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

Anonymous ACL submission

Abstract

Large language models (LLMs) are increasingly used for qualitative data analysis (QDA), yet their outputs often lack the richness and nuance of human coding teams. We attribute this gap to the absence of multi-perspective review: in human QDA practice, peer debriefing, a process where colleagues with different analytical backgrounds challenge and refine each other's coding, is a standard credibility procedure. We propose a multi-agent framework that operationalizes peer debriefing for LLM-assisted QDA. The framework employs a Hierarchical Coding Agent that generates codes, sub-themes, and themes with self-reflection memos, and three Peer-Debriefing Agents that re-examine and refine results from distinct analytical stances (theory-driven, data-driven, and applied). We evaluate the framework on three datasets across two domains using three LLMs and semantic similarity-based metrics against human-annotated codes. Results show that perspective-based refinement achieves substantial alignment with human-generated codes, outperforming both single-LLM baselines and perspective-free self-refinement. The three perspectives exhibit distinct trade-off profiles, suggesting that analytical perspective is a meaningful and controllable variable in LLM-assisted QDA. Our findings demonstrate that explicitly modeling analytical perspectives through multi-agent collaboration is a promising direction for improving the credibility of LLM-assisted qualitative analysis.

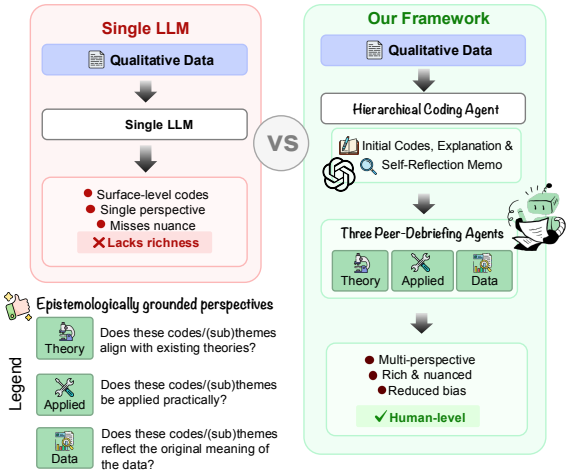

Figure 1: Motivation: A single LLM produces surface-level codes from one perspective. Our framework introduces peer-debriefing agents that perform peer debriefing, yielding richer, multi-perspective coding results.

1 Introduction

Large language models are increasingly used to assist qualitative data analysis (QDA), from generating initial codes to synthesizing themes (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 (Gao et al., 2024, 2023; Richards and Hemphill, 2018). 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: colleagues with different backgrounds review and question the coding from their respective standpoints. Peers are expected to understand the context but remain uninvolved as stakeholders, serving as "disinterested peers" (Hail et al., 2011). This process helps ensure that the analysis captures rich, nuanced information while reducing individual bias.

In this work, we ask: **what if we incorporate the peer debriefing mechanism into an LLM-powered QDA analysis process?** Can this increase the richness of coding results? We propose a multi-agent framework that simulates peer debriefing for LLM-assisted QDA. The framework introduces *peer-debriefing agents* that refine cod-

ing results from three distinct analytical stances within the coding process, mirroring the role of peer debriefers commonly found in QDA (Hail et al., 2011). Specifically, we investigate two research questions (RQ):

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

We evaluate this framework by comparing coding results refined by peer-debriefing agents against those from a single LLM and from human coders. Our results show that perspective-based refinement consistently improves match rate with human codes across all models and datasets (**RQ1**). The three perspectives exhibit distinct trade-off profiles: the Data-Driven perspective maximizes recall, the Theory-Driven and Applied perspectives maximize alignment (i.e., match rate) with human coding, while baseline coding often preserves the greatest code diversity (**RQ2**). These findings suggest that analytical perspective is a meaningful and controllable variable in LLM-assisted QDA, and that perspective selection should be guided by the analytical goal.

Our contributions are:

- We propose a multi-agent framework that operationalizes *peer debriefing* for LLM-assisted QDA, introducing peer-debriefing agents that refine coding from three epistemologically grounded analytical stances (Theory-Driven, Data-Driven, and Applied).
- We evaluate the framework across three datasets in two domains, three LLMs, and three hierarchical levels of thematic analysis (code, sub-theme, and theme), demonstrating its generalizability across domains, models, and abstraction levels.
- We show that different analytical perspectives produce systematic trade-offs in coding outcomes that interact with abstraction level, suggesting that perspective selection is a meaningful and controllable design choice in LLM-assisted QDA.

2 Related Work

2.1 LLM-assisted QDA

LLMs have been increasingly applied to assist qualitative data analysis across NLP and HCI research, from interactive coding tools (Rietz and Maedche, 2021; Gebreegziabher et al., 2023) to LLM-powered theme and code generation (Xiao et al., 2023; Gao et al., 2024, 2023; Bano et al., 2024; Hamilton et al., 2023). However, a recurring finding is that LLM-generated codes often lack alignment with human interpretations. Recent work (Tai et al., 2024) that used GPT-3.5 for thematic analysis showed that LLM coding can both miss and expand upon human codes, highlighting the need for multi-perspective review similar to that of human qualitative coding teams. This observation motivates our work.

2.2 Peer Debriefing with Multiple Perspectives

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. 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 disciplinary 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. Our framework operationalizes this practice by introducing Peer-Debriefing Agents that refine 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 System

Recent advances in LLM-based multi-agent systems have demonstrated the value of orchestrating multiple model instances for complex tasks. Foundational frameworks such as AutoGen (Wu et al., 2023) and MetaGPT (Hong et al., 2024) enable structured multi-agent collaboration, while debate-style paradigms like ChatEval (Chan et al., 2024) use multi-agent argumentation for evaluation. Thematic-LM (Qiao et al., 2025) leveraged multi-agent systems for theme generation, though only for specific topics (i.e., Climate Change). In con-

trast, our framework takes a social science-inspired approach to multi-agent design by explicitly simulating the peer debriefing process in qualitative research. Moreover, the perspectives assigned to our peer-debriefing agents are not domain-specific, making them broadly applicable across diverse datasets.

3 Multi-Agent Framework

Rather than implicitly embedding multi-perspective reasoning within a single model, our framework explicitly treats analytical perspective as a system component. This design enables LLMs to perform peer debriefing with different and fresh mindsets, instead of merely confirming their own reasoning through self-reflection. We first describe the overall process (Section 3.1), then detail the two agent types (Section 3.2) and the analytical perspectives that drive them (Section 3.3).

3.1 Framework Overview

Our framework proceeds through three hierarchical stages: code generation, sub-theme abstraction, and theme synthesis (Figure 2). At each stage, a Hierarchical Coding Agent performs the corresponding coding task and produces a structured result with explanations and self-reflection memo. Three Peer-Debriefing Agents then independently re-examine the result from distinct analytical perspectives and refine the structure using five operations (keep, rename, reassign, merge, split). Researchers review the revisions from all Peer-Debriefing Agents and select the output to carry forward, mirroring the analyst's role in peer debriefing (Lincoln, 1985). Once a stage is finalized, its content remains fixed in subsequent stages. By the completion of the theme stage, the entire hierarchical structure has undergone perspective-based review and revision at every level, forming the final analytical outcome.

Since the analytical perspectives are domain-general (Section 3.3), the entire framework is readily transferable to new qualitative analysis tasks without prompt redesign; the only task-specific inputs are the research question and the interview data. The following two sections describe the agents that execute this process and the perspectives that guide them.

3.2 Agent Design

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

Hierarchical Coding Agent The Hierarchical Coding Agent performs three sequential stages of abstraction over the qualitative data. In the *code generation* stage, it segments interview text into analytical units and assigns initial codes. In the *sub-theme abstraction* stage, semantically related codes are grouped into sub-themes. In the *theme synthesis* stage, sub-themes are further aggregated into overarching themes. The same agent architecture is used across stages with stage-specific instructions, differing only in input granularity and abstraction level. At each stage, the agent also produces a self-critique and explanation alongside the structural output, providing evidence and reasoning that serve as reference context for the Peer-Debriefing Agents during subsequent refinement.

Peer-Debriefing Agent The Peer-Debriefing Agent explicitly models different analytical perspectives. Given an existing coding structure and the reflection memo from the Hierarchical Coding Agent, it re-examines the relationships among codes, sub-themes, and themes from a specific perspective and performs structural adjustments through 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.
- **Merge**: Combines multiple codes into a single code.
- **Split**: Divides one code into multiple codes.

These operations do not modify the original textual content, preserving structural stability. Rather than repeated generation, the agent simulates a "disinterested peer" who reinterprets the same structure from a specific analytical perspective, making perspective differences explicit and comparable. Moreover, we intentionally exclude human-generated codes as few-shot examples from the agent prompts and instead provide only general analytical instructions derived from established qualitative research methodologies, so that the agents are not constrained by any individual researcher's analytical style. The full prompt templates for all agents are provided in Appendix C.

A critical design question remains: what perspectives should these agents adopt? We address

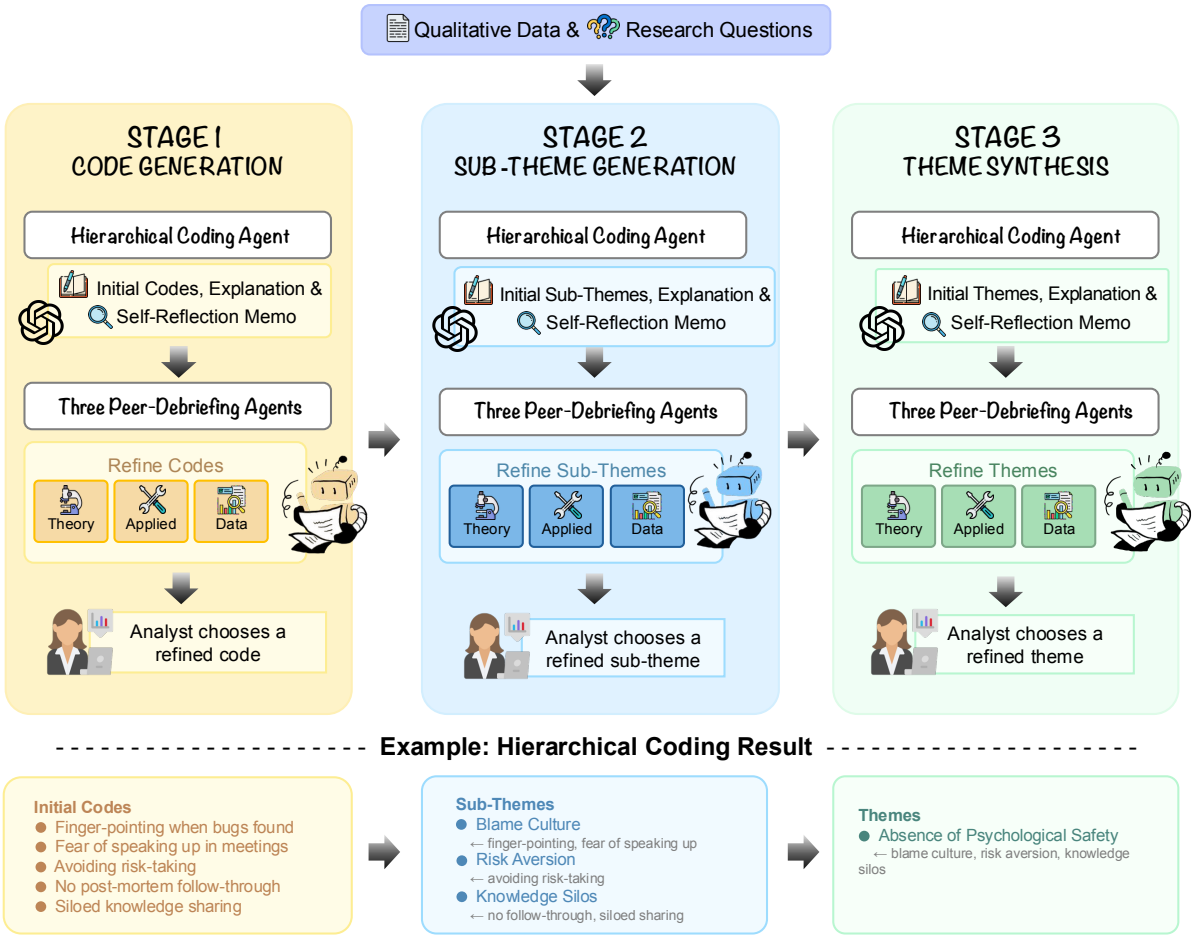

Figure 2: The framework consists of three hierarchical stages: code generation, sub-theme generation, and theme synthesis. At each stage, the Hierarchical Coding Agent produces structured results with explanations and self-critiques. Peer-Debriefing Agents then re-examine and refine these results from different analytical perspectives. Researchers select the preferred refinement at each stage, which remains fixed and is carried forward to form the final hierarchical analytical structure.

this next by grounding the perspective definitions in established epistemological traditions.

3.3 Perspective Modeling

We build three perspectives grounded in established epistemological traditions in qualitative research. The Theoretical perspective draws on constructivist epistemology, prioritizing conceptual coherence (Lincoln, 1985). The Data-Driven perspective reflects an empiricist stance rooted in grounded theory (Glaser and Strauss, 2017). The Applied perspective is informed by pragmatist epistemology (Vo, 2012), which evaluates analytical categories by their practical utility and interpretability. Together, these stances mirror the diversity of analytical lenses commonly found in peer debriefing.

A key design goal is domain generality: the analytical perspectives are grounded in widely recognized qualitative research methodologies (Saldaña, 2021; Charmaz, 2015) rather than tailored to any specific dataset or domain. We define three representative coding perspectives below. Each perspective applies a different *evaluation criterion* when determining what constitutes a good code, and we illustrate them using the same interview excerpt:

Theory-Driven Perspective The theory-driven perspective (Braun and Clarke, 2006) draws on *external knowledge*, such as theoretical frameworks and disciplinary constructs. The agent prioritizes conceptual coherence and alignment with existing literature, retaining the code with stronger theoretical grounding when alternatives conflict. This reflects researchers who interpret 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 generate a code such as `Absence`

of Psychological Safety, linking the observation to the construct of psychological safety (Edmondson, 1999) rather than remaining close to the participant's wording.

Data-Driven Perspective The data-driven perspective (Braun and Clarke, 2006) draws on *local context*: the language and meaning expressed in the data itself, without imposing external frameworks. Codes emerge inductively from what participants say, and when alternatives conflict, the agent retains the one that better preserves the participant's wording. A related concept is "in-vivo coding", in which researchers use participants' original words or phrases as codes (Saldaña, 2021). This reflects researchers practicing inductive qualitative analysis who remain faithful to the data. For the same interview excerpt, the agent may generate a code such as Finger-Pointing When Bugs Are Found, preserving the phrasing without abstracting it into a theoretical label.

Applied Perspective The applied perspective emphasizes *practical utility*: whether codes can directly inform decisions or actions. The agent prioritizes usefulness over theoretical elegance or strict data fidelity, retaining the code that more clearly suggests what a practitioner should do, with actionable implications. This reflects practitioners conducting qualitative analysis for decision-making. This is similar to "Process code" described in (Saldaña, 2021), which prioritizes action and activities in the data. For the same excerpt, the agent may generate a code such as Need for Blame-Free Post-Mortem Process, suggesting an organizational intervention that 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 spanning 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) investigating how Scrum contributes to software quality. Two research questions from the original study are evaluated separately: ScrumRQ1 and ScrumRQ2. The dataset provides per-interview coding, each data chunk is annotated with one or more codes, and a consolidated codebook with code definitions and frequency counts. The **UCSB** dataset[2] (Curty et al., 2021) comprises 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, enabling evaluation at the thematic level. The datasets are summarized in Table 1. By combining both, we evaluate coding quality at the per-interview level (Scrum) and thematic coherence at the subtheme and theme level (UCSB), while codebook-level metrics provide cross-dataset comparability.

To our knowledge, this selection of datasets represents 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, covering all three levels of thematic analysis (code, sub-theme, and theme) across three LLM backbones, with per-interview code generation and comparison with ground truth on the Scrum dataset. Moreover, all three datasets include published human-generated ground truth codes, enabling direct quantitative comparison at each analytical level and supporting the reproducibility of our results.

Perspective isolation Our automated evaluation measures each perspective independently to isolate its individual effect on coding quality. In practice, researchers can mix perspectives across stages (e.g., selecting a theory-driven result at the code level and an applied result at the sub-theme level). Evaluating such mixed-perspective strategies is left for future work.

Base models To evaluate the performance of our multi-agent framework, we instantiate the architecture using three LLMs (GPT-5 (Singh et al., 2025), DeepSeek-V3.2 (Liu et al., 2025), and Qwen3.5-Plus (Team, 2026)) as the backbone for the agents. This approach enables a controlled comparative analysis of how different base models influence the framework's efficacy in qualitative data analysis.

[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.

4.2 Metrics Definition

To evaluate the quality of generated codes, we use a semantic similarity-based matching strategy to compare the generated codes ($Code_{gen}$) with human-annotated codes ($Code_{gt}$), where "gt" represents the ground truth. This strategy allows us to define metrics that capture both the coverage of human codes and the semantic alignment of generated codes with human annotations. We define four metrics derived from the literature (Gao et al., 2023; Gebreegziabher et al., 2023): Recall, Match Rate, Code Diversity, and Text Utilization Rate, as described below.

In particular, Recall and Match Rate rely on similarity matching. For each generated code $c_i^{gen} \in Code_{gen}$, we compute its cosine similarity with all human-annotated codes $c_j^{gt} \in Code_{gt}$, yielding a similarity score $s_{i,j}$. The text embedding is encoded using all-MiniLM-L6-v2 (Wang et al., 2020) model. We then normalize this score to the range $[0, 1]$ using $\tilde{s}_{i,j} = (s_{i,j} + 1)/2$. The best match for c_i^{gen} is identified as $j^* = \arg\max_j \tilde{s}_{i,j}$. If the maximum similarity $\tilde{s}_{i,j^*}$ exceeds a predefined threshold τ (set to 0.7), we consider c_i^{gen} to be matched with $c_{j^*}^{gt}$; otherwise, it is deemed unmatched ($M(c_i^{gen}) = \varnothing$). We set $\tau = 0.7$ to balance two considerations: it is permissive enough to accommodate paraphrastic variation between generated and ground truth codes, while strict enough to exclude pairs that are topically related but conceptually distinct. We verified this threshold by manually inspecting code pairs near the boundary (0.65–0.75) and confirmed that it reliably separates true semantic matches from partial overlaps.

Note that this matching strategy yields a one-to-many mapping: each generated code can match at most one human code, but a single human code may correspond to multiple generated codes.

Recall Recall measures the proportion of human-annotated codes that find a semantic match among the generated codes: Recall $= |C_{gt}^{match}|/|Code_{gt}|$, where C_{gt}^{match} is the set of $Code_{gt}$ matched by at least one $Code_{gen}$.

Match Rate Akin to precision, Match Rate measures the proportion of generated codes that find a semantic match among the human-annotated codes: Match Rate $= |\{c_i^{gen} \mid M(c_i^{gen}) \neq \varnothing\}|/|Code_{gen}|$. 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 Code Diversity measures the proportion of semantically unique codes among generated codes. We iteratively remove one code from any pair where cosine$(c_i^{gen}, c_k^{gen}) \geq \tau$, yielding a reduced set $Code_{gen}^*$. Code Diversity is then $|Code_{gen}^*|/|Code_{gen}|$. Higher values indicate greater semantic 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 coding outcomes of the baseline (no perspective) with those of the three perspectives (Theory-Driven, Data-Driven, and Applied) across all models and datasets. We analyze the results using the four evaluation metrics defined above. All reported means are averaged over 39 interviews for the Scrum RQ1 and Scrum RQ2 datasets, and 10 interviews for the UCSB dataset.

4.3.1 RQ1: Does peer-debriefing agent improve coding quality?

Our results show that for each model, dataset combination, the best-performing perspective substantially improves Match Rate over the baseline (Figure 3). We show the best-performing perspective for each configuration as an illustration, full 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 for a given context.

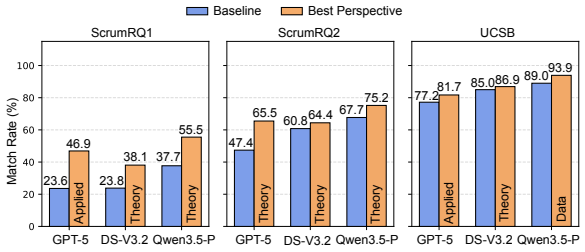

Figure 3: Match Rate comparison between baseline (no perspective) and the best-performing perspective for each model–dataset combination. The best-performing perspective for each configuration is identified from the full results reported in Table 3. DS-V3.2 → DeepSeek-V3.2; Qwen3.5-P → Qwen3.5-Plus.

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 consistently 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 fine-grained 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 data-specific 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 competitive 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 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 Qualitative Case Study

To illustrate how perspective-based refinement operates in practice, we present a concrete example from Scrum RQ1, Interview 0 (Table 9 in Appendix). Starting from the same 10 baseline codes, all three perspectives independently identify the `Retrospectives` code as problematic, it conflates participant statements with researcher utterances, and split it to separate these voices. This convergence demonstrates that perspective-based review can reliably surface coding issues that a single-pass analysis overlooks.

The perspectives diverge in consolidation and renaming. The Theory perspective merges testing-related codes into an abstract construct; the Applied perspective additionally consolidates testing-related codes using practitioner-oriented labels. The baseline code `Process vs product` is renamed differently by each perspective, mirroring how human researchers with different backgrounds would label the same phenomenon, precisely the multi-perspective review that peer debriefing aims to achieve.

4.5 Preliminary Ablation Study

To disentangle the effect of analytical perspective from the effect of 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 all 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 iterative self-correction; the explicit modeling of analytical perspectives provides additional value beyond what an undirected refinement pass achieves.

5 Discussion

To better understand how different analytical perspectives influence the coding process, we analyze the structural modifications performed by the Peer-debriefing Agent. As described in Section 3, the agent can revise the initial coding structure through five operations: **Keep**, **Rename**, **Reassign**, **Merge**, and **Split**. Since Keep preserves the existing structure without modification, we focus our analysis on the four revision operations. Figure 4 (Appendix) shows the average frequency of these operations across models, perspectives, and datasets.

Rename Among all operations, Renaming is the most frequent behavior. Across most model dataset configurations, the number of Rename operations exceeds those of Merge and Split. For example, in ScrumRQ1, GPT-5 (Applied) performs 4.6 Rename operations on average, compared with 2.3 Merge and 1.0 Split. This pattern suggests that, instead of substantially altering the code structure, the agent tends to refine how existing concepts are expressed. Such behavior resembles common practices in qualitative coding, where researchers iteratively revise code labels to better capture the meaning of the data.

Reassign Unlike Merge and Split, which restructure the coding scheme itself, Reassign captures independent data movement: relocating text chunks between codes, or moving a code to a different sub-theme or theme. This metric excludes the implicit chunk redistribution that occurs as a byproduct of Merge or Split. The frequency of independent reassignment varies substantially across models: GPT-5 uses it most actively (1.20–2.33 per interview), suggesting a fine-grained optimization strategy, while DeepSeek-V3.2 (0.67–1.02) and Qwen3.5-Plus (0.00–0.60) rely more on Merge.

Merge Compared with Rename, Merge operations occur less frequently, but show a noticeable association with theory-oriented perspectives. For instance, in ScrumRQ1, DeepSeek-V3.2 (Theory) performs 1.9 Merge operations on average, exceeding its counts under other perspectives. Merge reflects a process of conceptual abstraction, in which semantically related codes are integrated into higher-level categories. This behavior aligns with the analytical goals of theory-driven analysis, where empirical observations are often consolidated into broader conceptual constructs.

Split Split is the least frequent operation across most configurations, typically occurring fewer than once per run. However, certain perspective-dataset combinations exhibit higher frequencies. For example, in the UCSB dataset, GPT-5 (Data) performs 2.2 Split operations on average. Splitting reflects a tendency toward the fine-grained differentiation of codes. When an initial code encompasses multiple distinct phenomena, the data-driven perspective encourages separating them into more specific subcodes.

6 Conclusion

We propose a multi-agent framework for LLM-assisted qualitative analysis that integrates Peer-Debriefing Agents mimicking peer debriefing. Experiments show that perspective-based refinement produces richer, more nuanced, and less biased coding than single-LLM outputs, with distinct perspectives offering complementary strengths. Our results highlight that explicitly modeling analytical perspectives is a simple yet effective way to enhance LLM-assisted coding quality.

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.

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.

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.

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.

Barney Glaser and Anselm Strauss. 2017. *Discovery of grounded theory: Strategies for qualitative research*. Routledge.

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*.

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.

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.

Linh-Chi Vo. 2012. Pragmatist perspective on knowledge and knowledge management in organizations. *International Business Research*, 5(9):78.

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.

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.

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 Agents. 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 Agents.

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.

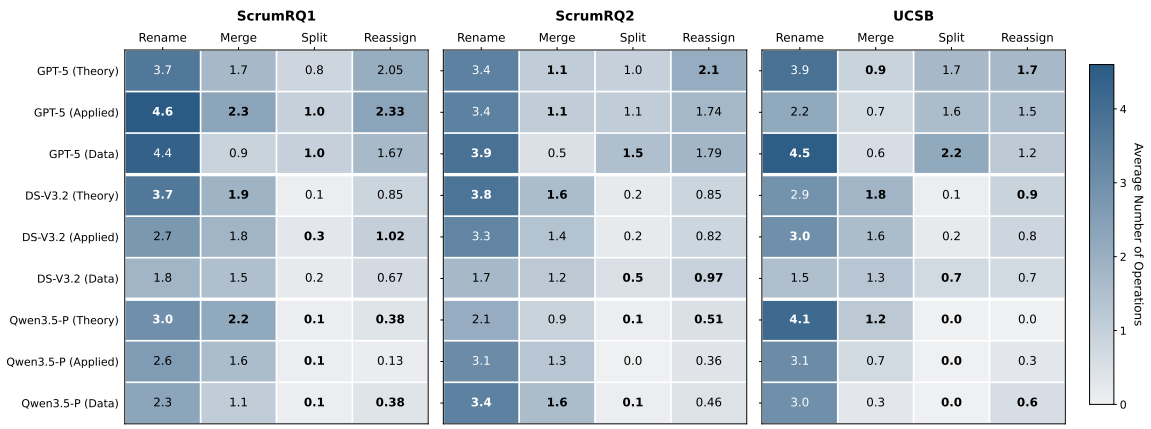

Figure 4: Heatmap of the average number of Rename, Merge, Split, and Reassign operations performed per interview. The columns represent the operation types, while the rows show the performance of different Peer-Debriefing Agents (Theory, Applied, Data) powered by GPT-5, DeepSeek-V3.2, and Qwen3.5-Plus 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 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 9: 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.

C 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.
```