# Development and Benchmarking of a Blended Human-AI Qualitative Research Assistant

**Joseph Matveyenko, James Liu, John David Parsons, Ryan A. Brown, Alina I. Palimaru, Vipul Gupta, Prateek Puri**

RAND Corporation
{jmatveyenko, ryan_brown, alina_palimaru, vipul, ppuri}@rand.org

## Abstract

Qualitative research emphasizes constructing meaning through iterative engagement with textual data. Traditionally, this human-driven process requires navigating coder fatigue and interpretive drift, thus posing challenges when scaling analysis to larger, more complex datasets. Computational approaches to augment qualitative research have been met with skepticism, partly due to their inability to replicate the nuance, context-awareness, and sophistication of human analysis. LLMs, however, present new opportunities to automate aspects of qualitative analysis while upholding rigor and research quality. In this work, we present and benchmark Muse, an interactive qualitative research system that allows researchers to identify themes and annotate datasets, achieving an inter-rater reliability between Muse and humans of Cohen's $\kappa = 0.7$ for well-specified codes.

## 1 Introduction

Qualitative research has long been characterized by its emphasis on interpretive depth, contextual understanding, and the careful construction of meaning from textual data (Denzin and Lincoln, 2011; Flick, 2014). The process of qualitative coding requires substantial time investment, with researchers often spending weeks or months in iterative cycles of reading, coding, discussing, and refining their analytical frameworks.

Recent advances in LLMs have shown promise for automating aspects of qualitative analysis (Bhaduri et al., 2024; Bennis and Mouaffaq, 2025; Laato et al., 2025; Morgan, 2023). Prior work has demonstrated that LLMs can extract high-level concepts from unstructured text and generate interpretable topic models that surpass clustering-based approaches in certain research contexts (Lam et al., 2024).

However, a critical gap remains: these novel systems need to be scalable and adaptable to the rich diversity of domains in qualitative research. One component of this optimization process is rigorous benchmarking against real-world, human-generated datasets to assess their performance, benefits, limitations, and trustworthiness. Another is obtaining viable latency and performance tradeoffs during deployment.

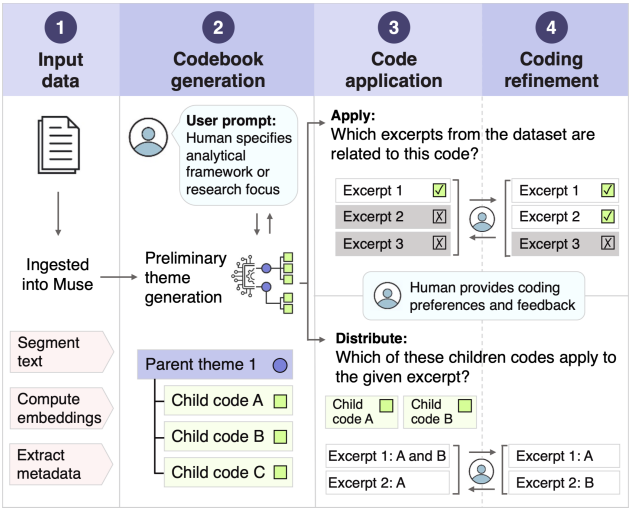


Figure 1: An illustration of the Muse system for AI-assisted qualitative research. Documents are ingested into the platform, at which point they are segmented, embedded, and cataloged. Users may prompt the system to identify themes and generate a codebook. Similarly, users can apply a code to the entire dataset or distribute a set of excerpts among a series of codes.

In this work, we present the Muse system (Figure 1), a qualitative research assistant designed through extensive feedback cycles with more than a hundred qualitative researchers. Muse is a production system optimized for real-world deployment and currently has more than 600 projects and 400 users. We demonstrate the performance of Muse through benchmarking and also discuss system optimization for latency and accuracy. We explore how Muse can address three key challenges: (1) consistent application of established coding

schemes across diverse datasets, (2) automated codebook generation that captures the nuances of qualitative frameworks, and (3) rigorous evaluation of AI-assisted coding quality using established inter-rater reliability (IRR) metrics. Through our analysis, we present the following contributions:

*Curated multi-domain evaluation dataset*: We compiled eleven publicly available qualitative research datasets spanning interviews, social media posts, survey responses, and domain-specific corpora. This collection, which we've made available on GitHub, addresses critical data sharing limitations in qualitative research and provides a foundation for standardized benchmarking.[1]

*Systematic hyperparameter optimization necessary for real-world deployment of qualitative AI systems*: Through comprehensive evaluation across multiple datasets, we identify optimal configurations for LLM-based coding systems, tuning over parameters such as prompt design, model selection, and dataset batching. We explore the performance and cost/latency trade-offs associated with different system settings, ultimately identifying parameters that judiciously balance both to go from prototypes to something we could deploy across our organization.

*Benchmarking of performance in qualitative research settings*: We demonstrate how AI assistance can achieve research-grade coding reliability (Cohen's $\kappa = 0.7$) for well-specified codes across a wide range of datasets and domains, comparable to performance demonstrated in domain-specific evaluations (Borse et al., 2025).[2]

## 2 System Design

To ensure our system addresses real-world research needs rather than technical benchmarks alone, we conducted several expert interviews with intended users and conducted a survey. Respondents spanned a range of seniority levels: 48% held senior or full-level positions, 28% were mid-level, 14% associate-level, and 5% were assistant or graduate-level researchers. The sample skewed towards experienced qualitative researchers - 80% reported 5 or more years of experience with qualitative methods, and 58% reported 10 or more years. By discipline, respondents were primarily policy researchers and analysts (41%), followed by behavioral and social scientists (17%), political scientists (4%), and information scientists, survey researchers, operations researchers, and engineers making up the remainder. These touchpoints informed critical design decisions from feature prioritization to interface design to hyperparameter selection. By grounding our technical development in the perspectives of researchers and the unique requirements of each individual project, we aimed to design a system that would generalize across research contexts and enable high levels of user interaction and steerability throughout its functionality.

When asked about high-priority tool features in our survey, researchers expressed a desire for flexible workflows that enable both inductive and deductive analysis as well as greater consistency in code application. Given these perspectives, we focused on the development and evaluation of three main tools within the Muse platform that align with these needs: *generate codebook*, *apply code*, and *distribute code*. Although the Muse platform contains many other features — such as chat-with-your-data functionality, automated code definition updating, and others — in this work, we focus primarily on these three features given their amenability to rigorous evaluation and their relevance to others building LLM-based systems for qualitative research.

While our survey revealed broad consensus on desired features, each researcher brings a unique analytical lens and distinct preferences to how they conduct their research. This posed a core design challenge common to many organizations deploying GenAI solutions: building a system that is both high-performing across diverse research domains and highly steerable to individual researcher taste. To navigate these competing demands, we conducted several experiments optimizing across latency, performance, and steerability. To track progress across our optimization dimensions, we built a monitoring and experimentation pipeline using Postgres, Grafana, and Langfuse to track user activity and system performance.

### 2.1 Codebook Generation

Our approach to automated codebook generation employs a multi-stage process that systematically

[1] https://github.com/jdmatv/muse-eval-dataset

[2] There is no single IRR value that is considered "human-level." IRR can vary significantly depending on factors such as the degree of interpretation required to make judgments about text and the training of coding teams to align on a specific coding task. Many teams will refine their coding scheme until an "acceptable" IRR level is reached. Broadly, the 0.61–0.80 $\kappa$ range has been described as representative of substantial agreement in the qualitative research domain (Cole, 2024).

analyzes qualitative data through progressive abstraction and refinement to enable the specific analysis each user would need. The algorithm first segments the document set, and a line-coding LLM is instructed to identify between $n_{\min}$ and $n_{\max}$ themes within each segment. The user has the ability to provide natural-language instructions to guide the LLM toward discovering concepts aligned with a particular research scope.

The resulting preliminary codebooks are consolidated through a codebook condensation process using a second LLM, merging sets of preliminary codebooks until a unified coding framework emerges that represents themes present across all data segments.

The system generates parent themes by prompting another LLM to (1) examine the relationships between the identified child codes in the previous set, (2) identify a set of parent concepts that encapsulate these codes, and (3) assign each child code to one of the identified parent codes.

To validate the codebook, we create a composite embedding for each code by averaging the embeddings of its definition and all associated quotations, and use Maximum Marginal Relevance scoring to identify the most relevant segments from the dataset. An LLM evaluates whether each code is sufficiently supported by these representative excerpts.

Finally, the algorithm performs hierarchical refinement by prompting another LLM to merge semantically redundant codes and split overly broad codes into more specific subcodes. This computational approach mirrors the process of traditional line-by-line coding while using natural language processing to achieve consistency and scalability across large qualitative datasets.

### 2.2 Code Application

Leveraging the generated codebook, Muse applies codes to documents in two ways: *apply code*, a tool which assigns a binary label or score to an excerpt given its relevance to a given code, and *distribute code*, which assigns one or multiple codes within a set of codes to an excerpt based on relevance.

We incorporate user feedback in our coding feature in two core ways. First, users are encouraged to modify their prompt definition to reflect their coding requirements, with additional features integrated that allow for rapid iteration and testing on a sample of representative excerpts. Second, users can provide positive or negative feedback about LLM annotations through dynamic user interface icons, creating examples that will be fed to the LLM as few-shot context in subsequent coding runs.

Single-code application works by splitting documents into segments of approximately 80 words and prompting an LLM to determine whether each is relevant to a given code, with excerpts classified in batches to optimize both throughput and latency. The LLM is provided with the code name, definition, inclusion and exclusion criteria, and few-shot examples. Multi-code application (*distribute code*) has a similar workflow; however, it differs in that it includes code information (definition, criteria, and few-shot examples) for every code in a chosen subset, often offering lower latencies than *apply code* for this task. The prompts for both methods are displayed in Appendix B.

## 3 Evaluation

### 3.1 Datasets

We used eleven publicly available human-coded datasets to assess the performance of Muse in generating codebooks and annotating qualitative data. These datasets cover a diverse range of both data formats (e.g., social media and interviews) as well as domain areas (e.g., disordered eating, political tension, and sustainability).[3]

### 3.2 Generate Codebook Evaluation

We benchmarked codebook generation against the arXiv Categories dataset (Schopf et al., 2023), which provides an ideal evaluation context given its hierarchical codebook structure, large scale, and vetted quality. While not strictly qualitative data, arXiv abstracts contain the conceptual complexity and domain-specific language characteristic of many qualitative datasets. We randomly selected three ground-truth parent codes from the arXiv corpus and drew 1,000 abstracts from documents labeled with at least one of those parents. We included every child code associated with those documents, yielding a sample-specific codebook containing the selected parents and all sampled descendants. We repeated this procedure with different parent-code triplets, producing evaluation codebooks ranging from 22 to 52 codes - a range aligned with the typical codebook size reported in our internal survey.

[3] Appendix A lists the evaluation datasets and describes our search strategy for identifying and classifying datasets.

| Batch Size | Few-Shot Examples | Scoring Type | CoT | Cohen's $\kappa$ [95% CI] |
|---|---|---|---|---|
| 5 | 4 | Discretized | Yes | **0.545 [0.427, 0.662]** |
| 5 | 2 | Discretized | Yes | 0.532 [0.425, 0.639] |
| 10 | 2 | Discretized | Yes | 0.524 [0.408, 0.640] |
| 10 | 4 | Discretized | Yes | 0.521 [0.403, 0.640] |
| 5 | 2 | Binary | Yes | 0.518 [0.407, 0.628] |
| 10 | 2 | Discretized | No | 0.517 [0.406, 0.629] |
| 5 | 4 | Binary | Yes | 0.513 [0.398, 0.628] |
| 10 | 2 | Binary | Yes | 0.510 [0.399, 0.620] |

Table 1: Hyperparameter optimization grid search (sorted by Cohen's $\kappa$, top 8 shown, highest bolded) for code application with GPT-4o. Batch size is the number of excerpts included in a single prompt (5 or 10). Number of few-shot examples tested was 0, 2, and 4. Discretized scoring is a 1-10 score generated by the LLM, whereas binary is a "yes" or "no" response. CoT prompting involves the LLM providing a short explanation of its output prior to producing it. Full table displayed in Appendix D.2.

Given the unique constraints of our project (codebook generation independent from the coding step), and that codebook similarity evaluation is an open problem in the literature, we developed a customized composite similarity metric to compare semantic similarity (label embeddings) and structural similarity (hierarchical depth and organization) between human- and LLM-generated codebooks.[4] We systematically varied the hyperparameters (including language model and segment length) and found that the algorithm is largely insensitive to these hyperparameters and shows only minimal accuracy–latency trade-offs. Our LLM-based method achieved comparable performance to the optimized BERTopic baseline (~0.90-0.91 similarity), demonstrating that our approach matches near-state-of-the-art clustering methods in accuracy, while offering additional steerability.

## 3.3 Code Application Evaluation

### 3.3.1 IRR as a Performance Metric

We assessed the performance of LLM annotations by measuring IRR with human coders across our evaluation dataset. While IRR is traditionally used to measure coding alignment between two human raters, here we use IRR as a metric for alignment between a human operator and an LLM annotator – with the degree of alignment reflecting how well the latter captures the opinions and preferences of the former (Borse et al., 2025).

The primary measure of IRR that we considered is Cohen's $\kappa$ (Cohen, 1960), a standard metric in qualitative research that inherently adjusts for chance agreement between two coders and ranges in value from -1 (full disagreement) to 1 (full agreement). This is more holistic than percent agreement, or statistics like accuracy or precision, because it factors in the frequency of positives and negatives, which is often extremely unbalanced in qualitative datasets.

### 3.3.2 Hyperparameter Selection

LLM responses can vary significantly with seemingly trivial input and output format changes (Khan et al., 2025). Given this sensitivity, we systematically explored hyperparameter configurations including two LLM-as-a-judge scoring output types (binary yes/no and discretized 1-10), chain-of-thought prompting, batch size within a prompt, and number of few-shot examples (Li et al., 2024; Nye et al., 2021; Wei et al., 2022).[5] We optimized prompt design and output structure by testing combinations of hyperparameters with a single LLM (GPT-4o) as shown in Table 1. We found that CoT prompting, providing true positive examples, and discretized 1-10 scoring produced higher IRR than their alternatives.

One observation we made as we reviewed the best and worst-performing datasets was that there is an inherent level of sensitivity with which qualitative researchers approach a target concept. Even

[4] Existing work approximates similarity through cosine similarities between code description embeddings (De Paoli and Mathis, 2025; Wang et al., 2025), which captures label semantics but ignores hierarchy. More general NLP topic modeling metrics often cite NMI, purity, coherence scores, etc., which require document-to-cluster assignments (Abdelrazek et al., 2023). Our metric, described in Appendix C, combines embedding-based semantic similarity with structural hierarchy matching.

[5] Appendix D.1 describes the significance and implementation of these hyperparameters.

| Model Name | Untuned Cohen's $\kappa$ [95% CI] | Code-tuned Cohen's $\kappa$ [95% CI] |
|---|---|---|
| gpt-4.1-2025-04-14-global | 0.515 [0.426, 0.604] | **0.593 [0.512, 0.674]** |
| o3-2025-04-16-global | **0.520 [0.423, 0.617]** | 0.590 [0.508, 0.672] |
| gpt-5-2025-08-07-us (default) | 0.515 [0.422, 0.608] | 0.578 [0.498, 0.658] |
| grok-3-v1 | 0.505 [0.425, 0.585] | 0.572 [0.496, 0.648] |
| grok-3-mini-v1 | 0.492 [0.402, 0.582] | 0.553 [0.470, 0.636] |
| o4-mini-2025-04-16-global | 0.492 [0.398, 0.586] | 0.546 [0.459, 0.633] |
| gpt-oss-120b (high reasoning) | 0.486 [0.386, 0.586] | 0.543 [0.450, 0.636] |
| Microsoft MAI-DS-R1 | 0.499 [0.413, 0.585] | 0.539 [0.456, 0.622] |

Table 2: Inter-rater reliability by LLM (sorted by code-tuned $\kappa$, top 8 shown, highest bolded). Code-tuned means setting a separate confidence score threshold for each code to align with the level of sensitivity in the human-coded data. Other models tested included GPT-4o, GPT-4.1 mini, DeepSeek V3, DeepSeek R1, Nemotron Ultra 253B, and Nemotron Super 49B. Full table displayed in Appendix D.3.

if a code is well-specified and contains inclusion and exclusion criteria, there is still often a level of human judgment as to whether it meets a certain threshold of salience that they desire for analysis.

Binary scoring outputs prevent the end user from tuning this parameter after code application is complete. On the other hand, discretized outputs provide the user with the ability to tune LLM confidence thresholds as desired. Another approach to convert binary scoring into a higher-granularity score is through normalizing output log probabilities, a common post-processing approach for LLM-as-a-Judge systems (Baysan et al., 2025; Thakur et al., 2025). We assessed the performance of using the log probabilities of "Yes" and "No" tokens in responses, which we mapped onto a 1-10 scale using a temperature-controlled softmax function (Gu et al., 2025), and found that regular discretized 1-10 scoring had higher IRR than log probabilities.

Additionally, we simulated that a researcher has tuned their confidence score (1-10) threshold for each code to align with their preferences (code-tuned) and found that this approach achieves greater alignment than assigning a single confidence score threshold for all codes in the system.[6]

### 3.3.3 Annotation Benchmarking Results

Once we arrived at optimal hyperparameters for GPT-4o (discretized scoring, CoT prompting, batch size = 5, and few-shot examples = 4), we tested performance across a range of closed-source, open-source, reasoning, and non-reasoning LLMs. It is of value to assess open-source model performance, as many real-world qualitative research environments are not compatible with transmitting protected data to API providers.

While we did not optimize prompts for each individual model, benchmarking IRR using one set of parameters provides a consistent and reproducible test arena for model comparisons. We tested a range of popular LLMs that were accessible to the team through Azure AI Foundry and Red Hat for cloud-based hosting with vLLM (Kwon et al., 2023) but recognize that omissions remain that future work could evaluate (e.g., Gemini, Claude, and Qwen). Nonetheless, the results shown in Table 2 demonstrate a similar range of performance across state-of-the-art frontier models.

Of the 15 models we tested, 8 outperformed GPT-4o. GPT-4.1 exhibited the highest level of performance - demonstrating parity with newer foundation models like o3 and GPT-5 - although this may be a relic of the fact that prompt optimization for GPT-4o through CoT maps most directly onto this model.

Using an unweighted average of the discretized scores of three of the top frontier models (GPT-4.1, o3, and Grok 3) achieves an untuned $\kappa$ of 0.55, code-tuned $\kappa$ of 0.61, and for codes with a clear definition, a code-tuned $\kappa$ of 0.7. In Appendix D.4, we compare these results to other supervised ML (sML) approaches on the same well-specified codes (EmbeddingGemma, SBERT, and TF-IDF). Both GPT-4.1 and the ensemble of three models outperform all sML methods tested, even when using 512 examples to train sML models compared to 4 few-shot examples and definitions for the LLM.

[6]Tuning this threshold is the intended workflow within our research system, so that a user can easily adjust the threshold and evaluate alignment to their coding preferences. The 10 evaluation datasets were created by groups with various coding preferences and sensitivities to concept salience. Tuning the confidence threshold allows learning of these preferences.

| Method | Batch Size | Avg. Seconds per Excerpt | Input Tokens per 100 Excerpt Tokens | Output Tokens per 100 Excerpt Tokens | Cohen's $\kappa$ |
|---|---|---|---|---|---|
| distribute | 20 | **0.11** | **92.6** | 4.1 | 0.57 |
| apply | 20 | 0.17 | 209.5 | **3.7** | **0.65** |
| distribute | 10 | 0.19 | 141.3 | 6.9 | 0.57 |
| apply | 10 | 0.33 | 287.0 | 5.5 | 0.64 |
| distribute | 5 | 0.38 | 241.3 | 11.6 | 0.60 |
| apply | 5 | 0.58 | 439.6 | 8.9 | 0.64 |

Table 3: Latency, token usage, and performance by annotation method and batch size per prompt. The *distribute* annotation method rates excerpts for applicability to all 6 codes at once. The *apply* method rates excerpts for applicability to each code separately, which requires more prompts, time, and tokens per excerpt. Average inter-rater reliability reported for the 6 most frequent codes from two well-specified datasets using GPT-4.1 and constant confidence threshold. Bold values indicate optimal performance within each metric column.

#### 3.3.4 Evaluating Multi-Code Application

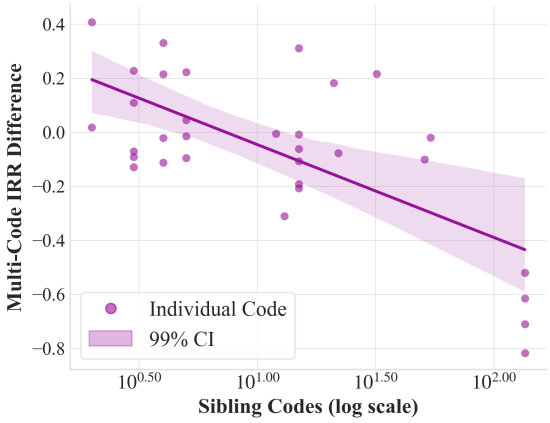


Figure 2: Difference in IRR (Cohen's $\kappa$) between multi-code application and single-code application with respect to the number of sibling codes. A greater number of sibling codes (i.e., options to choose from for multi-code application) is associated with a decrease in IRR compared to single-code application.

To assess the performance of the *distribute code* tool, which applies one or multiple codes within a set of codes to an excerpt, we compared its IRR with *apply code*. Figure 2 shows that when the number of target codes to select from is low (less than about 6), *distribute code* outperforms *apply code* and results in lower latency and cost as it requires fewer prompts. Also, when a codebook contains researcher-specified constraints such as requiring at least one code per document or no more than one code per document, the *distribute code* tool is strongly preferred and yields IRR scores that are significantly higher as it provides the LLM with these constraints (Cohen's $\kappa$ greater by an average of 0.18 than *apply code*).

#### 3.3.5 Cost, Latency, and Performance Trade-offs

In deployment, AI/ML systems must balance performance with operational efficiency. To characterize these trade-offs, we evaluated token usage, latency, and performance of the *distribute code* and *apply code* methods across varying batch sizes using the six most frequent codes from the arXiv Categories dataset and six most frequent from *Topic Annotations on Reddit Posts* (Qiu, 2025).

Table 3 shows that there are significant efficiency gains with respect to latency and input tokens from using multi-code application (*distribute code*) as opposed to *apply code*. At each batch size tested, *distribute* takes about 55–65% of the time per excerpt and 45–55% of input tokens as *apply code*. This is primarily due to the additional prompts required by single-code application for each code, and this difference would likely become less pronounced when using fewer sibling codes.

However, across all batch sizes, the IRR using *distribute* is lower than that of *apply*. The IRR difference would also likely be less significant with fewer sibling codes, as Section 3.3.4 shows that IRR can be even higher for *distribute* than *apply* with few sibling codes.

In our deployment of this system, we provide the user the opportunity to choose between the two methods depending on the nature of their codebook. Also, researchers can iteratively test and improve coding performance with random sample IRR validation to identify the optimal parameters for their dataset.

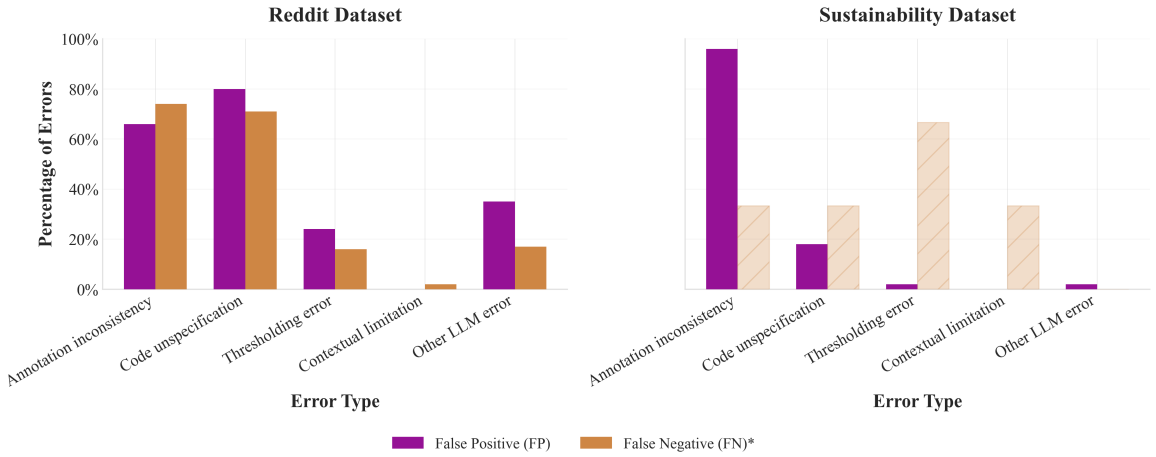


Figure 3: False positives (FPs) and false negatives (FNs) by error type and dataset. The two most common error types for both FPs and FNs across datasets are code under-specification and annotation inconsistencies (deviations from the established coding scheme). *FNs for the Sustainability dataset are made transparent because the dataset only had three false negatives to code, so the percentage of errors is not representative.

## 4 Error Analysis

While Muse applies annotations at an IRR comparable to humans, inevitably, not all annotations match those of qualitative researchers. We conducted an error analysis of inconsistencies between human and Muse codings to categorize failure modes and identify areas for future improvement. To gain a better understanding of the types of errors made by the LLM, our team analyzed a random sample of errors. For tractability, we narrowed our focus to two datasets, *Topic Annotations on Reddit Posts*, which contained label definitions, more document context, and overall clearer organization than many other datasets in our repository, and the *ATLAS.ti Sustainability dataset*, a sample interview-based project (ATLAS.ti, n.d.).[7]

After an initial review by two authors of the discrepancies between the unweighted average score of three LLMs (GPT-4.1, o3, and GPT-oss-120b) with human coders, we iteratively developed a taxonomy of the root causes of observed errors. Two other authors, who are experienced qualitative researchers with backgrounds in anthropology and health policy, independently coded the set of errors for the presence of each code in the codebook and then jointly resolved discrepancies. As shown in Figure 3, a large proportion of errors made by the LLM (66% of Reddit FPs, 74% of Reddit FNs, and 96% of Sustainability FPs) are explained by annotation inconsistencies where the human-assigned code deviated from the stated definition of a code.[8]

## 5 Conclusion

We present Muse, a deployed AI-assisted qualitative research system built through iterative engagement with over 100 researchers and currently serving more than 400 users across 600+ projects. We demonstrate that AI-assisted qualitative analysis can achieve research-grade inter-rater reliability while offering steerability and scalability that extend beyond traditional computational approaches.

However, our error analysis and deployment experience underscore the importance of carefully navigating trade-offs between latency, dynamism, and accuracy within qualitative research systems. Our error analysis reinforces this point: the majority of human-AI disagreements stem not from model failures but from inherent ambiguities in coding schemes, suggesting that organizations deploying these systems should invest as much in codebook design as in model selection. Looking forward, the value of AI-assisted qualitative research lies not in automating analysis but in enabling researchers to work at unprecedented scales and efficiencies.

[7] Because the Reddit dataset consists of social media posts rather than technical literature, and the Sustainability dataset is a sample project intended to be understood by researchers in any area, the language and content are more accessible for comprehensive error analysis compared to specialized domains requiring deep subject-matter expertise, such as medical documents or scientific abstracts.

[8] See Appendix E for full error taxonomy and discussion of error types.

## Acknowledgements

We acknowledge extensive support and guidance from the RAND Product Development and Management Team. In particular, William Marcellino, Teresa Ko, and Nelson Lim were highly influential in shaping and directing this research as well as the underlying product it supports.

## References

Aly Abdelrazek, Yomna Eid, Eman Gawish, Walaa Medhat, and Ahmed Hassan. 2023. Topic modeling algorithms and applications: A survey. *Information System*, 112:306–4379.

ATLAS.ti. n.d. Sample projects.

Mehmet Selman Baysan, Serkan Uysal, İrem İşlek, Çağla Çığ Karaman, and Tunga Güngör. 2025. Llm-as-a-judge: Automated evaluation of search query parsing using large language models. *Frontiers in Big Data*, 8:1611389.

Issam Bennis and Safwane Mouaffaq. 2025. Advancing ai-driven thematic analysis in qualitative research: A comparative study of nine generative models on cutaneous leishmaniasis data. *BMC Medical Informatics and Decision Making*, 25(1):124.

Sreyoshi Bhaduri, Satya Kapoor, Alex Gil, Anshul Mittal, and Rutu Mulkar. 2024. Reconciling methodological paradigms: Employing large language models as novice qualitative research assistants in talent management research. *Preprint*, arXiv:2408.11043.

Nikhil Sanjay Borse, Ravishankar Chatta Subramaniam, and N. Sanjay Rebello. 2025. Investigation of the inter-rater reliability between large language models and human raters in qualitative analysis. *Preprint*, arXiv:2508.14764.

Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46.

Rosanna Cole. 2024. Inter-rater reliability methods in qualitative case study research. *Sociological Methods Research*, 53(4):1944–1975.

Thomas Davidson, Dana Warmsley, Michael Macy, and Ingmar Weber. 2017. Automated hate speech detection and the problem of offensive language. In *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM)*, volume 11, pages 512–515.

Stefano De Paoli and Walter S. Mathis. 2025. Reflections on inductive thematic saturation as a potential metric for measuring the validity of an inductive thematic analysis with llms. *Quality Quantity*, 59:683–709.

Norman K. Denzin and Yvonna S. Lincoln, editors. 2011. *The SAGE Handbook of Qualitative Research*, 4 edition. SAGE Publications, Inc., Thousand Oaks, CA.

Oliver Ferschke, Iryna Gurevych, and Yevgen Chebotar. 2012. Behind the article: Recognizing dialog acts in wikipedia talk pages. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 777–786, Avignon, France. Association for Computational Linguistics.

Uwe Flick. 2014. *An Introduction to Qualitative Research*, 5 edition. SAGE Publications Ltd, Thousand Oaks, CA.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Saizhuo Wang, Kun Zhang, Yuanzhuo Wang, Wen Gao, Lionel Ni, and Jian Guo. 2025. A survey on llm-as-a-judge. *Preprint*, arXiv:2411.15594.

Andy Hickner. 2022. Data for "how do search systems impact systematic searching? a qualitative study". QDR Main Collection. Version 1 (V1).

Ariba Khan, Stephen Casper, and Dylan Hadfield-Menell. 2025. Randomness, not representation: The unreliability of evaluating cultural alignment in llms. In *Proceedings of the 2025 ACM Conference on Fairness, Accountability, and Transparency*, pages 2151–2165, New York, NY, USA. Association for Computing Machinery.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. *Preprint*, arXiv:2309.06180.

Joonatan Laato, Matti Mäntymäki, Bastian Kordyaka, and Samuli Laato. 2025. Automating qualitative data analysis with chain-of-thought reasoning models: A study with the gioia method. In *AMCIS 2025: Proceedings of the 31st Americas Conference on Information Systems*, Montréal, Canada. AIS Electronic Library (AISeL). Paper 2130.

Michelle S. Lam, Janice Teoh, James A. Landay, Jeffrey Heer, and Michael S. Bernstein. 2024. Concept induction: Analyzing unstructured text with high-level concepts using lloom. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, page Article 766, New York, NY, USA. Association for Computing Machinery.

David D. Lewis. 1987. Reuters-21578 text categorization collection. UCI Machine Learning Repository.

Haitao Li, Qian Dong, Junjie Chen, Huixue Su, Yujia Zhou, Qingyao Ai, Ziyi Ye, and Yiqun Liu. 2024. Llms-as-judges: A comprehensive survey on llm-based evaluation methods. *Preprint*, arXiv:2412.05579.

Sara Mannheimer. 2023. Interviews regarding data curation for qualitative data reuse and big social research. QDR Main Collection, Version V1.

David L. Morgan. 2023. Exploring the use of artificial intelligence for qualitative data analysis: The case of chatgpt. *International Journal of Qualitative Methods*, 22:16094069231211248.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *Preprint*, arXiv:2112.00114.

Jiaxing Qiu. 2025. Topic annotations on reddit posts from eating disorders and dieting forums by human and llms. University of Virginia Dataverse, Version V3.

Johnny Saldaña. 2012. *The Coding Manual for Qualitative Researchers*, 2 edition. SAGE Publications Ltd, London, UK.

Miriam Schirmer, Isaac Misael Olguín Nolasco, Edoardo Mosca, Shanshan Xu, and Jürgen Pfeffer. 2023. Uncovering trauma in genocide tribunals: An nlp approach using the genocide transcript corpus. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, pages 257–266, Braga, Portugal. Association for Computing Machinery.

Tim Schopf, Alexander Blatzheim, Nektarios Machner, and Florian Matthes. 2024. Efficient few-shot learning for multi-label classification of scientific documents with many classes. In *Proceedings of the 7th International Conference on Natural Language and Speech Processing (ICNLSP 2024)*, pages 186–198, Trento, Italy. Association for Computational Linguistics.

Tim Schopf, Daniel Braun, and Florian Matthes. 2023. Evaluating unsupervised text classification: Zero-shot and similarity-based approaches. In *Proceedings of the 6th International Conference on Natural Language Processing and Information Retrieval (NLPIR)*, pages 6–15, New York, NY, USA. Association for Computing Machinery.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. 2025. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges. In *Proceedings of the Fourth Workshop on Generation, Evaluation and Metrics (GEM$^2$)*, pages 404–430, Vienna, Austria and Virtual. Association for Computational Linguistics.

Qile Wang, Moath Erqsous, Kenneth E. Barner, and Matthew Louis Mauriello. 2025. Lata: A pilot study on llm-assisted thematic analysis of online social network data generation experiences. *Association for Computing Machinery*, 9(2):1–28.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, pages 24824–24837.

Yiming Yang and Xin Liu. 1999. A re-examination of text categorization methods. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 42–49, New York, NY, USA. Association for Computing Machinery.

Elena Zotova, Rodrigo Agerri, Manuel Nuñez, and German Rigau. 2020. Multilingual stance detection in tweets: The catalonia independence corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1368–1375, Marseille, France. European Language Resources Association.

## A Benchmarking Datasets

The datasets used for evaluation displayed in Table 4 include Reuters-21578, a canonical dataset for text classification, two other text classification datasets containing medical and scholarly article abstracts, and eight datasets generated specifically within qualitative research projects. We identified the text classification datasets by reviewing benchmark datasets used in previous machine learning research on text classification (Saldaña, 2012; Yang and Liu, 1999). To locate additional datasets from qualitative research projects, we searched Syracuse University's Qualitative Data Repository, UK Data Service, Center for Open Science, Papers With Code, TU Darmstadt's TUdatalib, Kaggle, and Google Dataset Search for annotated qualitative data and interviews.

To provide a rough reference point for the relative quality of each dataset, we assigned a 1-10 dataset quality score based on an unweighted average of our assessment of each dataset's codebook definition completeness, annotation consistency, and document completeness.

A score between 1-3 for codebook definition completeness means that few, if any, definitions are provided, and label names are not specific enough to apply the code. A score between 4-7 means that most labels contain a definition or descriptive name, but definitions are generally too short or vague to be of much use. A score of 8 or higher means that specific definitions containing criteria and nuanced theme understanding are provided.

For annotation consistency, a score of 1-3 means that codes are applied without any regularity or

| Dataset | Description | Type | Docs | Words | Codes | Quality |
|---|---|---|---|---|---|---|
| Reuters-21578 (1987) | Reuters newswire docs | News | 18.9K | 2.4M | 135 | 6/10 |
| Medical Abstracts (2023) | Medical abstracts | Medical | 11.6K | 2.1M | 5 | 7/10 |
| arXiv Categories (2024) | arXiv titles & abstracts | Academic | 163K | 24.1M | 142 | 9/10 |
| Reddit Posts (2025) | Eating disorders forums | Social media | 1.1K | 135K | 15 | 6/10 |
| Data Curation Interviews (2023) | Qualitative reuse study | Interviews | 29 | 200K | 238 | 5/10 |
| Search Systems (2022) | Systematic searching study | Interviews | 12 | 63K | 83 | 4/10 |
| Genocide Transcripts (2023) | Genocide tribunal | Legal | 52.8K | 1.8M | 1 | 7/10 |
| Wiki Discussion (2012) | Wikipedia talk pages | Discussion | 1.8K | 127K | 21 | 4/10 |
| Catalonia Independence (2020) | Spanish & Catalan Tweets | Social media | 12.1K | 362K | 3 | 4/10 |
| Hate Speech (2017) | Tweets w/ hate speech | Social media | 24.8K | 350K | 2 | 6/10 |
| ATLAS.ti (n.d.) | Sustainability interviews | Interviews | 30 | 13K | 65 | 8/10 |

Table 4: Datasets Used to Evaluate Muse. Codes are the number of unique human-coded labels and themes in each dataset. Quality is a 1-10 dataset quality score based on an unweighted average of our assessment of codebook definition completeness, annotation consistency, and document completeness. ATLAS.ti dataset used only to evaluate codebook generation and for error analysis.

experience significant inconsistencies because of study design or execution. A score of 4-7 means that code application generally aligns with stated criteria but has some deviations. A score of 8 or higher means no noticeable deviations in code application are present.

Document completeness is a measure of whether sufficient context is provided in documents to replicate the author's codings. A score of 1-3 means that it is impossible to replicate codings because the author(s) referenced other material to make their judgments that is not provided. A score of 4-7 means that some document context is omitted for each code, but codings are still made at the level of each document and can mostly be replicated. A score of 8 or higher means that very few document context concerns exist.

## B Prompts

The prompt text leveraged for the *apply code* method is shown in Figure 4 and for the *distribute code* method in Figure 5. These prompts are the foundational prompts for Muse's annotation features; however slightly modified prompts are used under different coding conditions.

## C Codebook Similarity Score

For every pairwise combination of codes between LLM-generated ($C_L$) and human-generated ($C_H$) codebooks, we computed:

1. **Semantic similarity** ($S_{\text{sim}}$): Cosine similarity between sentence embeddings of code labels and descriptions

2. **Structural similarity** ($S_{\text{str}}$): Hierarchical position, path length, and subtree characteristics within each codebook where

$$S_{\text{str}}(c_1, c_2) = s_{\text{level}}(c_1, c_2) + s_{\text{path}}(c_1, c_2) + s_{\text{subtree}}(c_1, c_2)$$

where $s_{\text{level}}$ measures depth similarity, $s_{\text{path}}$ compares root-to-node distances, and $s_{\text{subtree}}$ evaluates organizational complexity beneath each code.

These components were combined into a composite similarity score, then optimally aligned using the Hungarian algorithm to find the best global matching between codebooks. After matching all codes from $C_H$ to their associated codes in $C_L$, we calculated the average weighted similarity score across each pair,

$$S_A = \alpha S_{\text{sim}} + \beta S_{\text{str}},$$

providing a final codebook similarity score, $\bar{S}_A$, averaged across all pairs that considers both content and organizational logic. Here, $\alpha + \beta = 1$, and both variables were varied across the interval $[0.1, 0.9]$, with the final $\bar{S}_A$ value averaged across these weightings to ensure balance across both similarity components.

## D Hyperparameter Search and Benchmarking Results

### D.1 Hyperparameter Description

The hyperparameters tested included LLM-as-a-judge scoring type, chain-of-thought (CoT) prompting, batch size, and few-shot examples. In binary scoring, we prompted the LLM to produce a simple "Yes" or "No" response indicating whether a document excerpt related to a given code. Alternatively, in discretized scoring, we prompted the

```
You are assisting a qualitative researcher with tagging interview statements. Your task is to decide whether each provided statement should be tagged with the code: **‘tag‘**.
You will use the provided **code definition**, **tagged examples**, **untagged examples**, **inclusion criteria**, and **exclusion criteria** to make your decision. Your decision-making process should prioritize:
1) Alignment with the code name/definition. 2) Satisfying at least one inclusion criterion while satisfying none of the exclusion criteria. 3) Consistency with provided positively tagged examples, avoiding similarities to negatively tagged examples.
A worked-out example using the **AI for literature review tag** is provided to guide your decision-making process.
—
Output Instructions
For each statement, provide: 1. A brief explanation (5–10 words) of your reasoning 2. A confidence score (1–10) where 10 indicates highest confidence the statement should be tagged and 1 indicates the statement should definitely not be tagged
Your output will be a JSON with one numeric key for every provided input statement. The value for each key will be a JSON with two keys: "REASONING" and "SCORE".
—
Worked Example: Literature Review Tag
**Code Name**: Using AI for literature reviews
**Code Definition**: This code applies to statements that reference the use of Artificial Intelligence (AI) or Machine Learning (ML) to assist with literature reviews. Specifically, it focuses on how AI is applied to discover, organize, or summarize academic or scientific publications.
**Inclusion Criteria**: 1. Mentions AI or ML in the context of finding, summarizing, or organizing academic articles or studies. 2. Names a specific AI model or approach (e.g., GPT, BERT) used to conduct or facilitate a literature review. 3. Clearly states "literature review" (or a recognizable synonym, such as "systematic review") while also describing an AI-based process.
**Exclusion Criteria** 1. Mentions the word "plagiarism" (indicating AI is being used for plagiarism detection instead of literature review). 2. Describes AI usage for purposes unrelated to literature reviews (e.g., AI for financial forecasting, image classification). 3. Focuses on manual processes or workflows unrelated to AI (e.g., manually reviewing case studies without any AI involvement).
**Tagged Examples (Relevant Statements)**: 1. "We developed a GPT-based tool to summarize new articles for our scoping review." 2. "Our research uses AI-based clustering to find relevant papers, automating the literature review process."
**Untagged Examples (Irrelevant Statements)**: 1. "An AI model helped detect plagiarism in graduate theses before publication." 2. "Our study uses advanced ML to analyze survey data from an ongoing clinical trial."
Statements to Evaluate
1. "During our systematic review, we employed both GPT-4 for synthesizing findings and traditional manual screening methods in parallel."
2. "The team customized a BERT model to identify methodological weaknesses across published papers, though this was separate from our literature review process."
3. "Our platform combines AI text summarization with human expert oversight to conduct comprehensive literature reviews twice as fast as traditional methods."
4. "We processed 5,000 medical journals using deep learning, but ultimately our literature review was conducted through conventional critical appraisal techniques."
5. "The research assistant built an ML classifier that predicts which papers a researcher might want to include in their literature review based on previous selections."
6. "We’re using AI to extract study characteristics from papers we’ve already identified, but the actual literature search and selection was done manually following PRISMA guidelines."
7. "Our software uses natural language processing to generate personalized reading lists from academic databases, though it’s primarily designed for teaching purposes rather than formal literature reviews."
8. "We employed a GPT approach for our systematic review, but we also leveraged a plagiarism module to ensure originality."
Example Output
"1": "REASONING": "GPT-4 used directly for systematic review synthesis", "SCORE": 9, "2": "REASONING": "AI for paper analysis, not literature review", "SCORE": 3, "3": "REASONING": "AI summarizes texts for literature reviews", "SCORE": 10, "4": "REASONING": "AI only for processing, not review tasks", "SCORE": 4, "5": "REASONING": "ML assists paper selection for literature review", "SCORE": 8, "6": "REASONING": "AI extracts data from already-selected papers", "SCORE": 6, "7": "REASONING": "NLP for academic lists, not formal reviews", "SCORE": 5, "8": "REASONING": "GPT for review but mentions plagiarism detection", "SCORE": 7

Task Instructions for **tag**
Now, evaluate the following statements for the code **tag** following the same procedure as above. In some cases, I will also provide either (1) the question that the statement was given in response to OR (2) text that immediately preceded the statement of interest (denoted with '...'). Remember you are evaluating the STATEMENT itself - do not provide a high confidence score if the preceding context/question is related to the code but the statement itself is not.
I may also provide additional context on the qualitative research dataset from which the following statements extracted below. If provided, use this additional context to help guide your decision-making.
{project_context}
**Code Name:** {tag}
**Code Definition:** {tag_definition} {tag_context}
**Inclusion Criteria:** {inclusion_criteria}
**Exclusion Criteria:** {exclusion_criteria}
**Tagged Examples (Relevant Statements)**: {positive_statements}
**Untagged Examples (Irrelevant Statements)**: {negative_statements}
Statements to Evaluate {question_and_statements}
```

Figure 4: Prompt template for single-code application (*apply code*). `{tag}` is the name of the code being annotated, `{tag_definition}` is the definition of this associated code, `{tag_context}` is an arrow diagram describing the tag's position within the codebook hierarchy, `{question_and_statements}` are the excerpts to be evaluated as well as their preceding text provided as context, `{inclusion_criteria}` and `{exclusion_criteria}` are the associated code's inclusion and exclusion criteria (respectively), `{project_context}` is a brief user-specified description of the dataset being analyzed, and `{positive_statements}` and `{negative_statements}` are user-specified few-shot examples of both excerpts that should be coded with the associated code as well as those that should not, respectively.

```
You are an assistant to a qualitative researcher. Your task is to assign each statement from a qualitative dataset to subthemes based on the provided parent theme and subtheme definitions. If a statement does not clearly match any of the subthemes, respond with [].
—
Instructions
1. **Input Data:** - **PARENT THEME:** The overarching topic discussed within the statements - **SUBTHEMES:** A list of subthemes, each identified by an alphabetic identifier (e.g., ‘A. Positive sentiment‘). - **STATEMENTS:** A set of statements from the dataset, each identified by a numeric identifier. In some cases, I will also provide either (1) the question that the statement was given in response to OR (2) text that immediately preceded the statement of interest (denoted with ’...’). Do not apply a code if the preceding context/question is related to a code but the statement itself is not.
2. **Assignment Rules:** - **Flexible Assignment:** Each statement can be assigned to **one, multiple, or no** subthemes. - **Default Assignment:** If none of the provided subthemes clearly match a statement, assign [] to that statement. - **Strict Criteria:** Only assign a subtheme if the statement clearly meets the subtheme’s definition, inclusion criteria, and any provided examples.
3. **Output Format:** - Your response should be in JSON format. - Each key should be the numeric identifier of a statement (STATEMENT X formatted as ‘”X”‘). - Each value should be a list of the alphabetic identifier(s) of the assigned subtheme(s) (e.g., ‘[”A”]‘) or [] if no subthemes apply.
—
Example
**Input:**
- **PARENT THEME:** Social Media Posts About the Election
- **SUBTHEMES:** - A. Positive sentiment - B. Negative sentiment
- **STATEMENTS:** 1. I am so anxious and worried about the election 2. I can’t wait for the election to come because I’m excited to vote! 3. Thinking about the election is making me depressed 4. I’m really scared there is going to be another uprising again and people will get hurt 5. I’m feeling indifferent about the election. Given the candidates running, there really is much difference between them in terms of how they are going to change the country.
**Expected Output:**
”1”: [”B”], ”2”: [”A”], ”3”: [”B”], ”4”: [”B”,”C”], ”5”: []
**Explanation:**
- 1: Clearly aligns with negative sentiment (SUBTHEME B) - 2: Clearly aligns with positive sentiment (SUBTHEME A) - 3: Clearly aligns with negative sentiment (SUBTHEME B) - 4: Associated with both negative sentiment (SUBTHEME B) and statements about concern for violence (SUBTHEME C) - 5: Has a neutral tone and does not clearly match any subtheme, so it is assigned the default value of [].
—
Additional Information
At times, you may be provided with additional context on the qualitative research project from which the statements are extracted as well as further details for each subtheme (e.g., definitions, inclusion/exclusion criteria, positive and negative examples). Use this information to guide your assignment decisions.
Task Now, perform this task for the inputs below:
{project_context}
PARENT THEME: {theme}
SUBTHEMES: {subthemes}
STATEMENTS: {statements}
Please ensure your output is valid JSON with each statement key correctly mapped to a list of its assigned subtheme alphabetic identifier(s).
```

Figure 5: Prompt template for multi-code application (*distribute code*). Variables in curly braces are populated with user-specified content at runtime. `{statements}` are the set of statements to be evaluated, `{subthemes}` are the set of subcodes associated with the distribute operation along with their definitions, `{theme}` is the parent code of the associated subcodes along with its definition (if available), and `{project_context}` is a brief user-specified description of the dataset being analyzed.

| Batch Size | Few-Shot Examples | Scoring Type | CoT | F1 Score [95% CI] | Cohen's $\kappa$ [95% CI] |
|---|---|---|---|---|---|
| 5 | 4 | Discretized | Yes | **0.686 [0.611, 0.762]** | **0.545 [0.427, 0.662]** |
| 5 | 2 | Discretized | Yes | 0.670 [0.599, 0.742] | 0.532 [0.425, 0.639] |
| 10 | 2 | Discretized | Yes | 0.661 [0.577, 0.745] | 0.524 [0.408, 0.640] |
| 10 | 4 | Discretized | Yes | 0.663 [0.577, 0.748] | 0.521 [0.403, 0.640] |
| 5 | 2 | Binary | Yes | 0.657 [0.579, 0.734] | 0.518 [0.407, 0.628] |
| 10 | 2 | Discretized | No | 0.644 [0.563, 0.725] | 0.517 [0.406, 0.629] |
| 5 | 4 | Binary | Yes | 0.661 [0.583, 0.738] | 0.513 [0.398, 0.628] |
| 10 | 2 | Binary | Yes | 0.653 [0.574, 0.733] | 0.510 [0.399, 0.620] |
| 5 | 4 | Discretized | No | 0.652 [0.585, 0.719] | 0.508 [0.407, 0.610] |
| 5 | 2 | Binary | No | 0.645 [0.573, 0.717] | 0.508 [0.405, 0.611] |
| 10 | 4 | Binary | Yes | 0.652 [0.576, 0.728] | 0.507 [0.403, 0.610] |
| 5 | 0 | Binary | Yes | 0.651 [0.568, 0.735] | 0.502 [0.389, 0.616] |
| 5 | 0 | Discretized | Yes | 0.648 [0.566, 0.730] | 0.499 [0.387, 0.611] |
| 5 | 2 | Discretized | No | 0.634 [0.562, 0.705] | 0.498 [0.398, 0.599] |
| 10 | 0 | Discretized | Yes | 0.645 [0.566, 0.723] | 0.495 [0.385, 0.605] |
| 10 | 4 | Binary | No | 0.631 [0.556, 0.706] | 0.487 [0.381, 0.593] |
| 10 | 2 | Binary | No | 0.625 [0.550, 0.701] | 0.487 [0.379, 0.596] |
| 10 | 0 | Binary | Yes | 0.638 [0.563, 0.713] | 0.487 [0.384, 0.590] |
| 5 | 4 | Binary | No | 0.633 [0.560, 0.706] | 0.485 [0.382, 0.588] |
| 10 | 4 | Discretized | No | 0.623 [0.540, 0.706] | 0.481 [0.367, 0.596] |
| 10 | 0 | Discretized | No | 0.618 [0.535, 0.701] | 0.476 [0.366, 0.587] |

Table 5: Complete hyperparameter optimization grid search for code application with GPT-4o (sorted by Cohen's $\kappa$, highest bolded). Batch size is the number of excerpts included in a single prompt (5 or 10). Few-shot examples tested were 0, 2, and 4. Discretized scoring is a 1-10 score generated by the LLM, whereas binary is a "yes" or "no" response. CoT prompting involves the LLM providing a short (1 sentence) explanation of its output prior to producing it.

| Model Name | Optimal Threshold | Untuned Cohen's $\kappa$ [95% CI] | Code-tuned Cohen's $\kappa$ [95% CI] |
|---|---|---|---|
| gpt-4.1-2025-04-14-global | 8 | 0.515 [0.426, 0.604] | **0.593 [0.512, 0.674]** |
| o3-2025-04-16-global | 9 | **0.520 [0.423, 0.617]** | 0.590 [0.508, 0.672] |
| gpt-5-2025-08-07-us (default) | 9 | 0.515 [0.422, 0.608] | 0.578 [0.498, 0.658] |
| grok-3-v1 | 8 | 0.505 [0.425, 0.585] | 0.572 [0.496, 0.648] |
| grok-3-mini-v1 | 9 | 0.492 [0.402, 0.582] | 0.553 [0.470, 0.636] |
| o4-mini-2025-04-16-global | 9 | 0.492 [0.398, 0.586] | 0.546 [0.459, 0.633] |
| gpt-oss-120b (high reasoning) | 9 | 0.486 [0.386, 0.586] | 0.543 [0.450, 0.636] |
| Microsoft MAI-DS-R1 | 9 | 0.499 [0.413, 0.585] | 0.539 [0.456, 0.622] |
| gpt-4o-2024-08-06-us | 8 | 0.461 [0.372, 0.550] | 0.536 [0.453, 0.619] |
| gpt-oss-120b (low reasoning) | 8 | 0.483 [0.388, 0.578] | 0.535 [0.446, 0.624] |
| Llama-3.1-Nemotron-Ultra-253B-v1 (reas.) | 9 | 0.470 [0.382, 0.558] | 0.525 [0.440, 0.610] |
| DeepSeek-V3-0324 | 8 | 0.464 [0.381, 0.547] | 0.525 [0.445, 0.605] |
| DeepSeek-R1-0528 | 9 | 0.471 [0.384, 0.558] | 0.524 [0.445, 0.603] |
| Llama-3.3-Nemotron-Super-49B-v1 (reas.) | 9 | 0.466 [0.376, 0.556] | 0.516 [0.434, 0.598] |
| Llama-3.1-Nemotron-Ultra-253B-v1 (non-r.) | 9 | 0.428 [0.344, 0.512] | 0.488 [0.406, 0.570] |
| Llama-3.3-Nemotron-Super-49B-v1 (non-r.) | 9 | 0.429 [0.343, 0.515] | 0.484 [0.403, 0.565] |
| gpt-4.1-mini-2025-04-14-global | 9 | 0.383 [0.299, 0.467] | 0.454 [0.372, 0.536] |
| Nemotron-H-47B-Reasoning-128K (reas.) | 10 | 0.379 [0.296, 0.462] | 0.420 [0.343, 0.497] |
| Nemotron-H-47B-Reasoning-128K (non-r.) | 10 | 0.290 [0.216, 0.364] | 0.332 [0.262, 0.402] |

Table 6: Complete inter-rater reliability results by LLM (sorted by code-tuned Cohen's $\kappa$, highest bolded). Optimal threshold is a constant confidence threshold (1-10) for assigning positive and negative codes that optimizes IRR for each model. Code-tuned means setting a separate confidence score threshold for each code to align with the level of sensitivity in the human-coded data. Untuned means that every code across datasets uses the same "optimal threshold" to distinguish positives and negatives on a 1-10 scale.

LLM to rate each excerpt on a 1-10 relevance scale, allowing users to tune threshold confidence scores for determining final code assignments. When evaluating discretized scoring, we chose a threshold value that optimized Cohen's $\kappa$ in each code, simulating the performance a Muse user would observe if actively tuning this parameter within the platform. All reported Cohen's $\kappa$ values for discretized scoring in this work leverage this convention.

CoT prompting requests the LLM to provide brief explanations for its coding decisions prior to scoring, potentially improving both accuracy and

| Model | 4-shot | n=128 | n=256 | n=512 |
|---|---|---|---|---|
| GPT-4.1 + o3 + Grok3 (tuned) | **0.703** | — | — | — |
| GPT-4.1 + o3 + Grok3 (untuned) | 0.660 | — | — | — |
| GPT-4.1 (tuned) | 0.693 | — | — | — |
| GPT-4.1 (untuned) | 0.641 | — | — | — |
| GemmaEmb + LinearSVC | — | **0.531** | **0.574** | **0.607** |
| GemmaEmb + LogisticRegression | — | 0.518 | 0.538 | 0.556 |
| SBERT (MiniLM) + LinearSVC | — | 0.461 | 0.495 | 0.521 |
| SBERT (MiniLM) + LogisticRegression | — | 0.482 | 0.505 | 0.509 |
| TF-IDF + LogisticRegression | — | 0.137 | 0.349 | 0.465 |
| TF-IDF + LinearSVC | — | 0.117 | 0.290 | 0.418 |

Table 7: Cohen's $\kappa$ by model and training set size across 12 well-specified codes. LLM-based approaches use few-shot prompting (4 positive examples), while supervised ML methods use $n = 128$, 256, and 512 random samples as the training set. Supervised ML results averaged over 20 iterations. Bold values indicate highest performance within each column.

interpretability. We also varied batch size (number of excerpts processed simultaneously) and the number of few-shot examples provided, as both factors can significantly affect model performance and computational efficiency.

### D.2 Complete Hyperparameter Optimization Results

Table 5 displays the full set of hyperparameter combinations we tested. We observed that batch size did not significantly impact IRR while CoT improves performance for the GPT-4o model. We also found that discretized scoring, on average, produced higher IRR than its binary counterpart, with a confidence threshold of greater than or equal to 7 generally found to be optimal. Providing few-shot examples also increased performance; however, we did not observe a clear difference between two positive examples versus four positive examples.

IRR varied significantly by dataset, code, and prompting strategy. More abstract topics that require an additional level of interpretation, and thus subjectivity, tended to have lower IRR between the LLM and human code. Such codes are often sentiment-related (e.g., "in favor," "against," or "neutral" with respect to Catalan independence) or about the intent of language (e.g., "offensive language" or "hate speech"). These codes collectively averaged a Cohen's $\kappa$ of 0.23.

### D.3 Complete LLM Benchmarking Results

Table 6 displays IRR for all LLMs that we tested. Distilled, non-reasoning models, like gpt-4.1-mini and the Nemotron 49B and 47B models, lag behind full-sized and/or reasoning-based models considerably, while distilled reasoning models like o4-mini and Grok 3 Mini are much closer in IRR to top-performing models. OpenAI's gpt-oss-120b and Microsoft's post-trained version of DeepSeek-R1 (which fills information gaps in the original model) are the most aligned open-source models. Because gpt-oss is significantly smaller (nearly 6 times fewer parameters), it is the preferred open-source choice for optimizing cost, latency, and throughput.

### D.4 Comparison to Supervised ML Methods

We compared Muse coding performance to other supervised ML methods (sML) used for annotation on the 12 well-specified codes in our evaluation dataset. For the LLMs, we used GPT-4.1 as well as the ensemble of models and test both with a constant confidence score threshold across all codes and confidence score threshold tuning for each code. For the supervised ML methods, we used three different approaches to vectorize the text dataset: EmbeddingGemma, SBERT (all-MiniLM-L6-v2), and TF-IDF. For classification, we used LinearSVC and Logistic Regression in scikit-learn.

Table 7 shows that Muse outperforms all the sML methods we tested even with training sets with hundreds of examples for sML and only 4 positive examples and definitions for Muse. The best-performing sML approaches tested use EmbeddingGemma, which approaches untuned GPT-4.1 performance as the number of training examples increases. It's possible other embedding models and classification algorithms may result in higher performance than EmbeddingGemma and LinearSVC. Still, the LLM approach is more flexible for explor-

| Code | Definition |
|---|---|
| Code under-specification | The target code is defined too broadly and/or has ambiguity in its application criteria. This includes codebooks where multiple codes may have overlapping scope and application criteria. |
| Contextual limitation | Insufficient surrounding text or document context in excerpt to assess applicability. This includes tangential mentions of the target theme that are not salient concepts in the document. |
| Annotation inconsistency | Application of code is a deviation from its definition, criteria, and other positive examples. |
| Thresholding error | Assigned confidence score and reasoning are aligned with human code, but confidence score threshold is suboptimal for the particular excerpt. |
| Other LLM error | Other inexplicable error by the LLM. Reasoning and confidence score are incorrect. |

Table 8: Muse Error Codebook

ing qualitative datasets because it allows tweaking definition and refining codes iteratively as opposed to manually coding hundreds of examples during each iteration.

## E Error Analysis Discussion

To gain a better understanding of discrepancies in code application with human coders, we sampled 50 false positives and false negatives and analyzed them qualitatively by considering:

- Ground-truth positives associated with each target code
- The complete set of codes applied by the human researcher to a considered excerpt
- Target code label and definition
- Reasoning strings produced by the LLM
- LLM confidence scores

After an initial review, we iteratively developed and refined the codebook in Table 8 to describe the root causes of the observed errors. Two authors independently coded the errors for the presence of each code in the codebook and then jointly resolved discrepancies by refining code definitions and discussing differing interpretations.

Annotation inconsistencies, which were the most common error type, could potentially be due to coder fatigue or interpretive drift, and/or in the case of the Sustainability dataset, overly complex and long codebooks leading to inconsistent application of sub-codes. The observed inconsistencies highlight the broader challenge of establishing reliable ground truth in qualitative research and ultimately may reduce our ability to definitively assess LLM performance against human coders.

While the Reddit eating disorder dataset had a better specified codebook than many of the other evaluation datasets, most errors (80% of FPs and 71% of FNs, 18% of Sustainability FPs) were also incorrectly labeled in part or in full due to under-specified target codes lacking specific definitions. This often leads to code confusion between semantically similar codes with insufficient disambiguation criteria. The most frequent example of such confusion in the sample was between the code "thinspiration," defined as "drive for thinness, want to be thinner or skinny," and the code "weight loss," defined as "body weight loss."

Code under-specification often took the form of target codes with definitions that vaguely described the concept of interest. For example, the label "relationships" is defined as "family and social relationships," but lacked sufficient detail and criteria to accurately apply (or not apply) the code to Reddit posts about social interactions, relationship types, and community engagement. Adding specific information in the definition that the human coders implicitly or explicitly utilized when coding, such as "mention of family members, relatives, friends, significant others, or social roles," would have provided the LLM with much-needed context to improve alignment with the ground truth coding scheme.

Given this error analysis, if the following easily implementable steps are taken, we believe the error rate can be reduced substantially. First, codes

should be fully specified. This means providing complete concept definitions that provide clarity about what specifically is of interest to the researcher and what should be included and excluded. Second, if two codes are semantically similar, the researcher should describe how they differ and when one should be applied or both should be applied. If both appear to be describing the same underlying construct, the researcher should consider merging them. Finally, from the perspective of system development, some errors due to a lack of full document context could be addressed by processing full documents in single prompts as opposed to chunked excerpts - a more complex task yet one increasingly in reach as LLM context windows grow.