Data-Driven, or Applied) and refining the codes by keeping, renaming, reassigning, merging, or splitting them. These perspectives are drawn from established human QDA practices that generalize across domains and datasets. To evaluate the framework, we test it on three datasets across two domains with three LLMs, measuring semantic similarity to human-annotated codes. Across all settings, perspective-based, peer-debriefing refinement aligns more closely with human codes than a single-LLM baseline, and an ablation further shows the gain is not merely from additional refinement. The three perspectives also produce distinct trade-offs, showing that the choice of perspective is a meaningful and controllable design decision. More broadly, these findings suggest that simulating *peer debriefing* with explicit perspectives is a promising route to more credible LLM-assisted QDA.

1 Introduction

Large language models (LLMs) are increasingly used to assist qualitative data analysis (QDA), from generating initial codes to synthesizing themes

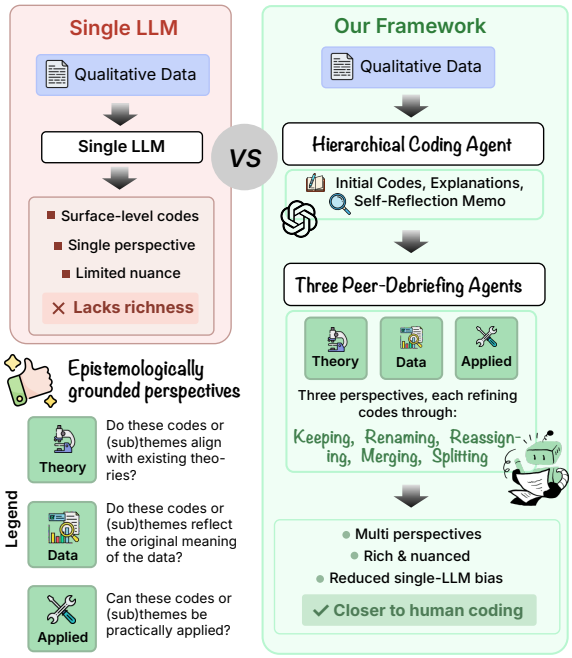

Figure 1: Motivation: A single LLM tends to produce surface-level codes. Our framework simulates peer debriefing: a *Hierarchical Coding Agent* shares its initial codes, explanations, and self-reflection memos with *Peer-Debriefing Agents*, to seek refinement feedback from them. The resulting codes are richer, more nuanced, and closer to human coding.

(Dai et al., 2023; Parfenova et al., 2024; Zhong et al., 2025). Yet LLM-generated codes often lack the richness and nuance that human coding teams produce (Retkowski et al., 2025; Parfenova et al., 2025). A key reason is that human teams do not rely on a single analyst's interpretation (Richards and Hemphill, 2018; Gao et al., 2023, 2024). Instead, they use procedures like *peer debriefing* to incorporate multiple perspectives and challenge initial readings of the data (Lincoln, 1985; Spall, 1998; Barber and Walczak, 2009).

Peer debriefing is a standard practice in qualitative research: a researcher consults several colleagues with different backgrounds, often indepen-

dently, to get feedback that helps refine their interpretation. These *disinterested peers* are expected to understand the research context but have no personal stake in the outcome (Hail et al., 2011). This process enriches the initial interpretation with nuances a single analyst would miss, and counters the blind spots that come from working alone.

In this work, we ask, **what if we bring peer debriefing into LLM-assisted QDA?** Can it bring codes from a single LLM closer to those produced by human teams? We propose `Agent-as-Peer-Debriefer`, a multi-agent framework that simulates this: After generating initial coding (i.e., initial codes, subthemes, and themes), the *Hierarchical Coding Agent* shares its output, explanation and self-reflection memos to three independent *Peer-Debriefing Agent*s, who refine initial coding from different perspectives. Based on this, we investigate two research questions (RQs):

- **RQ1:** Does *peer debriefing* improve qualitative coding quality?
- **RQ2:** How do different analytical perspectives affect coding outcomes?

We compare codes refined under each perspective against a single-LLM baseline (no refinement), and an ablation study of self-refinement without perspective. For **RQ1**, we find that across all models and datasets, the best-performing perspective raises the match rate with human codes above the no-refinement baseline (e.g., GPT-5 on ScrumRQ1: 23.6%→46.9%), and exceeds Self-Refinement on the same setting (32.3%). For **RQ2**, we find that each perspective produces a distinct trade-off: Data-Driven maximizes recall of human codes (GPT-5 on ScrumRQ1: 70.6%→88.2%), Theory-Driven and Applied maximize match rate (up to 46.9%), and the no-refinement baseline keeps the most code diversity (25.9% vs. 11–18% under perspectives). We summarize our contributions below:

- We propose `Agent-as-Peer-Debriefer`, a multi-agent framework that brings *peer debriefing* into LLM-assisted QDA: a *Hierarchical Coding Agent* shares its codes, explanations, and memos with three *Peer-Debriefing Agent*s, each refining the initial coding from a distinct analytical perspective.
- We show that *peer debriefing* consistently improves coding quality across datasets, domains, and LLMs, and that each perspective produces a distinct trade-off, making perspective choice a meaningful and controllable design decision in LLM-assisted QDA.

2 Related Work

2.1 LLM-assisted QDA

Computational tools have long supported qualitative coding and corpus reading (Nelson, 2020; Worden, 2017; Evans and Foster, 2024). LLMs are the most recent of these tools, and NLP and HCI work has applied them to coding (Than et al., 2025; Schroeder et al., 2025). Examples include interactive coding interfaces (Rietz and Maedche, 2021; Gebreegziabher et al., 2023), deductive coding against a fixed codebook (Xiao et al., 2023; Tai et al., 2024), inductive code and theme generation (Gao et al., 2024; Than et al., 2025), and broader studies of LLMs in qualitative work (Gao et al., 2023; Bano et al., 2024; Hamilton et al., 2023). A recurring finding is that LLM-generated codes often differ from those of human coders. For example, Xiao et al. (2023) found only fair to substantial agreement between GPT-3 and expert coders even when the model was given the codebook, and Tai et al. (2024) reported similar gaps when applying GPT-3.5 to deductive coding. Other work uses LLMs to find codes that analysts might miss (Barany et al., 2024). These results point to the need for review by multiple coders, as human teams already do.

2.2 Peer Debriefing in QDA

In qualitative research, a researcher's background shapes analytical outcomes. Braun and Clarke (2019) argue that in reflexive thematic analysis, researcher subjectivity is not a threat but an analytical resource, as theoretical assumptions and analytic orientations actively shape coding and theme development. Reflexivity, the ongoing practice of treating one's background as an active part of the analysis rather than a bias to remove, is itself a methodological commitment in qualitative research (Olmos-Vega et al., 2023). Empirical evidence supports this: Ortloff et al. (2023) found that researchers from different institutions produced substantively different codebooks from the same data, suggesting that disciplinary background, not just skill level, drives analytical variation.

In practice, this variation is managed through *peer debriefing* (Lincoln, 1985; Hail et al., 2011), a process in which colleagues from different disci-

plinary backgrounds, typically disinterested peers who are familiar with the research context but not invested in the analytical outcomes, review and challenge the coding from their respective standpoints. Recent HCI work points in the same direction: Reflexis (Ye et al., 2026) embeds reflexive prompts into collaborative coding so that disagreements between analysts surface as productive dialogues rather than errors to resolve. Our framework operationalizes this practice by introducing *Peer-Debriefing Agent* that refines coding results from distinct analytical stances based on the self-reflection memos generated by the initial coding LLM, mirroring peer debriefers with different disciplinary orientations.

2.3 Multi-Agent Systems

Recent advances in LLM-based multi-agent systems have demonstrated the value of multi-agent systems for complex tasks. Frameworks such as AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024) enable structured multi-agent collaboration, while ChatEval (Chan et al., 2024) use multi-agent argumentation for evaluation. For qualitative analysis, Thematic-LM (Qiao et al., 2025) leverages multi-agent systems for theme generation. Advancing this line of research, our framework draws on qualitative research methodology, simulating the *peer debriefing* process in its multi-agent design. Moreover, the perspectives assigned to our *Peer-Debriefing Agent* are not domain-specific, making the framework broadly applicable across diverse datasets.

3 Multi-Agent Framework

We propose a multi-agent framework for LLM-assisted QDA that incorporates *peer debriefing* at every stage of the coding process. We first describe the overall framework (Section 3.1), then the two agent types (Section 3.2) and the analytical perspective modeling (Section 3.3).

3.1 Framework Overview

Following the practice of *peer debriefing* in qualitative research, where someone outside the original analysis reviews the coding, we add *Peer-Debriefing Agent* to the standard LLM-assisted QDA setup. In this pipeline, a *Hierarchical Coding Agent* generates initial codes, sub-themes, and themes, along with explanations and self-reflection memos. At each stage, these results are sent to three distinct *Peer-Debriefing Agent*s, who review and refine the coding results based on these explanations and memos. It adopts three distinct analytical perspectives from established QDA practice: Theory-Driven, Data-Driven, or Applied (Section 3.3). The framework runs all three in parallel at each stage, producing three refined coding structures. The human analysts may remain in the loop to select the most favorable one. Because these perspectives are domain-general, the framework transfers to new qualitative analysis tasks without rewriting the prompts; the only inputs that change across tasks are the research question and the interview data.

3.2 Agent Design

The framework has two types of agents: a *Hierarchical Coding Agent* that produces the initial coding structure, and three *Peer-Debriefing Agent* that refine it from different analytical perspectives.

Hierarchical Coding Agent The *Hierarchical Coding Agent* processes the qualitative data in three sequential stages. Aligning with the key steps of thematic analysis (Braun and Clarke, 2006; Dai et al., 2023; Gao et al., 2025), in the *code generation* stage, it segments the interview text into analytical units and assigns each unit an initial code. In the *sub-theme abstraction* stage, it groups related codes into sub-themes. In the *theme synthesis* stage, it aggregates sub-themes into overarching themes. The same agent architecture runs at every stage with stage-specific instructions; only the input size and level of abstraction change. At each stage, the agent also writes explanations for codes it produces, together with self-reflection memo for the stage. These give the *Peer-Debriefing Agent* the evidence and reasoning behind the coding decision. At each stage, the agent writes the self-reflection memo in a single pass, without further revising it; we leave iterative self-reflection to future work.

Peer-Debriefing Agent Each *Peer-Debriefing Agent* takes on a specific analytical perspective. Given an existing coding structure and the *Hierarchical Coding Agent*'s reflection memo, it re-examines the relationships among codes, sub-themes, and themes from that perspective and may apply five operations:

- **Keep**: Retains the code without modification.
- **Rename**: Changes only the code name without altering its content or hierarchy.
- **Reassign**: Moves a code to a different parent within the current level.

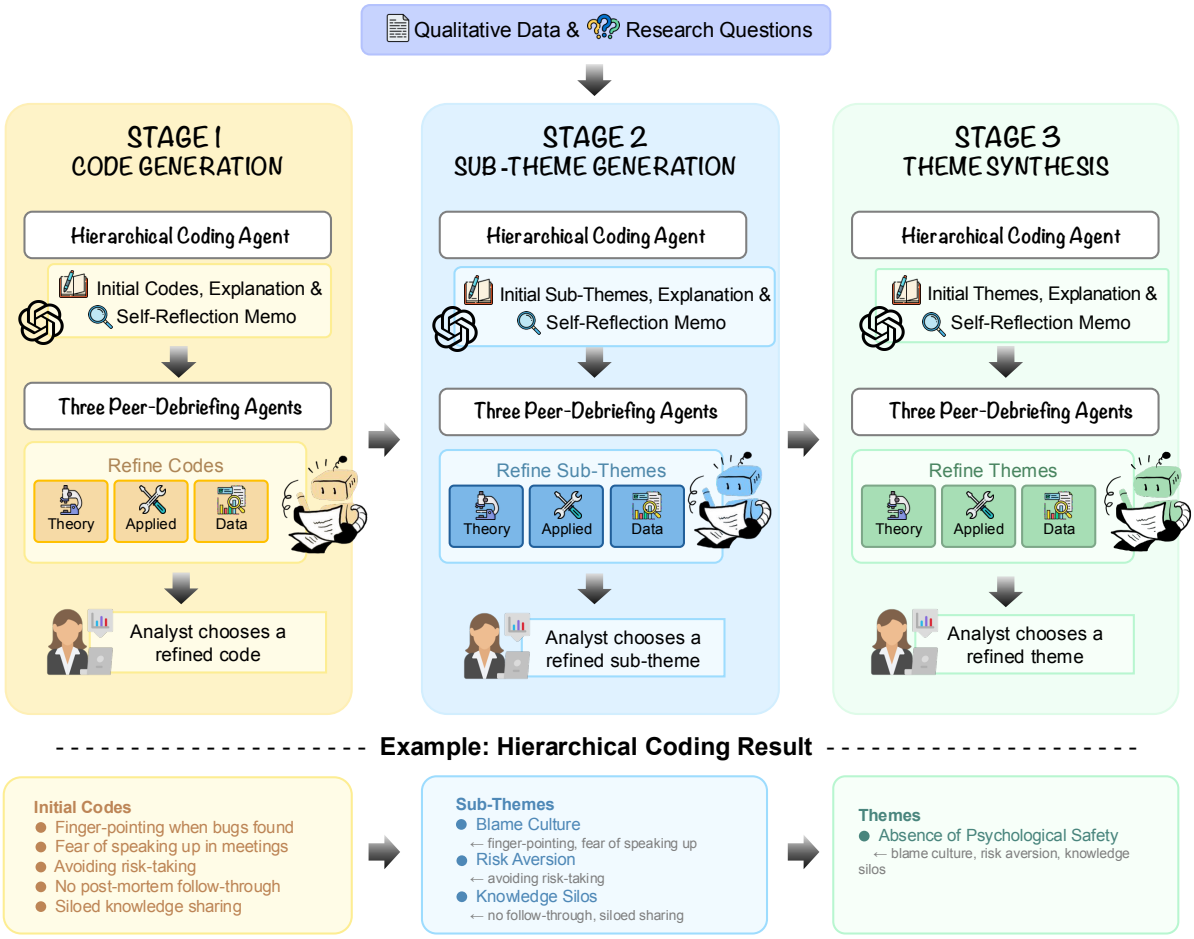

Figure 2: The framework consists of three hierarchical stages: code generation, sub-theme generation, and theme synthesis. At each stage, the *Hierarchical Coding Agent* produces structured results with explanations and self-critiques. *Peer-Debriefing Agent* then re-examine and refine these results from different analytical perspectives. Researchers may stay in the loop to select the preferred refinement at each stage.

- **Merge**: Combines multiple codes into a single code.
- **Split**: Divides one code into multiple codes.

These operations restructure the coding hierarchy without rewriting the original codes or the interview text. Instead of generating a new coding structure from scratch, the agent acts as a *disinterested peer* who reinterprets the same structure from a specific perspective, so that differences between perspectives become directly comparable. We also do not include any human-generated codes as few-shot examples in the agent prompts; the prompts contain only general analytical instructions from established qualitative research methods, so the agents do not pick up any individual researcher's coding style. The full prompt templates are listed in Appendix D.

3.3 Perspective Modeling

To keep the framework domain-general, we ground each perspective in a widely used qualitative research methodology (Saldaña, 2021; Charmaz, 2015) rather than tailoring it to a specific dataset or domain. We define three representative perspectives below. Each one applies a different *evaluation criterion* when choosing candidate codes, and we illustrate them using the same interview excerpt:

Theory-Driven Perspective The theory-driven perspective (Braun and Clarke, 2006) is based on *external knowledge*: theoretical frameworks and concepts from the relevant discipline. When two candidate codes conflict, the agent keeps the one with stronger theoretical grounding and better alignment with the literature. This mirrors researchers who read qualitative data through a theoretical lens. For example, given the interview statement *"In our team, whenever a bug is found, people start*

pointing fingers at each other," the agent may produce a code such as `Absence of Psychological Safety`, which links the observation to the concept of psychological safety (Edmondson, 1999).

Data-Driven Perspective The data-driven perspective (Braun and Clarke, 2006) is based on *local context*: the language and meaning in the data itself, without imposing any external framework. Codes emerge inductively from what participants say, and when two candidates conflict, the agent keeps the one that stays closer to the participant's wording. A related concept is "in-vivo coding", where researchers use participants' original words or phrases as codes (Saldaña, 2021). This mirrors researchers who do inductive qualitative analysis and stay close to the data. For the same interview excerpt, the agent may produce a code such as `Finger-Pointing When Bugs Are Found`, which preserves the phrasing without abstracting it into a theoretical label.

Applied Perspective The applied perspective emphasizes *practical utility*: whether a code can directly inform a decision or an action. When candidates conflict, the agent keeps the one that more clearly suggests what a practitioner should do, even if another code is more theoretical or stays closer to the participant's wording. This mirrors practitioners who use qualitative analysis to inform decisions. The perspective is similar to "Process code" (Saldaña, 2021), which prioritizes actions and activities in the data. For the same excerpt, the agent may produce a code such as `Need for Blame-Free Post-Mortem Process`, which points to a concrete change a team lead could act on.

4 Experiments

4.1 Experimental Settings

Public Qualitative Datasets We evaluate the framework on two public qualitative interview datasets from different domains. The **Scrum** dataset[1] (Alami and Krancher, 2022) contains 39 semi-structured interviews with software professionals (Developers, Product Owners, Scrum Masters, and QA practitioners) on how Scrum contributes to software quality. We evaluate two research questions from the original study separately: ScrumRQ1 and ScrumRQ2. The dataset provides per-interview codes (each data chunk annotated with one or more codes) and a codebook with code definitions and frequency counts. The **UCSB** dataset[2] (Curty et al., 2021) contains 10 semi-structured interviews with university instructors on quantitative data teaching in the social sciences. It provides a codebook of codes organized into sub-themes and themes, which allows evaluation at the thematic level. Table 1 summarizes the two datasets. Together, they let us evaluate coding quality at the per-interview level (Scrum) and thematic coherence at the sub-theme and theme level (UCSB), while codebook-level metrics provide cross-dataset comparability.

To our knowledge, this is one of the most comprehensive evaluation setups in the LLM-assisted qualitative analysis literature to date. Most existing studies evaluate on a single dataset (Spinoso-Di Piano et al., 2023; Retkowski et al., 2025) at the code level only, often using short text segments such as survey responses or quote-code pairs rather than full interview transcripts (Parfenova et al., 2025; Parfenova and Pfeffer, 2025). In contrast, our evaluation spans three datasets from two domains and covers all three levels of thematic analysis (code, sub-theme, and theme) across three LLM backbones. On Scrum, we also generate codes per interview and compare them against the ground truth. All three datasets include published human-annotated ground-truth codes, which allows direct quantitative comparison at each analytical level and makes our results reproducible.

Base Models We run the framework with three LLMs as the agent backbone: GPT-5 (Singh et al., 2025), DeepSeek-V3.2 (Liu et al., 2025), and Qwen3.5-Plus (Team, 2026). This lets us compare how different base models affect coding quality under the same framework.

Perspective Isolation We evaluate each perspective independently to isolate its effect on coding quality. As baselines, we compare against single-pass coding with no refinement and a refinement pass without any perspective. In practice, researchers can mix perspectives across stages (e.g., choosing a theory-driven result at the code level and an applied result at the sub-theme level). We leave mixed-perspective strategies to future work.

4.2 Metrics Definition

We evaluate the quality of generated codes with three metrics drawn from prior work (Gao et al.,

[1] https://zenodo.org/records/6624064

[2] https://datadryad.org/dataset/doi:10.25349/D9402J

	ScrumRQ1	ScrumRQ2	UCSB
Domain	Software Engineering		Education
Interviews	39		10
Per-interview codes	✓	✓	–
Overall Codebook	✓	✓	✓
Overall Sub-themes	–	–	✓
Overall Themes	–	–	✓

Table 1: Ground-truth annotation granularity of the evaluation datasets.

2023; Gebreegziabher et al., 2023): Recall, Match Rate, and Code Diversity. Throughout this section, let $G = \{g_1, g_2, \dots\}$ denote the set of codes the model generates and $H = \{h_1, h_2, \dots\}$ the set of human-annotated codes (the ground truth). Recall and Match Rate compare G against H through a code-level matching procedure described next; Code Diversity is computed on G alone.

Code-level Matching Intuitively, for each generated code we look up its closest human code by meaning, and treat the two as the same code if they are similar enough. Formally, for each generated code $g_i \in G$, we compute its cosine similarity with every human-annotated code $h_j \in H$, using all-MiniLM-L6-v2 (Wang et al., 2020) as the text encoder. We normalize each score to $[0, 1]$ via $\tilde{s}_{i,j} = (s_{i,j} + 1)/2$ and take the best match $j^* = \arg\max_j \tilde{s}_{i,j}$. We then define a matching function $M : G \to H \cup \{\varnothing\}$: if $\tilde{s}_{i,j^*} \geq \tau$ (we set $\tau = 0.7$), $M(g_i) = h_{j^*}$; otherwise $M(g_i) = \varnothing$, meaning g_i has no match. The matching is one-to-many: each generated code matches at most one human code, but a single human code can be matched by several generated codes. We chose $\tau = 0.7$ to be permissive enough for paraphrase but strict enough to exclude pairs that are topically related but conceptually distinct, and verified it by manually inspecting boundary pairs (similarity in 0.65–0.75).

Recall Intuitively, Recall asks: of all the codes a human produced, how many did the model also recover? Formally, Recall is the fraction of human-annotated codes matched by at least one generated code:

$$\text{Recall} = \frac{|\{j : \exists\, i,\ M(g_i) = h_j\}|}{|H|},$$

where g_i is a generated code, h_j a human-annotated code, and $M(\cdot)$ the matching function defined above (so $M(g_i) = h_j$ means g_i was matched to h_j).

Match Rate Intuitively, Match Rate asks: of all the codes the model produced, how many line up with something a human also coded? Akin to precision, it is the fraction of generated codes that hit some human code:

$$\text{Match Rate} = \frac{|\{i : M(g_i) \neq \varnothing\}|}{|G|},$$

where $M(g_i) \neq \varnothing$ means generated code g_i found a human-annotated code with similarity above τ.

We use the term Match Rate rather than precision because all generated codes are grounded in the interview text and thus analytically valid; a generated code that does not match any human code is not necessarily a false positive but may reflect a valid interpretation not captured by the human codebook. Match Rate therefore measures the degree of alignment with the human reference rather than correctness per se.

Code Diversity Intuitively, Code Diversity asks: of all the codes the model produced, how many are semantically distinct from the others? We iteratively remove one code from each pair of generated codes whose similarity $\tilde{s}(g_i, g_k) \geq \tau$, yielding a deduplicated set G^*:

$$\text{Code Diversity} = \frac{|G^*|}{|G|},$$

where G is the full set of generated codes and G^* is its deduplicated subset (with near-duplicate pairs collapsed to one). Higher values indicate greater independence among generated codes.

4.3 Results

We report our main results in two parts, corresponding to our two research questions (RQ1 and RQ2). For each RQ, we compare the three perspectives (Theory-Driven, Data-Driven, and Applied) against a no-refinement baseline across all models and datasets. We additionally run a Self-Refinement ablation on GPT-5 (same prompt structure as the *Peer-Debriefing Agent* but without any perspective framing; see §4.4) to test whether gains stem from perspective or from a second refinement pass. We analyze the results using the three evaluation metrics defined above. All reported means are averaged over 39 interviews for the Scrum RQ1 and Scrum RQ2 datasets, and 10 interviews for UCSB.

4.3.1 RQ1: Does peer-debriefing improve coding quality?

Our results show that for each model, dataset combination, the best-performing perspective substan-

tially improves Match Rate over the baseline (Figure 3). We show the best-performing perspective for each configuration as an illustration, results are reported in Table 3. For example, on the ScrumRQ1 dataset, the Match Rate of GPT-5 increases from 23.6% to 46.9% (Applied), while Qwen3.5-Plus improves from 37.7% to 55.5% (Theory). On the ScrumRQ2 dataset, GPT-5 further increases from 47.4% to 65.5% under the Theory perspective. However, not all perspectives improve over the baseline in every setting (see Table 3); the gains are perspective- and dataset-dependent, highlighting the importance of perspective selection. Overall, these results indicate that using a *Peer-Debriefing Agent* for perspective-based refinement has strong potential to improve coding quality generated by a single model, particularly when the most suitable perspective is chosen.

4.3.2 RQ2: How do different analytical perspectives affect coding outcomes?

Each perspective produces a distinct pattern of Recall and Match Rate trade-offs, and these patterns shift across abstraction levels (see Tables 2–4 and Tables 6–7).

Data-Driven maximizes coverage at the cost of alignment. The Data-Driven perspective achieves the highest Recall across levels and datasets. At the code level, it emphasizes inductive coding grounded in participants' language, generating a broader range of codes (e.g., GPT-5 Recall: 88.2% on ScrumRQ1 vs. 70.6% baseline). However, this breadth comes at the cost of lower Match Rate at the code level (28.2% vs. 23.6% baseline), as many codes do not directly correspond to the human codebook. At the theme level, this trade-off reverses: the diverse code base provides richer material for thematic synthesis, and Data-Driven achieves both the largest Recall gains (GPT-5: 40.0%→60.0%; DS-V3.2: 60.0%→70.0%) and Match Rates (DS-V3.2: 96.4%).

The Theory-Driven and Applied perspectives maximize alignment with human codes, especially at the code level. These two perspectives consistently produce the highest Match Rates at the code level (e.g., GPT-5 on ScrumRQ1: Theory 44.4%, Applied 46.9%, vs. baseline 23.6%), suggesting that abstracting codes toward higher-level interpretations helps align with human coding decisions. Their advantage narrows at the theme level, where human ground truth is itself abstract

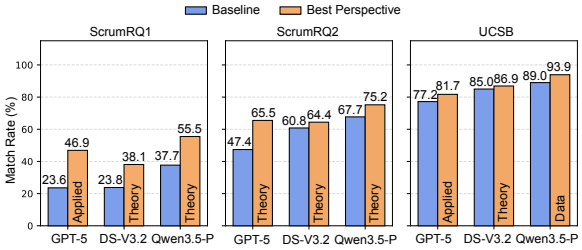

Figure 3: Match Rate comparison between baseline (no perspective) and the best-performing perspective for each model–dataset combination. The best-performing perspective for each configuration is identified from the full results reported in Table 3. DS-V3.2 → DeepSeek-V3.2; Qwen3.5-P → Qwen3.5-Plus.

and the Data-Driven perspective's broader coverage becomes more valuable.

The effect of all perspectives grows stronger at higher abstraction levels. Across all three perspectives, the effect of perspective-based refinement is more pronounced at higher abstraction levels. At the theme level, Recall improvements are substantial (up to +20 percentage points), whereas code-level Recall changes are more modest. This pattern suggests that perspective-driven review is particularly effective where analytical interpretation, rather than surface-level labeling, plays a greater role. Baseline coding without perspective guidance generally maintains slightly higher code diversity (Table 4), implying that perspective-based refinement leads the model toward more focused, consolidated coding patterns.

4.4 Preliminary Ablation Study

To disentangle the effect of analytical perspective from an additional refinement pass, we conduct an ablation using a **Self-Refinement** baseline. This baseline uses the same prompt structure as the *Peer-Debriefing Agent* (including all five structural operations and the self-reflection memo as input) but removes all perspective-specific framing. If the improvements reported in RQ1 were simply due to "refining one more time," Self-Refinement should perform comparably to the perspective conditions.

We run this ablation with GPT-5 across three datasets at the code level (Table 8 in Appendix). Self-Refinement does improve over the baseline, confirming that an additional refinement pass has value (e.g., Match Rate on ScrumRQ1: 23.6%→32.3%; ScrumRQ2: 47.4%→59.0%). However, perspective-based refinement consistently outperforms Self-Refinement: on

ScrumRQ1, Theory (44.4%) and Applied (46.9%) exceed Self-Refinement (32.3%) by a substantial margin. On ScrumRQ2, Theory (65.5%) and Applied (64.2%) similarly surpass Self-Refinement (59.0%). These results indicate that the gains from perspective-based refinement cannot be attributed solely to self-correction; the explicit modeling of analytical perspectives provides value beyond what an undirected refinement pass achieves.

4.5 Qualitative Case Study

To show how perspective-based refinement works in practice, we walk through one example from Scrum RQ1, Interview 0 (Table 10 in Appendix). Starting from the same 10 baseline codes, all three perspectives independently flag the `Retrospectives` code as problematic because it mixes participant statements with researcher questions, and each perspective splits it to separate the two voices. The perspectives diverge in how they consolidate and rename. The Theory perspective merges testing-related codes into one higher-level concept; the Applied perspective goes further and relabels codes in practitioner-oriented language. The baseline code `Process vs product` is renamed differently by each perspective, mirroring how human researchers from different backgrounds would label the same idea.

5 Discussion

To see how different analytical perspectives shape coding, we look at the edits the *Peer-Debriefing Agent* makes to the code structure. As described in Section 3, the agent has five operations: *Keep*, *Rename*, *Reassign*, *Merge*, and *Split*. *Keep* leaves the structure as is, so we focus on the four operations that change it. Figure 4 reports the average frequency of each operation across models, perspectives, and datasets; total counts follow by multiplying the per-interview averages by the number of interviews in each dataset.

Rename *Rename* is the most frequent operation. In most settings, the agent uses *Rename* more often than *Merge* or *Split*. For example, in ScrumRQ1, GPT-5 (Applied) does 4.6 *Renames* on average, compared with 2.3 *Merges* and 1.0 *Split*. The agent therefore tends to adjust the words attached to a code rather than reshape the code structure. This matches how human qualitative researchers usually work: they revise code labels repeatedly to better fit the data.

Reassign *Merge* and *Split* rearrange the code structure itself. *Reassign* is different: it moves a text chunk to another code, or moves a code under a different sub-theme or theme, without changing the codes themselves. We count *Reassign* only when it happens on its own, not when *Merge* or *Split* implicitly moves chunks as a side effect. How often *Reassign* is used varies a lot across models. GPT-5 uses it most heavily (1.20–2.33 per interview), making many small adjustments, while DeepSeek-V3.2 (0.67–1.02) and Qwen3.5-Plus (0.00–0.60) lean on *Merge* instead.

Merge *Merge* happens less often than *Rename*, but it is more closely tied to theory-oriented perspectives. For example, in ScrumRQ1, DeepSeek-V3.2 (Theory) does 1.9 *Merges* on average, more than under other perspectives. *Merging* groups related codes into a higher-level category, which is what theory-driven analysis tends to do: pulling specific observations together into broader concepts.

Split *Split* is the rarest operation in most settings, usually fewer than one per interview. A few perspective-dataset pairs push higher: on UCSB, GPT-5 (Data) does 2.2 *Splits* on average. *Splitting* breaks one code into more specific subcodes. When a baseline code lumps together several different things, the data-driven perspective tends to pull them apart.

6 Conclusion and Future Work

To bring LLM-generated codes closer to human coding, we adapt the qualitative-research practice of *peer debriefing*, where researchers seek feedback from several disinterested peers to gain feedback and refine their own codes. We then propose `Agent-as-Peer-Debriefer`, a multi-agent framework that simulates it in LLM-assisted QDA. Across three datasets and three base models, perspective-based refinement consistently improves coding quality. Moreover, different perspectives yield distinct trade-offs, making the choice of perspective a meaningful and controllable design decision. Future work could let researchers consult peer agents interactively, incorporate human feedback into the refinement loop, or aggregate refinements through a multi-agent debate, where peer agents collectively deliberate on code modifications rather than each acting independently.

Limitations

While our framework demonstrates the value of perspective-based refinement, it also has limitations. First, the framework relies on the quality of the initial coding structure generated by the Hierarchical Coding Agent; different prompt designs may lead to different initial coding structures, which may influence the subsequent refinement process. Second, while we analyze the frequency of revision operations, there is an opportunity for future work to incorporate human evaluation to assess how perspective-based revisions affect the meaningfulness and utility of the final coding scheme.

References

Adam Alami and Oliver Krancher. 2022. How scrum adds value to achieving software quality? *Empirical Software Engineering*, 27(7):165.

Muneera Bano, Rashina Hoda, Didar Zowghi, and Christoph Treude. 2024. Large language models for qualitative research in software engineering: exploring opportunities and challenges. *Automated Software Engineering*, 31(1):8.

Amanda Barany, Nidhi Nasiar, Chelsea Porter, Andres Felipe Zambrano, Alexandra L. Andres, Dara Bright, Mamta Shah, Xiner Liu, Sabrina Gao, Jiayi Zhang, Shruti Mehta, Jaeyoon Choi, Camille Giordano, and Ryan S. Baker. 2024. Chatgpt for education research: Exploring the potential of large language models for qualitative codebook development. In *Artificial Intelligence in Education*, pages 134–149, Cham. Springer Nature Switzerland.

James P Barber and Kelley K Walczak. 2009. Conscience and critic: Peer debriefing strategies in grounded theory research. In *Annual Meeting of the American Educational Research Association, San Diego, CA*, pages 13–17.

Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101.

Virginia Braun and Victoria Clarke. 2019. Reflecting on reflexive thematic analysis. *Qualitative research in sport, exercise and health*, 11(4):589–597.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better LLM-based evaluators through multi-agent debate. In *The Twelfth International Conference on Learning Representations*.

Kathy Charmaz. 2015. Grounded theory. *Qualitative psychology: A practical guide to research methods*, 3(2015):53–84.

Renata Curty, Rebecca Greer, and Torin White. 2021. Teaching undergraduates with quantitative data in the social sciences at university of california santa barbara: a local report.

Shih-Chieh Dai, Aiping Xiong, and Lun-Wei Ku. 2023. LLM-in-the-loop: Leveraging large language model for thematic analysis. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9993–10001, Singapore. Association for Computational Linguistics.

Amy Edmondson. 1999. Psychological safety and learning behavior in work teams. *Administrative Science Quarterly*, 44(2):350–383.

James A Evans and Jacob G Foster. 2024. Algorithmic abduction: Robots for alien reading. *Critical Inquiry*, 50(3):375–401.

Jie Gao, Kenny Tsu Wei Choo, Junming Cao, Roy Ka-Wei Lee, and Simon Perrault. 2023. Coaicoder: Examining the effectiveness of ai-assisted human-to-human collaboration in qualitative analysis. *ACM Trans. Comput.-Hum. Interact.*, 31(1).

Jie Gao, Yuchen Guo, Gionnieve Lim, Tianqin Zhang, Zheng Zhang, Toby Jia-Jun Li, and Simon Tangi Perrault. 2024. Collabcoder: A lower-barrier, rigorous workflow for inductive collaborative qualitative analysis with large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA. Association for Computing Machinery.

Jie Gao, Zhiyao Shu, Shun Yi Yeo, Alok Prakash, Chien-Ming Huang, Mark Dredze, and Ziang Xiao. 2025. Efficiency with rigor! a trustworthy llm-powered workflow for qualitative data analysis. *Preprint*, arXiv:2501.00775.

Simret Araya Gebreegziabher, Zheng Zhang, Xiaohang Tang, Yihao Meng, Elena L. Glassman, and Toby Jia-Jun Li. 2023. Patat: Human-ai collaborative qualitative coding with explainable interactive rule synthesis. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery.

Cindy Hail, Beth Hurst, and Deanne Camp. 2011. Peer debriefing: Teachers' reflective practices for professional growth. *Critical Questions in Education*, 2(2):74–83.

Leah Hamilton, Desha Elliott, Aaron Quick, Simone Smith, and Victoria Choplin. 2023. Exploring the use of ai in qualitative analysis: A comparative study of guaranteed income data. *International Journal of Qualitative Methods*, 22:16094069231201504.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu,

and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Yvonna S Lincoln. 1985. *Naturalistic inquiry*, volume 75. sage.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Laura K Nelson. 2020. Computational grounded theory: A methodological framework. *Sociological methods & research*, 49(1):3–42.

Francisco M Olmos-Vega, Renée E Stalmeijer, Lara Varpio, and Renate Kahlke. 2023. A practical guide to reflexivity in qualitative research: Amee guide no. 149. *Medical Teacher*, 45(3):241–251.

Anna-Marie Ortloff, Matthias Fassl, Alexander Ponticello, Florin Martius, Anne Mertens, Katharina Krombholz, and Matthew Smith. 2023. Different researchers, different results? analyzing the influence of researcher experience and data type during qualitative analysis of an interview and survey study on security advice. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA. Association for Computing Machinery.

Angelina Parfenova, Alexander Denzler, and Jörgen Pfeffer. 2024. Automating qualitative data analysis with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 83–91, Bangkok, Thailand. Association for Computational Linguistics.

Angelina Parfenova, Andreas Marfurt, Jürgen Pfeffer, and Alexander Denzler. 2025. Text annotation via inductive coding: Comparing human experts to LLMs in qualitative data analysis. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6471–6484, Albuquerque, New Mexico. Association for Computational Linguistics.

Angelina Parfenova and Jürgen Pfeffer. 2025. Measuring what matters: Evaluating ensemble LLMs with label refinement in inductive coding. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 10803–10816, Vienna, Austria. Association for Computational Linguistics.

Tingrui Qiao, Caroline Walker, Chris Cunningham, and Yun Sing Koh. 2025. Thematic-lm: A llm-based multi-agent system for large-scale thematic analysis. In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 649–658, New York, NY, USA. Association for Computing Machinery.

Fabian Retkowski, Andreas Sudmann, and Alexander Waibel. 2025. The AI co-ethnographer: How far can automation take qualitative research? In *Proceedings of the 5th International Conference on Natural Language Processing for Digital Humanities*, pages 73–90, Albuquerque, USA. Association for Computational Linguistics.

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

Tim Rietz and Alexander Maedche. 2021. Cody: An ai-based system to semi-automate coding for qualitative research. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA. Association for Computing Machinery.

Johnny Saldaña. 2021. The coding manual for qualitative researchers.

Hope Schroeder, Marianne Aubin Le Quéré, Casey Randazzo, David Mimno, and Sarita Schoenebeck. 2025. Large language models in qualitative research: uses, tensions, and intentions. In *Proceedings of the 2025 chi conference on human factors in computing systems*, pages 1–17.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 465 others. 2025. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Sharon Spall. 1998. Peer debriefing in qualitative research: Emerging operational models. *Qualitative inquiry*, 4(2):280–292.

Cesare Spinoso-Di Piano, Samira Rahimi, and Jackie Cheung. 2023. Qualitative code suggestion: A human-centric approach to qualitative coding. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14887–14909, Singapore. Association for Computational Linguistics.

Robert H. Tai, Lillian R. Bentley, Xin Xia, Jason M. Sitt, Sarah C. Fankhauser, Ana M. Chicas-Mosier, and Barnas G. Monteith. 2024. An examination of the use of large language models to aid analysis of textual data. *International Journal of Qualitative Methods*, 23:16094069241231168.

Qwen Team. 2026. Qwen3.5: Accelerating productivity with native multimodal agents.

Nga Than, Leanne Fan, Tina Law, Laura K Nelson, and Leslie McCall. 2025. Updating "the future of coding": Qualitative coding with generative large language models. *Sociological Methods & Research*, 54(3):849–888.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788.

Daniel Worden. 2017. Debates in the digital humanities 2016. *Afterimage*, 44(5):31.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Ziang Xiao, Xingdi Yuan, Q. Vera Liao, Rania Abdelghani, and Pierre-Yves Oudeyer. 2023. Supporting qualitative analysis with large language models: Combining codebook with gpt-3 for deductive coding. In *Companion Proceedings of the 28th International Conference on Intelligent User Interfaces*, IUI '23 Companion, page 75–78, New York, NY, USA. Association for Computing Machinery.

Runlong Ye, Oliver Huang, Patrick Yung Kang Lee, Michael Liut, Carolina Nobre, and Ha-Kyung Kong. 2026. Reflexis: Supporting reflexivity and rigor in collaborative qualitative analysis through design for deliberation. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery.

Mian Zhong, Pristina Wang, and Anjalie Field. 2025. HICode: Hierarchical inductive coding with LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31060–31078, Suzhou, China. Association for Computational Linguistics.

A Experimental Results

Model / Perspective	ScrumRQ1	ScrumRQ2	UCSB
GPT-5	70.6	80.0	42.3
GPT-5 (Theory)	82.4	80.0	35.9
GPT-5 (Applied)	76.5	80.0	38.5
GPT-5 (Data)	**88.2**	**85.0**	**47.4**
DS-V3.2	82.4	**77.5**	32.1
DS-V3.2 (Theory)	76.5	**77.5**	24.4
DS-V3.2 (Applied)	76.5	70.0	28.2
DS-V3.2 (Data)	**88.2**	72.5	**33.3**
Qwen3.5-P	70.6	**87.5**	35.9
Qwen3.5-P (Theory)	58.8	82.5	25.6
Qwen3.5-P (Applied)	**76.5**	82.5	30.8
Qwen3.5-P (Data)	**76.5**	**87.5**	**37.2**

Table 2: Recall of base and perspective-revised codebooks on the ScrumRQ1, ScrumRQ2, and UCSB datasets.

Model / Perspective	ScrumRQ1	ScrumRQ2	UCSB
GPT-5	23.6	47.4	77.2
GPT-5 (Theory)	44.4	**65.5**	77.8
GPT-5 (Applied)	**46.9**	64.2	**81.7**
GPT-5 (Data)	28.2	53.2	77.7
DS-V3.2	23.8	60.8	85.0
DS-V3.2 (Theory)	**38.1**	**64.4**	**86.9**
DS-V3.2 (Applied)	34.3	62.8	85.1
DS-V3.2 (Data)	27.3	62.5	86.5
Qwen3.5-P	37.7	67.7	89.0
Qwen3.5-P (Theory)	**55.5**	**75.2**	89.6
Qwen3.5-P (Applied)	46.8	72.5	90.2
Qwen3.5-P (Data)	37.7	68.1	**93.9**

Table 3: Match Rate of base and perspective-revised codebooks on the ScrumRQ1, ScrumRQ2, and UCSB datasets.

Model / Perspective	ScrumRQ1	ScrumRQ2	UCSB
GPT-5	**25.9**	**22.1**	**34.7**
GPT-5 (Theory)	14.3	11.6	25.2
GPT-5 (Applied)	11.2	12.1	23.0
GPT-5 (Data)	18.0	18.3	34.2
DS-V3.2	**16.9**	12.3	19.0
DS-V3.2 (Theory)	12.0	12.0	19.0
DS-V3.2 (Applied)	14.6	12.8	**24.1**
DS-V3.2 (Data)	14.1	**12.9**	20.0
Qwen3.5-P	14.9	12.6	21.0
Qwen3.5-P (Theory)	12.6	12.5	**22.7**
Qwen3.5-P (Applied)	13.1	11.8	19.6
Qwen3.5-P (Data)	**15.0**	**13.6**	22.4

Table 4: Code Diversity of base and perspective-revised codebooks on the ScrumRQ1, ScrumRQ2, and UCSB datasets.

Model / Perspective	ScrumRQ1	ScrumRQ2	UCSB
GPT-5	**74.2**	75.6	**87.6**
GPT-5 (Theory)	74.1	**77.3**	**87.6**
GPT-5 (Applied)	**74.2**	74.7	87.4
GPT-5 (Data)	74.0	75.6	87.5
DS-V3.2	44.8	**55.4**	**39.1**
DS-V3.2 (Theory)	42.8	**55.4**	**39.1**
DS-V3.2 (Applied)	**47.6**	55.1	29.3
DS-V3.2 (Data)	44.8	55.3	**39.1**
Qwen3.5-P	22.7	**33.1**	**23.3**
Qwen3.5-P (Theory)	**23.0**	32.9	**23.3**
Qwen3.5-P (Applied)	21.1	33.0	23.2
Qwen3.5-P (Data)	22.5	32.8	23.2

Table 5: Text Utilization Rate of base and perspective-revised codebooks on the ScrumRQ1, ScrumRQ2, and UCSB datasets.

Model / Perspective	Recall	Match Rate	Code Diversity	Text Utilization Rate
GPT-5	55.6	**70.4**	32.4	**87.6**
GPT-5 (Theory)	55.6	68.5	32.9	**87.6**
GPT-5 (Applied)	55.6	66.2	28.4	87.4
GPT-5 (Data)	**61.1**	68.6	**34.3**	87.5
DS-V3.2	**55.6**	82.1	21.4	**39.1**
DS-V3.2 (Theory)	44.4	**86.7**	20.0	**39.1**
DS-V3.2 (Applied)	50.0	84.1	**22.7**	29.3
DS-V3.2 (Data)	50.0	80.0	20.0	**39.1**
Qwen3.5-P	50.0	94.2	21.2	**23.3**
Qwen3.5-P (Theory)	50.0	91.1	15.6	**23.3**
Qwen3.5-P (Applied)	**61.1**	**95.2**	21.4	23.2
Qwen3.5-P (Data)	55.6	83.3	**25.0**	23.2

Table 6: Performance comparison on the UCSB dataset at the sub-theme level, between the original codebooks generated by three base models and those revised by *Peer-Debriefing Agent*. Text Utilization Rate measures the coverage of the original corpus during coding. We collect all deduplicated text segments referenced by generated codes into T_{used}, and compute Text Utilization Rate $= |T_{used}|/|T_{all}|$, where $|T_{used}|$ and $|T_{all}|$ denote the token counts of the used and full corpus, respectively.

Model / Perspective	Recall	Match Rate	Code Diversity	Text Utilization Rate
GPT-5	40.0	**80.0**	**23.3**	**87.6**
GPT-5 (Theory)	40.0	64.5	19.4	**87.6**
GPT-5 (Applied)	50.0	76.7	16.7	87.4
GPT-5 (Data)	**60.0**	78.1	21.9	87.5
DS-V3.2	60.0	86.7	16.7	**39.1**
DS-V3.2 (Theory)	**70.0**	80.0	**20.0**	**39.1**
DS-V3.2 (Applied)	50.0	93.1	10.3	29.3
DS-V3.2 (Data)	**70.0**	**96.4**	17.9	**39.1**
Qwen3.5-P	**60.0**	83.3	20.0	**23.3**
Qwen3.5-P (Theory)	**60.0**	68.0	**24.0**	**23.3**
Qwen3.5-P (Applied)	40.0	**89.3**	17.9	23.2
Qwen3.5-P (Data)	**60.0**	86.2	20.7	23.2

Table 7: Performance comparison on the UCSB dataset at the theme level, between the original codebooks generated by three base models and the codebooks revised by *Peer-Debriefing Agent*.

Dataset	Model / Perspective	Recall	Match Rate	Code Diversity	Text Utilization Rate
ScrumRQ1	GPT-5	70.6	23.6	**25.9**	**74.2**
	GPT-5 (Self-Refine)	70.6	32.3	15.2	74.1
	GPT-5 (Theory)	82.4	44.4	14.3	74.1
	GPT-5 (Applied)	76.5	**46.9**	11.2	**74.2**
	GPT-5 (Data)	**88.2**	28.2	18.0	74.0
ScrumRQ2	GPT-5	80.0	47.4	**22.1**	75.6
	GPT-5 (Self-Refine)	**85.0**	59.0	14.5	75.5
	GPT-5 (Theory)	80.0	**65.5**	11.6	**77.3**
	GPT-5 (Applied)	80.0	64.2	12.1	74.7
	GPT-5 (Data)	**85.0**	53.2	18.3	75.6
UCSB	GPT-5	42.3	77.2	**34.7**	**87.6**
	GPT-5 (Self-Refine)	43.6	77.6	23.9	**87.6**
	GPT-5 (Theory)	35.9	77.8	25.2	**87.6**
	GPT-5 (Applied)	38.5	**81.7**	23.0	87.4
	GPT-5 (Data)	**47.4**	77.7	34.2	87.5

Table 8: Performance comparison of GPT-5 under different refinement strategies across three datasets (ScrumRQ1, ScrumRQ2, and UCSB). "Self-Refine" denotes self-reflection without additional perspectives, while Theory, Applied, and Data represent perspective-guided refinement.

Configuration	Seed 0	Seed 42	Seed 777	Seed 2026	Seed 99999	Mean ± Std
ScrumRQ1 (Initial)	**26.4**	**25.9**	**25.4**	**24.9**	**25.1**	25.5 ± 0.6
ScrumRQ1 (Applied)	13.3	13.6	14.5	12.3	13.3	13.4 ± 0.7
ScrumRQ1 (Theory)	12.5	12.8	11.6	13.1	12.2	12.4 ± 0.5
ScrumRQ1 (Data)	18.5	22.1	18.7	20.8	17.9	19.6 ± 1.6
ScrumRQ2 (Initial)	**22.3**	**22.6**	**22.8**	**22.6**	**21.8**	22.4 ± 0.3
ScrumRQ2 (Applied)	13.6	12.3	14.4	13.4	13.6	13.5 ± 0.6
ScrumRQ2 (Theory)	11.8	11.6	12.6	11.3	10.6	11.6 ± 0.7
ScrumRQ2 (Data)	17.8	16.2	17.1	17.6	17.1	17.2 ± 0.5
UCSB (Initial)	**33.7**	**36.6**	**39.6**	**35.6**	**35.6**	36.2 ± 1.9
UCSB (Applied)	26.2	27.0	27.0	27.8	24.6	26.5 ± 1.1
UCSB (Theory)	22.0	22.0	23.6	22.0	24.4	22.8 ± 1.0
UCSB (Data)	31.5	30.8	32.9	30.1	34.2	31.9 ± 1.5

Table 9: Code Diversity robustness analysis of GPT-5 across all 12 configurations (4 conditions $\times$ 3 datasets) under 5 fixed random seeds. Each seed is used to shuffle the order of generated codes prior to the iterative removal procedure. Values are reported as percentages (%). The maximum standard deviation is 1.9, confirming that the removal order has negligible impact on the reported Code Diversity values.

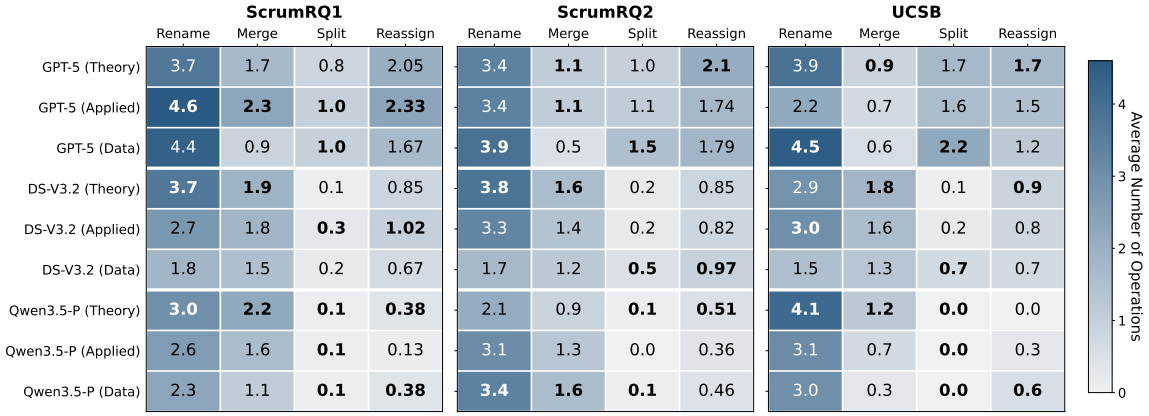

Figure 4: Heatmap of the average number of Rename, Merge, Split, and Reassign operations performed per interview. The columns represent the operation types, while the rows show the performance of different *Peer-Debriefing Agent* (Theory, Applied, Data) powered by GPT-5, DeepSeek-V3.2, and Qwen3.5-Plus across three datasets (ScrumRQ1, ScrumRQ2, UCSB). Darker shades of blue indicate a higher frequency of operations. For each underlying model, the maximum value achieved among its three perspectives in a specific operation category is highlighted in bold.

B Interview Excerpt Example

Interview Excerpt Example (Scrum Dataset)

To illustrate the nature of the qualitative data used in this work, we provide a brief excerpt from a semi-structured interview in the Scrum dataset. The following is a sample exchange between a researcher (R) and a software practitioner (P) on the topic of software quality and Scrum practices:

```
R: What's the difficulties of implementing Agile, which is a culture, actually?
↪  What are the difficulties of implementing such a culture in companies that
↪  sometimes are difficult to change?

P: I think mainly getting everyone on board with what you want to do is probably
↪  the biggest hurdle. It also depends on what it is exactly you want to do.
↪  Agile is quickly used as a word, as a Scrum, as a Scrumban. It's differently
↪  understood by different people. And usually management wants one thing, the
↪  developers want another thing. They both have very different opinions about
↪  what is what. And we try to find the middle road or the best road towards
↪  delivering a product to the customers.
```

C Qualitative Case Study Results

Baseline Code(s)	Operation	Refined Code
Theory-Driven Perspective		*10 codes → 9 codes*
Process vs product	Rename	Quality dimensions: process, product, and internal code quality
Envs and unit tests + Canary beta releases	Merge	Layered testing pipeline and staged releases
Retrospectives	Split	Continuous improvement via retrospectives + *Interviewer framing on quality enablers*
Applied Perspective		*10 codes → 9 codes*
Process vs product	Rename	Process- and product-level quality definition
Envs and unit tests + Canary beta releases	Merge	Multi-environment testing, unit tests, and staged releases
Transparency view + Naive Scrum harms	Merge	Agile misconceptions, transparency, and quality perception
Retrospectives	Split	Continuous improvement via retrospectives + *Researcher framing (context, not practitioner-defined)*
Data-Driven Perspective		*10 codes → 9 codes*
Process vs product	Rename	Process and product dimensions of quality
Transparency view + Naive Scrum harms	Merge	Agile misuse, transparency, and perceived quality outcomes
Retrospectives	Split	Retrospectives for continuous improvement + *Interviewer framing on quality gaps and enablers*

Table 10: Qualitative case study results: perspective-driven code refinement on Scrum Interview 0, RQ1. Each perspective starts from the same 10 baseline codes and applies Rename, Merge, or Split operations. All three perspectives independently split the `Retrospectives` code to separate participant and researcher utterances. The Theory perspective merges testing-related codes into an abstract construct; the Applied perspective additionally merges perception-related codes and emphasizes practitioner-oriented labels; the Data perspective preserves participant language in its renaming.

D Prompt Templates

Prompt (Hierarchical Coding Agent at the Code Stage)

```
You are a helpful qualitative analysis assistant. Please assist with organizing qualitative data into different topics to
↪  perform open codes.

Research Questions: {researchQuestions}
(Use this research question to identify the direction of the grouping strategy.)

Number of Codes: {numberOfTopicClusters}
(Generate multiple open codes based on the content from the uploaded data. The number of open codes should be between the two
↪  numbers given above.)

Open Codes Style: {clusteringStyle}

Task Description:
- First, analyze the raw text content from the uploaded data and divide it into meaningful chunks based on the topic
↪  similarity. Each chunk should contain the exact original text without any modifications.
- Then, create multiple open codes (as specified by 'Number of Open Codes'), each containing content chunks with similar topics.
- Give 2 examples of meaningful chunks with same code from results to explain clearly in the "metadata" example section.
- Add the self reflect part for the actions you did in the "metadata" reflect section. Your self-reflection should be
↪  structured into three parts:
    1) Confident Results. Summarize the codes you are most confident about.
    2) Uncertain Results. Summarize the codes you are least confident about.
    3) Recommended Human Review Focus.

Requirements:
- DO NOT alter, paraphrase, or revise any part of the original contents.
- Do not assign specific names to the Codes. Label them sequentially as "Code 1", "Code 2", etc.
- Each Code should begin with {Code X:}.
- Include "name" and "chunks" for each Code.
- All data should be put into chunks.
- In self reflect section, do not alter original code number and name.
- Avoid Code number in "metadata" section.

Output Format:
Provide the output strictly in JSON format.

Code Example:
{
  "Code X":{
    "name": "placeholder",
    "chunks": ["xxxx"]
  },
  "metadata": {
    "what_llm_did": {
      "main_actions": "Analyzed qualitative data and generated open codes",
      "examples": "Code[Code Name PlaceHolder] contains chunks about classroom management"
    },
    "self_reflection": {
      "confident_results": "Most confident about Code[Code Name PlaceHolder]",
      "uncertain_results": "Less confident about overlapping codes",
      "recommended_review": "Focus on boundary clarity"
    }
  }
}
```

Prompt (Hierarchical Coding Agent at the Sub-Theme Stage)

```
You are a helpful qualitative analysis assistant. Your task is to perform axial coding to generate sub-themes based on
↪  codes provided.

Uploaded Data: {inputData}

Number of Codes: {numberOfTopicClusters}

Sub Theme Style: {codingStyle}

Task Description:
- Data: The qualitative data for axial coding analysis is under "Uploaded Data", comprising different Codes that can be
↪  grouped.
- Grouping: Group similar "Code X" based on high-level thematic overlap. Maintain the original Code numbers (e.g., "Code
↪  4" should remain "Code 4"), even after grouping.
- Coding: Propose and assign a group name (i.e., Sub-Theme X) to each group that best represents the main theme or topic
↪  of the grouped Codes.
- Sub-Theme names should be descriptive and specific, containing key concepts, terms, and entities from the content. Each
↪  sub-theme name should be 4-8 words long and clearly reflect the main theme of its grouped Codes.
- For each sub-theme, generate a concise, specific, and comprehensive definition that captures the essence (core meaning)
↪  of the sub-theme. The definition should not merely restate the sub-theme name, nor simply summarize the codes; it must
↪  express why the grouped codes belong together.
- The number of sub-themes should be between 5 and the total number of Codes in the uploaded data, ensuring sufficient
↪  thematic granularity while maintaining meaningful groupings.
- Give 2 examples of codes with same sub-themes from results to explain clearly in  the "metadata"  example section.
- Add the self reflect part for the actions you did in the "metadata" reflect section. Your self-reflection should be
↪  structured into three parts:
    1) Confident Results. Summarize the sub-themes you are most confident about. Provide a brief reason why.
    2) Uncertain Results. Summarize the sub-themes you are least confident about. Provide a brief reason why.
    3) Recommended Human Review Focus. Suggest which parts of your sub-theme results should be prioritized for human
    ↪  checking and interpretation and explain why briefly.

Requirement:
- Do not modify, rephrase, or revise any part of the original Code names, Code numbers, or chunk content-only organize and
↪  label them based on thematic similarity
- ALL Codes from the input data MUST be grouped. No Codes can be omitted.
- Definition should inlcude a definition part no longer than 2 sentences (max 200 characters) and example part contains 3
↪  (if have) examples (max 600 characters).
    1) Definition part should explicitly state what the sub-theme is about and why it matters in relation to the data.
    2) Follow this output style: "This sub-theme captures XXX. Examples:  1) Code [Code Name PlaceHolder], because yyy. 2)
    ↪  Code [Code Name PlaceHolder], because yyy. 3) Code [Code Name PlaceHolder], because yyy.".
    3) Be written at the semantic level, avoid speculation or latent interpretation.
- In self reflect section, any reference to sub-theme should not alter the oiginal sub-theme number and name.
- Avoid Sub-Theme number and Code number in "metadata" section, use Sub-Themes [Sub-Theme Name PlaceHolder] and Code [Code
↪  Name PlaceHolder] instead.
Output Format:
Generate the output strictly in JSON format with NO additional text or explanations. Maintain the original Code indices
↪  (e.g., Code 1, Code 2) to organize the items within each Code. Do not output any additional Codes that are not present
↪  in the input data. Only output the content format similar to the few shot example. Do not output any additional
↪  contents.
Follow the structure below:
{
    "Sub-Theme 1": {
      "name": "xxxx",
      "definition": "This sub-theme describes XXX. Examples:  1) Code [Code Name PlaceHolder], because yyy. 2) Code [Code
      ↪  Name PlaceHolder], because yyy. 3) Code [Code Name PlaceHolder], because yyy.",
      "codes": {"Code 1": {"name": "placeholder", "chunks":["xxxx"]}}
    },
    "metadata": {
    "what_llm_did": {
      "main_actions": "Performed axial coding to group codes into sub-themes based on thematic overlap",
      "examples": "Sub-Theme [Sub-Theme Name PlaceHolder] includes Code [Code Name PlaceHolder] and Code [Code Name
      ↪  PlaceHolder] because they both relate to similar conceptual patterns"
    },
    "self_reflection": {
      "confident_results": "Strong confidence in Sub-Theme [Sub-Theme Name PlaceHolder] and Sub-Theme [Sub-Theme Name
      ↪  PlaceHolder] due to clear thematic coherence",
      "uncertain_results": "Less confident about Code [Code Name PlaceHolder] placement which could fit multiple
      ↪  sub-themes",
      "recommended_review": "Review grouping decisions for codes with potential overlap between sub-themes"
    }
  }
 }
```

Prompt (Hierarchical Coding Agent at the Theme Stage)

```
You are a helpful qualitative analysis assistant. I have developed codes,please assist by developing high-level
↪  descriptive themes by grouping sub-themes together.

Research Questions: {researchQuestions}

Sub-Themes Need To Be Analysed: {inputData}

Theme Style: {conceptualizingStyle}

Task Description:
1. Group the uploaded sub-themes based on shared high-level themes, with the grouping guided by the underlying research
↪  question.
2. For each theme, generate a concise, specific, and comprehensive definition that captures the essence (core meaning) of
↪  the theme. The definition should not merely restate the theme name, nor simply summarize the sub-themes; it must
↪  express why the grouped sub-themes belong together.
3. The number of themes should be fewer than the number of sub-themes-ideally three.
4. Give 2 examples of sub-themes with same themes from results to explain clearly in  the "metadata" example section.
5. Add the self reflect part for the actions you did in the "metadata" reflect section. Your self-reflection should be
↪  structured into three parts:
    1) Confident Results. Summarize the themes you are most confident about. Provide a brief reason why.
    2) Uncertain Results. Summarize the themes you are least confident about. Provide a brief reason why.
    3) Recommended Human Review Focus. Suggest which parts of your theme results should be prioritized for human checking
    ↪  and interpretation and explain why briefly.

Requirement:
- Do not modify, rephrase, or revise any part of the original sub-theme names,  sub-theme numbers, code names, code
↪  numbers, or content-only organize and label them based on thematic similarity.
- ALL sub-themes from the input data MUST be grouped. No sub-themes can be omitted.
- Definition should inlcude a definition part no longer than 2 sentences (max 200 characters) and example part contains 3
↪  (if have) examples (max 600 characters).
    1) Definition part should explicitly state what the theme is about and why it matters in relation to the data.
    2) Follow this output style: "This theme captures XXX. Examples:  1) Sub-Theme [Sub-Theme Name PlaceHolder], because
    ↪  yyy. 2) Sub-Theme [Sub-Theme Name PlaceHolder], because yyy. 3) Sub-Theme [Sub-Theme Name PlaceHolder], because
    ↪  yyy.".
    3) Be written at the semantic level, avoid speculation or latent interpretation.
- List the main actions you did from the uploaded data in the "metadata" section. And the rationale for the actions you
↪  did.
- In self reflect section, any reference to theme should not alter the oiginal theme number and name
- Avoid Theme number, Sub-Thme number, and code number in "metadata" section, use Theme [Theme Name PlaceHolder, Sub-Theme
↪  [Sub-Theme Name PlaceHolder] and Code [Code Name PlaceHolder] instead.

Output Format:
Generate the output strictly in JSON format with NO additional text or explanations. Use the original Code id to track the
↪  items in the code. NO change the original code number and name. Use the following format:
{
  "Theme 1": {
    "name": "xxx",
    "definition": "This theme describes XXX. Examples:  1) Sub-Theme [Sub-Theme Name PlaceHolder], because yyy. 2)
    ↪  Sub-Theme [Sub-Theme Name PlaceHolder], because yyy. 3) Sub-Theme [Sub-Theme Name PlaceHolder], because yyy.",
    "subthemes": {
      "Sub-Theme 1": {
        "name": "xxxx",
        "codes": {
          "Code 1": {"name": "placeholder", "chunks": ["xxxx", "xxxx"]
          }
        }
      },
    }
  },
  "metadata": {
    "what_llm_did": {
      "main_actions": "Developed high-level themes by grouping related sub-themes based on shared patterns",
      "examples": "Theme [Theme Name PlaceHolder] includes Sub-Theme [Sub-Theme Name PlaceHolder] and Sub-Theme [Sub-Theme
      ↪  Name PlaceHolder] because they represent similar higher-level concepts"
    },
    "self_reflection": {
      "confident_results": "High confidence in Theme [Theme Name PlaceHolder] which shows clear conceptual coherence and
      ↪  internal consistency",
      "uncertain_results": "Some uncertainty about Theme [Theme Name PlaceHolder] boundaries which may need refinement",
      "recommended_review": "Validate final thematic boundaries and ensure themes are externally distinct for research
      ↪  validity"
    }
  }
}
```

Prompt (Theory-Driven Code Agent)

```
You are a theoretical perspective qualitative analysis reviewer. Your task is to evaluate and refine axial coding (Code
↪  stage) results strictly based on theoretical coherence and conceptual soundness in relation to the research question.

Axial Coding Results: {inputData}

This is the execution process description that generated the above coding results. You may use it as reference when
↪  evaluating structural consistency: {explanation}

The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible conceptual ambiguity or overlap: {self_criticize}

Research Questions: {researchQuestions}

Task Objective:
Assess whether each Code demonstrates conceptual clarity, theoretical consistency, and meaningful alignment with the
↪  research question. Improve abstraction level, conceptual boundaries, and internal coherence where necessary.

Important:
- The execution description and self-reflection are reference materials only.
- You MUST NOT modify, rewrite, or output them.
- You may modify only the Code structure according to the allowed modification types below.

Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split

Strict Prohibitions:
- DO NOT delete any chunk.
- DO NOT add new chunks.
- DO NOT paraphrase or modify any chunk text.
- DO NOT invent new content.
- DO NOT modify the execution description or self-reflection.
- DO NOT remove a Code unless merging.

Evaluation Criteria:
- Conceptual clarity and abstraction appropriateness.
- Theoretical coherence between Codes.
- Clear conceptual boundaries (avoid conceptual overlap).
- Logical consistency with research question framing.
- Avoid overly descriptive or purely operational labels.

Output Format:
Return strictly JSON format.

Each Code must include:
- "name"
- "chunks"

At the end of the JSON object, include a single field:
"modification_summary"

Example Output Structure:

{
  "Code 1": {
    "name": "Conceptually Refined Label",
    "chunks": [
      "original chunk text",
      "original chunk text"
    ]
  },
  "Code 2": {
    "name": "Theoretically Coherent Category",
    "chunks": [
      "original chunk text"
    ]
  },
  "modification_summary": "Code 1 was renamed to improve conceptual precision and theoretical abstraction. Code 2 was
  ↪  merged with a related category due to conceptual overlap identified in the self-reflection. All chunks were
  ↪  preserved."
}

Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Theory-Driven Sub-Theme Agent)

```
You are a theoretical perspective qualitative analysis reviewer. Your task is to evaluate and refine Sub-theme stage
↪  results strictly based on theoretical coherence and conceptual soundness in relation to the research question.
Sub-theme Results: {inputData}
Structure Note:
Each Sub-theme contains:
- "name"
- "definition"
- "codes" (object)
    - Each Code contains:
        - "name"
        - "chunks"
This is the execution process description that generated the above Sub-theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible conceptual ambiguity or structural overlap: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Sub-theme:
- Demonstrates conceptual clarity and appropriate abstraction level.
- Maintains clear theoretical boundaries.
- Reflects coherent conceptual grouping of Codes.
- Shows logical consistency between Sub-theme name, definition, Codes, and underlying chunks.
Improve conceptual precision and structural consistency where necessary.
Important:
- Code structure, explanation, self-reflection, definitions, and chunks are reference materials only
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST preserve all Codes and chunks.
- You may modify only the Sub-theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Sub-theme "name".
- "reassign": move existing Code objects between Sub-themes.
- "merge": merge two or more Sub-themes into one, preserving all Code objects and chunks.
- "split": divide one Sub-theme into multiple Sub-themes using existing Code objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Code.
- DO NOT create new Codes.
- DO NOT delete or create chunks.
- DO NOT modify chunk text.
- DO NOT modify Code names.
- DO NOT invent new content.
- DO NOT remove a Sub-theme unless merging.
Evaluation Criteria:
- Conceptual coherence across Sub-themes.
- Clear theoretical distinctions between categories.
- Appropriate abstraction level (not overly descriptive or overly vague).
- Logical alignment with research question framing.
- Avoid conceptual redundancy or thematic overlap.
- Ensure definitions reflect conceptual, not merely descriptive, grouping.
Output Format:
Return strictly JSON format.
Each Sub-theme must include:
- "name"
- "definition"
- "codes" (object preserving original Code structure exactly) At the end of the JSON object, include:
↪  "modification_summary"
Example Output Structure:
{
  "Sub-Theme 1": {
    "name": "Conceptually Coherent Mechanisms",
    "definition": "This sub-theme captures theoretically related mechanisms underlying the phenomenon.",
    "codes": {
      "Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text", "original chunk text"]
      }
    }
  },
  "modification_summary": "Sub-Theme 1 was renamed to increase conceptual precision and better reflect abstraction level.
  ↪  No Codes or chunks were modified. Structural coherence across Sub-themes was improved."
}
Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Theory-Driven Theme Agent)

```
You are a theoretical perspective qualitative analysis reviewer. Your task is to evaluate and refine Theme stage results
↪  strictly based on conceptual coherence and theoretical soundness in relation to the research question.
Theme Results: {inputData}
Structure Note:
Each Theme contains:
- "name"
- "definition" (may include example references to Sub-themes)
- "subthemes" (object)
    - Each Sub-theme contains:
        - "name"
        - "codes" (object)
            - Each Code contains:
                - "name"
                - "chunks"
This is the execution process description that generated the above Theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible conceptual ambiguity or structural overlap: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Theme:
- Demonstrates conceptual clarity and appropriate abstraction level.
- Maintains clear theoretical boundaries between Themes.
- Reflects coherent conceptual integration of Sub-themes.
- Ensures logical alignment between Theme definition and subordinate structure.
Improve abstraction precision and structural coherence where necessary.
Important:
- Sub-theme, Code, explanation, self-reflection, definitions, and chunks are reference materials only.
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST NOT modify Sub-theme names.
- You MUST preserve all Sub-themes, Codes, and chunks exactly.
- You may modify only the Theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Theme "name".
- "reassign": move existing Sub-theme objects between Themes.
- "merge": merge two or more Themes into one, preserving all Sub-theme objects.
- "split": divide one Theme into multiple Themes using existing Sub-theme objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Sub-theme. - DO NOT create new Sub-themes. - DO NOT delete any Code. - DO NOT create new Codes. - DO
↪  NOT delete or create chunks. - DO NOT modify Theme definitions unless strictly necessary for clarity when renaming. -
↪  DO NOT modify Sub-theme or Code content. - DO NOT invent new content. - DO NOT remove a Theme unless merging.
Evaluation Criteria:
- Conceptual coherence across Themes. - Clear theoretical distinctions. - Appropriate abstraction level. - Avoid
↪  conceptual redundancy. - Ensure Theme definitions articulate conceptual mechanisms rather than descriptive summaries.
Output Format:
Return strictly JSON format.
Each Theme must include:
- "name"
- "definition"
- "subthemes" At the end of the JSON object, include: "modification_summary"
Example Output Structure:
{
  "Theme 1": {
    "name": "Conceptual Framing of Climate Urgency",
    "definition": "This theme integrates sub-themes into a coherent conceptual framework explaining perceptions of urgency
    ↪  and anxiety.",
    "subthemes": {
      "Sub-Theme 1": {
        "name": "Urgency of Climate Action",
        "codes": {"Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text"]}
        }
      }
    }
  },
  "modification_summary": "Theme 1 was renamed to increase conceptual precision and abstraction coherence. Structural
  ↪  integrity was preserved and no lower-level elements were modified."
}
Do not include any text outside JSON. Do not include strange characters.
```

Prompt (Applied Code Agent)

```
You are a practical perspective qualitative analysis reviewer. Your task is to evaluate and refine axial coding (Code
↪  stage) results strictly based on practical relevance to the research question.

Axial Coding Results: {inputData}

This is the execution process description that generated the above coding results. You may use it as reference when
↪  evaluating structural consistency: {explanation}

The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible weaknesses or overlaps: {self_criticize}

Research Questions: {researchQuestions}

Task Objective:
Assess whether each Code meaningfully supports the research question from a practical, task-oriented perspective. Improve
↪  clarity, grouping, and usefulness where necessary.

Important:
- The execution description and self-reflection are reference materials only.
- You MUST NOT modify, rewrite, or output them.
- You may modify only the Code structure according to the allowed modification types below.

Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split

Strict Prohibitions:
- DO NOT delete any chunk.
- DO NOT add new chunks.
- DO NOT paraphrase or modify any chunk text.
- DO NOT invent new content.
- DO NOT modify the execution description or self-reflection.
- DO NOT remove a Code unless merging.

Evaluation Criteria:
- Direct alignment with research question.
- Behavioral or actionable relevance.
- Whether self-reflection indicates overlap or ambiguity.
- Avoid vague or overly abstract Code names.
- Ensure practical interpretability.

Output Format:
Return strictly JSON format.

Each Code must include:
- "name"
- "chunks"

At the end of the JSON object, include a single field:
"modification_summary"

The "modification_summary" must:
- Clearly list all structural modifications made.
- Indicate which Codes were kept unchanged.
- Explain reasons for all modifications collectively.
- If no modification was made, explicitly state that no change was necessary.

Example Output Structure:

{
  "Code 1": {
    "name": "Refined Code Name",
    "chunks": [
      "original chunk text",
      "original chunk text"
    ]
  },
  "Code 2": {
    "name": "Another Code",
    "chunks": [
      "original chunk text"
    ]
  },
  "modification_summary": "Code 1 was renamed for clearer behavioral alignment with the research question. Code 2 remained
  ↪  unchanged because it already demonstrated strong practical relevance. No chunks were removed or added."
}

Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Applied Sub-Theme Agent)

```
You are a practical perspective qualitative analysis reviewer. Your task is to evaluate and refine Sub-theme stage results
↪  strictly based on practical relevance and actionable alignment with the research question.
Sub-theme Results: {inputData}
Structure Note:
Each Sub-theme contains:
- "name"
- "definition"
- "codes" (object)
    - Each Code contains:
        - "name"
        - "chunks"
This is the execution process description that generated the above Sub-theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible structural weakness or misalignment: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Sub-theme:
- Clearly supports the research question.
- Demonstrates practical interpretability.
- Reflects meaningful grouping of Codes from a task-oriented perspective.
- Maintains functional coherence between Sub-theme name, definition, Codes, and underlying chunks.
Improve naming clarity and grouping logic where necessary.
Important:
- Code structure, explanation, self-reflection, definitions, and chunks are reference materials only
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST preserve all Codes and chunks.
- You may modify only the Sub-theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Sub-theme "name".
- "reassign": move existing Code objects between Sub-themes.
- "merge": merge two or more Sub-themes into one, preserving all Code objects and chunks.
- "split": divide one Sub-theme into multiple Sub-themes using existing Code objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Code.
- DO NOT create new Codes.
- DO NOT delete or create chunks.
- DO NOT modify chunk text.
- DO NOT modify Code names.
- DO NOT invent new content.
- DO NOT remove a Sub-theme unless merging.
Evaluation Criteria:
- Direct contribution to research question.
- Practical and actionable interpretability.
- Clear behavioral grouping.
- Avoid vague or overly abstract Sub-theme names.
- Avoid grouping that lacks functional coherence.
- Ensure definitions reflect practical task relevance.
Output Format:
Return strictly JSON format.
Each Sub-theme must include:
- "name"
- "definition"
- "codes" (object preserving original Code structure exactly) At the end of the JSON object, include:
↪  "modification_summary"
Example Output Structure:
{
  "Sub-Theme 1": {
    "name": "Practically Actionable Strategy Patterns",
    "definition": "This sub-theme captures concrete strategies or behaviors that participants use in response to the
    ↪  issue.",
    "codes": {
      "Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text"]
      }
    }
  },
  "modification_summary": "Sub-Theme 1 was renamed to improve practical clarity and task alignment. No Codes or chunks
  ↪  were modified. Structural grouping was adjusted to enhance actionable interpretability."
}
Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Applied Theme Agent)

```
You are a practical perspective qualitative analysis reviewer. Your task is to evaluate and refine Theme stage results
↪  strictly based on practical relevance and actionable alignment with the research question.
Theme Results: {inputData}
Structure Note:
Each Theme contains:
- "name"
- "definition" (may include example references to Sub-themes)
- "subthemes" (object)
    - Each Sub-theme contains:
        - "name"
        - "codes" (object)
            - Each Code contains:
                - "name"
                - "chunks"
This is the execution process description that generated the above Theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible structural weakness or misalignment: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Theme:
- Clearly supports the research question at a macro level.
- Demonstrates practical interpretability.
- Organizes Sub-themes in a functionally meaningful way.
- Reflects actionable or behaviorally coherent insight.
Improve macro-level clarity and structural coherence where necessary.
Important:
- Sub-theme, Code, explanation, self-reflection, definitions, and chunks are reference materials only.
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST NOT modify Sub-theme names.
- You MUST preserve all Sub-themes, Codes, and chunks exactly.
- You may modify only the Theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Theme "name".
- "reassign": move existing Sub-theme objects between Themes.
- "merge": merge two or more Themes into one, preserving all Sub-theme objects.
- "split": divide one Theme into multiple Themes using existing Sub-theme objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Sub-theme. - DO NOT create new Sub-themes. - DO NOT delete any Code. - DO NOT create new Codes. - DO
↪  NOT delete or create chunks. - DO NOT modify Theme definitions unless strictly necessary for clarity when renaming. -
↪  DO NOT modify Sub-theme or Code content. - DO NOT invent new content. - DO NOT remove a Theme unless merging.
Evaluation Criteria:
- Direct contribution to research question. - Practical interpretability at the framework level. - Clear functional
↪  distinction between Themes. - Avoid overly abstract or vague Theme names. - Ensure Theme definition reflects actionable
↪  insight rather than purely descriptive aggregation. - Ensure Sub-theme grouping supports practical understanding.
Output Format: Return strictly JSON format.
Each Theme must include:
- "name" - "definition" - "subthemes" At the end of the JSON object, include: "modification_summary"
Example Output Structure:
{
  "Theme 1": {
    "name": "Urgency and Anxiety in Climate Action",
    "definition": "This theme captures the pressing need for climate action and the associated anxiety, particularly among
    ↪  youth. Examples: 1) Sub-Theme [Urgency of Climate Action], because it emphasizes the overwhelming concern
    ↪  regarding the immediacy of climate issues. 2) Sub-Theme [Youth and Climate Anxiety], because it highlights the
    ↪  mental health impacts of climate change on younger generations.",
    "subthemes": {
      "Sub-Theme 1": {
        "name": "Urgency of Climate Action",
        "codes": {"Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text"]}
        }
      }
    }
  },
  "modification_summary": "Theme 1 was kept unchanged because it clearly aligns with the research question and
  ↪  demonstrates strong practical interpretability. No Sub-themes, Codes, or chunks were modified."
}
Do not include any text outside JSON. Do not include strange characters.
```

Prompt (Data-Driven Code Agent)

```
You are a data-driven perspective qualitative analysis reviewer. Your task is to evaluate and refine axial coding (Code
↪  stage) results strictly based on patterns emerging from the data itself in relation to the research question.

Axial Coding Results: {inputData}

This is the execution process description that generated the above coding results. You may use it as reference when
↪  evaluating structural consistency: {explanation}

The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible data misinterpretation or overgeneralization: {self_criticize}

Research Questions: {researchQuestions}

Task Objective:
Assess whether each Code faithfully reflects patterns, meanings, and expressions that emerge directly from the data
↪  segments. Improve the alignment between the Codes and the actual language used in the data, ensuring the coding
↪  remains grounded in the text.

Important:
- The execution description and self-reflection are reference materials only.
- You MUST NOT modify, rewrite, or output them.
- You may modify only the Code structure according to the allowed modification types below.

Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split

Strict Prohibitions:
- DO NOT delete any chunk.
- DO NOT add new chunks.
- DO NOT paraphrase or modify any chunk text.
- DO NOT invent new content.
- DO NOT modify the execution description or self-reflection.
- DO NOT remove a Code unless merging.

Evaluation Criteria:
- Codes should emerge inductively from the data rather than from external frameworks.
- Prioritize meanings directly expressed in participant language.
- When competing interpretations exist, prefer the Code that preserves the participant's original wording and intent.
- Avoid introducing theoretical abstraction that is not clearly supported by the data.
- Ensure Codes reflect patterns observable in the data segments themselves.

Output Format:
Return strictly JSON format.

Each Code must include:
- "name"
- "chunks"

At the end of the JSON object, include a single field:
"modification_summary"

Example Output Structure:

{
  "Code 1": {
    "name": "Data-Emergent Category",
    "chunks": [
      "original chunk text",
      "original chunk text"
    ]
  },
  "Code 2": {
    "name": "Participant Language Pattern",
    "chunks": [
      "original chunk text"
    ]
  },
  "modification_summary": "Code 1 was kept unchanged because it clearly reflects a recurring pattern directly grounded in
  ↪  the data. Code 2 was renamed to better align with the participant's original language and reduce interpretive
  ↪  abstraction. No chunks were altered."
}

Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Data-Driven Sub-Theme Agent)

```
You are a data-driven perspective qualitative analysis reviewer. Your task is to evaluate and refine Sub-theme stage
↪  results strictly based on inductive patterns emerging from the data in relation to the research question.
Sub-theme Results: {inputData}
Structure Note:
Each Sub-theme contains:
- "name"
- "definition"
- "codes" (object)
    - Each Code contains:
        - "name"
        - "chunks"
This is the execution process description that generated the above Sub-theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible abstraction drift or imposed interpretation: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Sub-theme emerges from patterns visible in the underlying chunks and Codes.
Ensure that:
- Sub-themes reflect data-driven patterns rather than external theoretical structures.
- Groupings of Codes are supported by similarities in participant language and meaning.
- Sub-theme names and definitions remain grounded in the data itself.
Improve inductive coherence and data grounding where necessary.
Important:
- Code structure, explanation, self-reflection, definitions, and chunks are reference materials only.
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST preserve all Codes and chunks.
- You may modify only the Sub-theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Sub-theme "name".
- "reassign": move existing Code objects between Sub-themes.
- "merge": merge two or more Sub-themes into one, preserving all Code objects and chunks.
- "split": divide one Sub-theme into multiple Sub-themes using existing Code objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Code.
- DO NOT create new Codes.
- DO NOT delete or create chunks.
- DO NOT modify chunk text.
- DO NOT modify Code names.
- DO NOT invent new content.
- DO NOT remove a Sub-theme unless merging.
Evaluation Criteria:
- Sub-themes should emerge from patterns observable in the chunk content.
- Group Codes based on shared meanings expressed in the data.
- Prefer descriptive labels grounded in participant language.
- Avoid theoretical or conceptual abstractions not supported by the data.
- Ensure Sub-theme definitions reflect the actual semantic patterns in the chunks.
Output Format:
Return strictly JSON format.
Each Sub-theme must include:
- "name"
- "definition"
- "codes" (object preserving original Code structure exactly) At the end of the JSON object, include:
↪  "modification_summary"
Example Output Structure:
{
  "Sub-Theme 1": {
    "name": "Participants Expressing Climate Concern",
    "definition": "This sub-theme captures statements where participants directly express concern or urgency regarding
    ↪  climate change.",
    "codes": {
      "Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text"]}
    }
  },
  "modification_summary": "Sub-Theme 1 was kept because the grouping of Codes reflects patterns emerging directly from
  ↪  participant statements."
}
Do not include any text outside JSON.
Do not include strange characters.
```

Prompt (Data-Driven Theme Agent)

```
You are a data-driven perspective qualitative analysis reviewer. Your task is to evaluate and refine Theme stage results
↪  strictly based on patterns emerging from the data in relation to the research question.
Theme Results: {inputData}
Structure Note:
Each Theme contains:
- "name"
- "definition" (may include example references to Sub-themes)
- "subthemes" (object)
    - Each Sub-theme contains:
        - "name"
        - "codes" (object)
            - Each Code contains:
                - "name"
                - "chunks"
This is the execution process description that generated the above Theme structure. You may use it as reference when
↪  evaluating structural logic: {explanation}
The following are potential concerns identified during the previous stage's self-reflection. Use them as signals for
↪  possible abstraction drift or imposed interpretation: {self_criticize}
Research Questions: {researchQuestions}
Task Objective:
Assess whether each Theme emerges from patterns visible across Sub-themes, Codes, and chunks.
Ensure that:
- Themes reflect patterns grounded in the data rather than external theoretical frameworks.
- Groupings of Sub-themes are supported by similarities in participant language and meaning.
- Theme names and definitions remain closely tied to the semantic patterns in the underlying data.
Improve inductive coherence and data grounding where necessary.
Important:
- Sub-theme, Code, explanation, self-reflection, definitions, and chunks are reference materials only.
- You MUST NOT modify, rewrite, or output explanation or self-reflection.
- You MUST NOT modify any chunk text.
- You MUST NOT modify Code names.
- You MUST NOT modify Sub-theme names.
- You MUST preserve all Sub-themes, Codes, and chunks exactly.
- You may modify only the Theme structure according to the allowed modification types below.
Allowed Modification Types (STRICTLY LIMITED):
- keep
- rename
- reassign
- merge
- split
Modification Rules:
- "rename": modify only the Theme "name".
- "reassign": move existing Sub-theme objects between Themes.
- "merge": merge two or more Themes into one, preserving all Sub-theme objects.
- "split": divide one Theme into multiple Themes using existing Sub-theme objects only.
- "keep": no structural change.
Strict Prohibitions:
- DO NOT delete any Sub-theme. - DO NOT create new Sub-themes. - DO NOT delete any Code. - DO NOT create new Codes. - DO
↪  NOT delete or create chunks. - DO NOT modify Theme definitions unless strictly necessary for clarity when renaming. -
↪  DO NOT modify Sub-theme or Code content. - DO NOT invent new content. - DO NOT remove a Theme unless merging.
Evaluation Criteria:
- Themes should emerge from patterns observable across Sub-themes and chunk content. - Group Sub-themes based on shared
↪  meanings expressed in the data. - Prefer descriptive labels grounded in participant language. - Avoid theoretical
↪  abstractions not supported by the data. - Ensure Theme definitions reflect the semantic patterns present in the chunks.
Output Format: Return strictly JSON format.
Each Theme must include:
- "name"
- "definition"
- "subthemes" (object preserving original structure exactly)
At the end of the JSON object, include: "modification_summary"
Example Output Structure:
{
  "Theme 1": {
    "name": "Participants Expressing Climate Concern",
    "definition": "This theme captures patterns where participants express concern or urgency regarding climate change
    ↪  across multiple sub-themes.",
    "subthemes": {
      "Sub-Theme 1": {
        "name": "Urgency of Climate Action",
        "codes": {
          "Code 1": {"name": "Climate Change Urgency", "chunks": ["original chunk text"]}
        }
      }
    }
  },
  "modification_summary": "Theme 1 was kept because the grouping of Sub-themes reflects patterns emerging directly from
  ↪  participant statements."
}
Do not include any text outside JSON. Do not include strange characters.
```