Are LLMs Reliable for Qualitative Coding in Security?

Sumit Nawathe
snawathe@umd.edu
University of Maryland
College Park, MD, USA

Sahit Kadthala
kadthala@umd.edu
University of Maryland
College Park, MD, USA

Ayman Chowdhury
achowdh1@umd.edu
University of Maryland
College Park, MD, USA

Abstract

Qualitative Coding is an important tool for classifying unstructured data and identifying its fundamental themes. However, the process can be very time-consuming and tedious to do. The rapid advancement of large-language models (LLMs) offers opportunities to automate much of this procedure. While prior studies have explored LLMs in qualitative coding across various domains, this work investigates their reliability specifically for security-related data. Using a dataset of tweets about users' perceptions of ChatGPT in security contexts, we conducted manual qualitative coding, iteratively refining the codebook to better represent the data. Afterwards, we passed these codebook iterations to Llama 3 and GPT-4o mini as well as various well-crafted prompts and compared their results with the manual ones. Chain-of-Thought prompting, which asks the model to justify its decisions, significantly improved performance. Our findings indicate that LLMs perform reasonably well with mature, well-defined codebooks but are not yet capable of fully automating the classification process. We propose an LLM-human codebook co-development approach to leverage the strengths of both, enabling faster, accurate, and reliable qualitative coding.

Keywords

Qualitative Coding, LLMs, Llama 3.1 8B, ChatGPT-4o Mini

ACM Reference Format:
Sumit Nawathe, Sahit Kadthala, and Ayman Chowdhury. 2024. Are LLMs Reliable for Qualitative Coding in Security?. In *Proceedings of 7th Annual UMD GradSec Symposium (GRADSEC).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Qualitative Coding is a procedure for systematically classifying qualitative data. In practice, this procedure is often performed by several reviewers who come together to develop a shared classification system known as a codebook, which they then utilize to classify subsets of the data on their own; periodically, they reconvene to resolve discrepancies, refine the codebook, and maintain consistency. This procedure is especially prominent in security subfields such as human-centric security, where it serves as a vital tool to understand how users and developers perceive threat models, engage with security tools and protocols, and make security-related decisions. For example, [7] employed qualitative coding to classify vulnerabilities commonly introduced by developers, examining their type, the degree of attacker control they allowed, and their ease of exploitation to provide valuable insights into mitigation strategies. Similarly, [1] applied qualitative coding to classify how and why developers propagated insecure code from Stack Overflow.

Although qualitative coding is a vital tool for classifying unstructured data and uncovering overarching patterns in it, the procedure is time-consuming and requires significant manual effort. The need for reviewers to not only label the data but also periodically resolve discrepancies, refine the codebook, and maintain consistency means that the procedure becomes increasingly challenging as the data grows in size and complexity. The rise of large-language models (LLMs) in recent years opens up the door to automate much of this work, leveraging their ability to process and interpret unstructured information, identify patterns, and generate consistent labels based on context. Notably, several new studies, including [2], have explored using LLMs to assist with labeling general qualitative data across various domains. A pertinent question is whether LLMs are reliable for qualitative coding in security. Specifically, the central question we aim to investigate in this paper is as follows: are LLMs reliable enough to interpret a codebook and apply it correctly to security-related data?

To address this question, we evaluated the performance of two popular LLMs, Llama 3 and GPT-4o mini using the dataset presented in [5], which contains tweets about users' perceptions about ChatGPT in the context of security. We analyzed prompting strategies to identify those that improved model performance and applied the most effective of such strategies to assess how well these models could interpret and apply predefined codebooks. These predefined codebooks were developed from two of us conducting qualitative coding on the dataset and updating the codebook at various points to better capture the characteristics of the tweets encountered throughout the procedure. Each codebook iteration has a sentiment category, a discussion type category, and a topic category, which is consistent with the format used in [5], and each one has the same exact codes; the only difference between them is the code definitions, which become progressively more refined with each codebook. With that in mind, our approach was to systematically apply the codebooks to the LLMs using well-crafted prompts and compare their outputs against the manual coding results, which served as the ground truth.

Our results demonstrate that that the LLMs performed better when prompted to justify their answers. Additionally, they show that LLMs can be reasonably reliable when using mature codebooks. Overall, our findings suggest that qualitative coding is most effective when humans and LLMs collaborate and take advantage of each other's strengths. More detailed insights can be found in the subsequent sections.

GRADSEC, December 03–05, 2024, College Park, MD
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

2 Related Work

We briefly discuss related work that informed our investigation, including studies on security-related qualitative coding, guidelines for utilizing LLMs in qualitative coding, and approaches for measuring inter-coder agreement (ICA) to analyze our results.

[5] performed qualitative coding to understand users' perceptions of ChatGPT in security contexts and tested its most frequently mentioned use cases. To collect data, the researchers used the Twitter API to compile a list of tweets that matched keywords closely associated to the topic and filtered for the most relevant ones. Their qualitative analysis examined the users' sentiment (the user's feelings about how ChatGPT might influence society), discussion type (the manner in which the user conveyed the information), and topic (the specific content the user discussed). Among the security tasks discussed, vulnerability detection stood out as a prominent focus. The researchers investigated ChatGPT's performance in this domain and found that ChatGPT often provided generic responses, indicating that it may not yet be suitable for the industry-level vulnerability detection that many users were hoping for. The paper provides an online reproduction package, including its dataset of annotated tweets. Although the study did not share the paper's codebooks or labels outright (we did the full qualitative coding ourselves), the outline of analysis it used (general direction, and the categories contining the codes) helped inform the structure and formatting of our own codebooks during the process.

Building on the insights from [5], we now turn to guidelines for effectively utilizing LLMs in qualitative coding. [3] discusses the influence of Chain-of-Thought prompting in qualitative analysis. Chain-of-Thought prompting involves asking the LLM to justify its coding decisions, and the study found that this approach resulted in more accurate model behavior. Given the success of Chain-of-Thought prompting in general qualitative coding scenarios, we sought to investigate its potential in the context of security-related qualitative coding. This result motivated a part of our experiment in which we compared the performance of LLMs when asked to justify their coding decisions on the ChatGPT-Security-Twitter dataset versus when they were not. Further contributing to this discussion, [2] outlines a tested approach called LLM-assisted content analysis (LACA) for humans to interact with LLMs during qualitative coding. It emphasizes evaluating the LLM's understanding of the coding task, refining the codebook to enhance clarity, and comparing human-human agreement with human-model agreement, all of which are integral to the framework of our work.

[2] also analyzes the utility of LLM for deductive coding, but places more emphasis on experiment design, dataset and code variation, and prompt construction; we loosely based the design of our experiments on their work. The authors found that while GPT-3.5 performed poorly, GPT-4 was competent for qualitative coding. They ran experiments on four diverse datasets, and computed a battery of metrics to consider many possible biases.

While these papers papers largely focused on model prompting techniques and codebook formatting, they did not consider the evolution of the codebook that would normally occur during manual coding. [6] conducts aa meta-study involving qualitative coding, and finds that a significant number of disagreements occur between coders pair-wise. However, the authors argue that this complex interaction eventually benefits the codebook and the quality of coding. One goal of our study is to see how LLMs can interface with this disagreement resolution and codebook evolution process.

An imperative part of qualitative analysis is ensuring reliability through ICA metrics. [4] describes widely used ICA metrics such as percent agreement, Cohen's Kappa, and Krippendorf's alpha. [4] not only defines each of these metrics but also explains the interpretation behind them and what each metric best reveals about coder consistency. This thorough explanation fueled our analysis later in the work, particularly in assessing whether the models could be considered reliable for qualitative coding in the security domain.

3 Methodology Overview

Our experiment consists of two parts: manual coding and LLM coding. The manual coding starts with two human coders developing an initial codebook while working together to code the first batch of tweets. Then, the two human coders iteratively refine the codebook by independently analyzing the next batches of Twitter tweets and modifying the codebook after each batch. The agreement between the coders must reach satisfactory scores in terms of the Inter-Coder Agreement (ICA) metrics detailed in [4]. Then, for each batch after the initial batch (when the initial codebook is created), we have various LLMs perform the coding of the subsequent batch alongside the humans (using the same codebook). We compare the LLM output to the human-agreed results of the batch and compute another set of ICA metrics. The LLM coding was done using two models, Llama 3.2 8B Instruct and GPT-4o mini. We provided each of these models with the various versions of the codebook developed in manual coding to maintain consistency between the code definitions followed by the humans and LLMs. The prompts were constructed to get scores for all three categories so that we can effectively compare with the manual results. We also experimented with modifying the prompt to ask the LLMs to provide justification, as this has been observed to improve coding accuracy, as observed in [3]. Our experimental design is visualized in Figure 1.

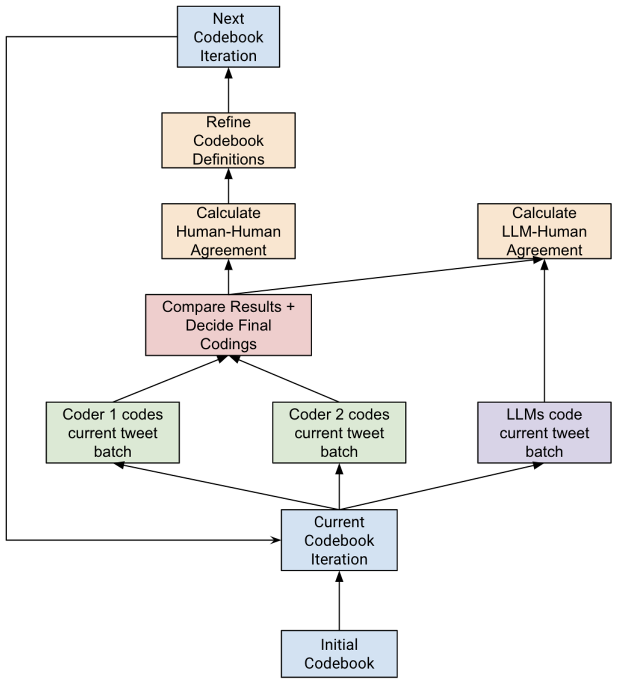
4 Manual Coding

In this section, we describe the process of manually coding a dataset of Twitter tweets to analyze sentiments, discussion types, and topics related to ChatGPT/AI use in security. This process involved the iterative development of a codebook through the collaborative efforts of two human coders, ensuring high inter-coder agreement. The goal was to establish a refined codebook that could serve as a reliable benchmark for evaluating the performance of automated coding using large language models (LLMs).

4.1 Dataset Processing

We examined a dataset of Twitter tweets, retrieved from [5], to qualitatively code the authors' sentiments towards leveraging ChatGPT, or generally AI, in security use cases. The dataset initially contained 704 tweets, each with information about the author, engagement metrics (likes, retweets, replies), and the text scraped from the tweet. Each tweet was included because it included keywords related to the topic, like "ChatGPT", "AI", or "security". However, we quickly

Figure 1: Process diagram showing the steps involved.


Next Codebook Iteration
Refine Codebook Definitions
Calculate Human-Human Agreement
Calculate LLM-Human Agreement
Compare Results + Decide Final Codings
Coder 1 codes current tweet batch
Coder 2 codes current tweet batch
LLMs code current tweet batch
Current Codebook Iteration
Initial Codebook

realized that in these 704 tweets, there were numerous examples of tweets that did not directly relate to the use of AI for security purposes, which prompted a trimming of the dataset. The authors of [5] supported this notion by mentioning how they trimmed the dataset down to 266 tweets, but they did not describe their approach in the paper, so we created our approach with the following guidelines.

The first guideline is an incorrect use of the word security, which was the leading cause for removing a tweet from the dataset. For example, instead of referring to security in a computer context, the tweet could be referring to job security or income security, which are clearly in the wrong context. We also removed tweets that were duplicates because we found several instances of tweets that were being repeatedly sent on the Twitter platform. We also noticed several instances where the tweet was not referencing ChatGPT In the end, our new dataset consisted of 273 tweets that communicated information relevant to understanding user opinions towards ChatGPT's role, and this dataset is what we conducted our manual coding process on.

4.2 Manual Coding Process

We started by dividing the dataset into six different batches, one for each iteration of the codebook development process. The manual coding process begins with two coders working together to create a codebook by analyzing the first batch of tweets. There are three categories that we created codes for which are sentiment, discussion type, and topic. The goal is to determine what the overall tone of the tweet is, what type of format the tweet follows, and what the specific topic of the tweet is. We provided definitions for each of the codes so that when we independently code the next batch of tweets, we know what criteria to follow. These definitions are also necessary for the LLM portion of the experiment because the LLM needs to understand what we meant for each of the codes. This codebook is known as codebook v1 and its resulting codes are as follows.

- **Sentiment:**
  (1) Positive
  (2) Negative
  (3) Neutral
- **Discussion:**
  (1) Advertisement
  (2) Complaint
  (3) Warning
  (4) Joke
  (5) Informative Discourse
  (6) Endorsement
- **Topic:**
  (1) ChatGPT/AI Vulnerability
  (2) ChatGPT/AI Application
  (3) Finding
  (4) Security Solution
  (5) Other

The next step was the two coders needed to independently define the codes for the second batch of tweets following the codes above. Then, the coders discussed their results and tried to find common issues in the labeling of tweets where their codes differed. We ensured that a final code was agreed upon so that we could maintain a set of agreed-upon codes that could be compared to the LLMs' codes. The codebook definitions were modified to better account for these issues in future iterations, and this codebook is known as codebook v2. Then the process repeats for the next four batches of tweets, with the codebook being finalized after the last batch. The codes themselves did not change between the iterations of the codebook, as we were able to fit all of the tweets into one of the codes for all three categories. However, the definitions changed significantly because we encountered common issues that needed to be accounted for, examples of which are shown in Table 1.

During each of the meetings after completing a batch of tweet codings, many conversations about differences, like the three examples above, iteratively the definitions in the codebook over all of the iterations. This resulted in a refined final codebook that we were confident in providing to the LLMs, but we needed to quantitatively ensure that this codebook is valid using ICA metrics.

4.3 Inter-Coder Agreement (ICA) Metrics

To believe that our manually created codebook can be treated as a ground truth to compare the results of the LLM Coding against, we need to ensure that the two coders are in agreement. If our codebook is usable by two humans to produce codes that are mostly similar, then we can expect the codebook to be usable by one human and one LLM to produce similar results as well. [4] outlines three metrics that can be used to determine the strength of our codebook.

The simplest of the metrics is Percent Agreement which is calculated by determining the number of times 2 coders agree on a given code divided by the total number of codes (times 100 to get

Table 1: Examples of tweets on which the two human coders disagreed, prompting re-examination of the appropriate code definitions. One example for each category is presented, along with commentary.

Category of Discrepancy	**Tweet**	**Commentary**
Sentiment	*"Everyone's chatting about #ChatGPT. Here are 11 things it can do for #malware analysts, #security researchers, and #reverse engineers. A thread – 1/13"*	The coding differed because one coder thought this is a neutral tweet that simply states facts about what ChatGPT can do, whereas the other coder thought this is a positive tweet that endorses ChatGPT's abilities in a security context. We ultimately decided that this is a positive sentiment tweet, and the definition of positive was modified to account for instances where ChatGPT is promoted as a tool to aid security.
Discussion	*"4. Amazon, JP Morgan, and many other organizations are restricting employee access to ChatGPT due to security concerns. https://t.co/oh3r2UzfHc"*	The coding differed because one coder thought this tweet is an informative discourse where the author is trying to inform about this topic, but the other coder thought this tweet is an advertisement for the linked article. We decided that this should be classified as an informative discourse because the main point of the tweet is to present a fact rather than try to promote the provided link. We modified the definition of advertisement to not account for tweets that do not explicitly promote the external link.
Topic	*"Working with Kubernetes security? It's so cool to see that @armosec platform users can now utilize ChatGPT to quickly and easily create custom controls"*	The coding differed because one coder thought this tweet was talking about using ChatGPT as a security solution, and the other thought this tweet was talking about using ChatGPT as an application. We decided that the topic of this tweet is a ChatGPT/AI application. The definition of security solution was modified to account for instances where ChatGPT/AI is the primary engine of at least a majority of a security solution. Application was modified to account for instances where ChatGPT/AI is not meant to solve an entire security need, but rather provide a solution to a specific task.

a percentage). As Table 2 shows, the percent agreement numbers improved significantly as the codebook was refined, with agreements in all three categories exceeding 90%. This metric is a good indicator of how often the coders agree, but it struggles to show whether the agreement occurred by chance. Percent agreement does not account for the possibility that coders might randomly assign the same code to an item, which can inflate the perceived level of agreement.

To address this limitation, more robust metrics such as Cohen's Kappa or Krippendorff's Alpha are often used. These metrics adjust for chance agreement and provide a more accurate measure of inter-coder reliability. The scores for both of these metrics range from 0 to 1 with higher numbers being better, so being close to 1 is ideal. Cohen's Kappa compares the observed agreement with the agreement expected by chance, providing a more well-rounded evaluation of coder consistency. Table 2 shows that the kappa scores that we received for the final codebook were strong as sentiment was 0.951, discussion was 0.878, and topic was 0.91. Interestingly, each of these scores was almost double the kappa scores given to the v1 codebook, showing that the definition refinements made a significant difference in agreement.

The last metric, Krippendorff's Alpha, accounts for not only chance agreement, but also the number of codes being used and the prevalence of each code. By accounting for both the number of codes and their prevalence, Krippendorff's Alpha ensures that agreement is not overstated in situations where certain codes dominate. For instance, in datasets where one code is disproportionately frequent, percent agreement might suggest high reliability even if coders are only agreeing on the dominant code by default. Table 2 shows that the alpha scores for the final codebook reached 0.952 for sentiment, 0.879 for discussion, and 0.911 for topic, all of which are strong scores.

Using these three metrics, we have established that the manual codebook we developed leads to strong ICA between two humans, so now we can use it as a ground truth to compare to the results of LLM coding and determine the ICA between a human and an LLM.

5 LLM Coding

After reviewing the relevant literature, we compile a comprehensive collection of parameters that control LLM qualitative coding performance. We fix several parameters to beneficial values based on a literature assessment, and perform experiments with others. We make use of the Llama 3.1 8B Instruct and GPT-4o mini models for LLM coding. We report their performance in comparison with the results from our manual coding as previously described.

5.1 Assorted Considerations

[3] and [2] both extensively discuss best practices for using LLMS in qualitative coding tasks. We review some of these and their motivations here.

Model size/complexity: The size of popular large language models are significant indicators of their comprehension, level of expression, reasoning ability, strength of parsing and interpretation, and ability to follow instructions. [3] conducted experiments with both GPT-3.5 and GPT-4, and found that the former was not able to exhibit human-equivalent performance while the latter was. Unfortunately, we do not have extensive computational resources or funds. For a smaller model, we use Llama 3.2 8B Instruct, which

Table 2: Agreement metrics between the two human evaluators. For each codebook iteration, the metrics are computed over the next batch (~ 40 samples) using that codebook. The metrics computed are Percent Agreement, Cohen's Kappa, and Krippendorff's Alpha, separately for each of the categories: Sentiment, Discussion Type, and Topic.

Codebook	# Tweets	Percent Agreement			Cohen's Kappa			Krippendorff's Alpha		
		Sentiment	Discussion	Topic	Sentiment	Discussion	Topic	Sentiment	Discussion	Topic
v1	62	79%	60%	66%	0.552	0.432	0.483	0.554	0.434	0.485
v2	49	65%	63%	63%	0.387	0.555	0.472	0.392	0.544	0.471
v3	42	88%	79%	88%	0.638	0.704	0.799	0.639	0.704	0.801
v4	43	98%	95%	95%	0.959	0.939	0.933	0.959	0.940	0.934
v5	33	97%	91%	94%	0.951	0.878	0.910	0.952	0.879	0.911

we run locally. For a large model, we OpenAI's developer per-token pricing on GPT-4o mini, which is similar to GPT-4 but smaller (and significantly cheaper).

Dividing categories: Performing coding for multiple categories simultaneously is a fairly complex task. While it could be useful, especially when the categories are related to each other in complex ways, models perform better with simpler, delineated tasks. Thus, we perform the coding of each category separately: each (example, category) pair is fed as a separate prompt.

Codebook formatting: For the chosen category, each label was fed to the model alongside its definition. The human coders took care to be specific and descriptive, but avoid overly-complex word choice and clause order. We also try to be concise, as larger prompts lead to worse performance/generalizeability and longer computation time.

Providing examples: Providing examples of datapoints and human-assigned codes could in theory allow the model to associate codes with characteristics of sample text through its own understanding; this application of in-context learning could make application very easy for the end-user, and allow for more complex latent relationships to develop. However, [3] notes that providing examples greatly increases the prompt size, which negatively has a negative impact on the results; we also experienced this from our ad-hoc testing, so we elected to not pursue this in our experiments.

Model justification: Having the model justify its responses according to the codebook after outputting the correct code is known to improve its accuracy and reliability. This is closely related to the Chain-of-Thought prompting technique explored in [3] and [2]. We run experiments both with and without asking for justification for coding, and observe a slight increase (~ 5% in accuracy on average across all codebooks/categories) for both models. The difference is not significant; for all experimental results discussed, we use the outputs from the prompts with justification.

5.2 LLM Prompt Construction

We use the same prompt format for both of the LLMs that we test. We begin with a brief system prompt, which tells the model that this is a qualitative coding task regarding the security risk of LLMs in programming. Next, we provide the name of the category (Sentiment, Discussion, or Topic), and all of the codes in that category. Each code is given as with its name followed by its definition on the same line; each code is given on a separate line. After providing the codebook, we instruct the LLM to read the tweet that follows, output the correct code, and depending on which experiment we are conducting, either output nothing else or output a justification for its decision on the next line. Finally, we provide the tweet to perform coding on.

We attempted to remain relatively concise and direct in our instructions to the model, in order to avoid confusing it and to decrease computation time.

5.3 Llama Experiments

We conducted our experiments with Llama 3.2 8B Instruct, a recent small instruction-tuned LLM by Meta. We used the 8B model as we did not have the computational resources to run the next-smallest model (70B), and because it would act as a good proxy for a "small" model comparison.

Table 3 contains the experimental results. Note that the agreement metrics are fairly poor, and uniformly lower than the human-human agreement found in Table 2. Most of the percent agreement numbers are below 60%, and almost none of the Cohen's Kappa metrics are above 0.40, which [4] suggests as a lower bound for acceptability. The agreement for Sentiment tends to be higher than for Discussion and Topic. Additionally, all metrics for all categories seem to be increasing with the codebook iteration.

5.4 GPT Experiments

We conducted our experiments with GPT-4o mini using the OpenAI developer API. This model is similar to GPT-4, but much smaller and thereby more cost efficient by an order of magnitude.[1] Additionally, its reasoning ability and long context length make it a preferred option as a large-scale language model for experimentation.

Table 4 contains the experimental results. These results are still worse than the Human-Human agreement, but much better than the Llama-Human agreement. Most of the percent agreement numbers are above 50%, and many of the Cohen's Kappa metrics are above 0.40. As was the case for Llama, the Sentiment agreement metrics are the highest, and, all metrics for all categories are generally increasing with the codebook iteration.

6 Discussion

Both LLMs performed poorly in our experiments; indeed, worse than expected, as compared to the results in [3] and [2]. This is

[1]https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/

Table 3: Agreement metrics between Llama and human evaluation. For each codebook iteration, the metrics are computed over the next batch (~ 40 samples) using that codebook (the same codebook is used by both the model and humans). The metrics computed are Percent Agreement, Cohen's Kappa, and Krippendorff's Alpha, separately for each of the categories: Sentiment, Discussion Type, and Topic.

Codebook	Percent Agreement			Cohen's Kappa			Krippendorff's Alpha		
	Sentiment	Discussion	Topic	Sentiment	Discussion	Topic	Sentiment	Discussion	Topic
v1	59%	42%	50%	0.354	0.240	0.292	0.320	0.164	0.254
v2	60%	45%	49%	0.371	0.272	0.274	0.336	0.237	0.233
v3	59%	45%	48%	0.373	0.271	0.252	0.337	0.242	0.208
v4	57%	46%	50%	0.339	0.299	0.266	0.299	0.267	0.219
v5	64%	46%	50%	0.436	0.289	0.256	0.587	0.361	0.091

Table 4: Agreement metrics between GPT-4o mini and human evaluation. For each codebook iteration, the metrics are computed over the next batch (~ 40 samples) using that codebook (the same codebook is used by both the model and humans). The metrics computed are Percent Agreement, Cohen's Kappa, and Krippendorff's Alpha, separately for each of the categories: Sentiment, Discussion Type, and Topic.

Codebook	Percent Agreement			Cohen's Kappa			Krippendorff's Alpha		
	Sentiment	Discussion	Topic	Sentiment	Discussion	Topic	Sentiment	Discussion	Topic
v1	68%	59%	46%	0.475	0.453	0.292	0.468	0.448	0.262
v2	66%	62%	45%	0.435	0.492	0.282	0.427	0.488	0.253
v3	66%	63%	47%	0.441	0.508	0.293	0.432	0.505	0.274
v4	67%	64%	48%	0.460	0.511	0.314	0.451	0.507	0.295
v5	71%	60%	48%	0.512	0.466	0.313	0.507	0.463	0.297

likely due to us (1) using smaller models than the ones in those papers, (2) having relatively small sample sizes, and (3) not performing as extensive prompt optimizations. Nevertheless, the variation in our results is still insightful; furthermore, our setting is somewhat realistic: most researchers attempting to leverage LLMs for qualitative coding in practice (in general science) would likely prefer to use off-the-shelf models and simple human-interpretable prompt construction.

Our experimental analysis reaffirms some results from prior work. First, the larger model GPT-4o mini performed moderately better than the smaller Llama 3.2 model, showing that the more complex model can better understand the definitions of each class and discriminate the between the text distributions underlying each. Second, the justification-based prompting approach did improve performance. The metrics in Tables 3 and 4 are both using the justification-based prompt; the metrics using the prompt without asking for justification were slightly lower.

The results for the Sentiment category tended to be higher than for Discussion and Topic. We believe this is due to Sentiment having fewer total codes than the others; context length is an important factor in LLM performance. However, Sentiment is arguably the most "abstract" and emotional of the topics, which contradicts intuition. It seems that the context length and still relative simplicity of the codes are the more important factors.

Our departure from prior work was in using the LLM on incremental versions of the codebook as created by the human evaluators. For both LLMs, we see that all metrics tend to increase as the codebook develops, matures, becomes more comprehensive, and is more representative of the wide distribution of samples that fall within each code. This demonstrates that (1) the model is actually using the codebook definitions to perform the sample assessments, and (2) the quality of the codebook is an important factor in the LLMs' reliability for coding. This presents a challenge for practical application: while the LLMs perform decently well when given the final iteration of the codebook, in practice researchers would probably want to use early iterations (as the goal of using LLMs is to save time and not code as many samples).

Our proposed solution to this is LLM-human co-development of a codebook. Human evaluators should manually perform the coding process, produce several iterations of the codebook, and have the LLM perform the same qualitative coding using the same codebooks that they are on the appropriate batches of samples. Once the codebook is sufficiently developed and the LLM using a mature codebook reaches sufficient inter-coder agreement with the human evaluations, the LLM can be used to code the remaining samples. This process is most applicable to large datasets: human evaluators will still need to do significant work to develop a sufficiently mature codebook, but the LLM will still save effort on the remaining samples. Determining the correct thresholds for this condition would require a much more extensive and diverse study.

There are a few minor issues with our study design that could have influenced our results. Our dataset is medium-sized; our results may have been more favorable for the LLMs and our trends may have been more pronounced if we had additional samples and been able to produce more codebook iterations. [4] notes that Cohen's Kappa can be misleading when codes are not evenly applied across the dataset, which is true in our case; this could explain why some Kappa values seem low despite achieving high percent agreement. We hope that our using three metrics simultaneously ameliorates this bias. Finally, we could have conducted more extensive prompting design and hyperparameter tuning to make better use of the models. However, in a real-world setting, we suspect researchers would generally like to use simple prompt constructions on off-the-shelf models for coding; thus, our experimental design is appropriate.

7 Conclusion

In this work, we conducted an experiment to analyze the reliability of LLMs for qualitative coding in security, assessing whether or not they can understand and correctly apply a codebook to security-related data. Based on the coding decisions of Llama 3 and GPT-4o mini when provided with various codebook iterations, we reaffirmed the effectiveness of chain-of-thought prompting and found that large LLMs can be reasonably reliable when given mature, well-defined codebooks. This suggests that while LLMs show promise in applying codebooks to security-related information, they still require a structured starting point to ensure consistent and accurate coding. As such, we recommend an LLM-human co-development approach for the codebook rather than relying on nearly complete automation.

Our work also opens up several promising avenues for future research. One direction is to examine the patterns in incorrect coding decisions made by the LLMs. Comparing these errors with the human coding decisions could provide valuable insight into the nature and extent of each model's shortcomings, offering a clearer understanding of the gaps in their comprehension of the codebook. For example, how "reasonable" are the errors made by the LLM; were the human annotators also in disagreement about those samples, and in the same way? Another avenue is to study the agreement between different LLM models. Finally, though this may not be possible now, eventually we would like to explore allowing LLMs to develop and/or update the codebook independently, rather than simply applying a predefined one; using longer-context technology, eventually they may be able to learn from examples and update their definitions accordingly, as humans do, without strong supervision.

Acknowledgments

Thanks to Prof. Dave Levin for the inspiration, support, feedback, and encouragement throughout this project.

References

[1] Wei Bai, Omer Akgul, and Michelle L. Mazurek. 2019. A Qualitative Investigation of Insecure Code Propagation from Online Forums. en. In *2019 IEEE Cybersecurity Development (SecDev)*. IEEE, Tysons Corner, VA, USA, (Sept. 2019), 34–48. ISBN: 978-1-5386-7289-1. DOI: 10.1109/SecDev.2019.00016.

[2] Robert Chew, John Bollenbacher, Michael Wenger, Jessica Speer, and Annice Kim. 2023. LLM-Assisted Content Analysis: Using Large Language Models to Support Deductive Coding. arXiv:2306.14924 [cs]. (June 2023). DOI: 10.48550/arXiv.2306.14924.

[3] Zackary Okun Dunivin. 2024. Scalable Qualitative Coding with LLMs: Chain-of-Thought Reasoning Matches Human Performance in Some Hermeneutic Tasks. arXiv:2401.15170 [cs]. (Feb. 2024). DOI: 10.48550/arXiv.2401.15170.

[4] Sean N. Halpin. 2024. Inter-Coder Agreement in Qualitative Coding: Considerations for its Use. en. *American Journal of Qualitative Research*, 8, 3, (July 2024), 23–43. DOI: 10.29333/ajqr/14887.

[5] M. Mehdi Kholoosi, M. Ali Babar, and Roland Croft. 2024. A Qualitative Study on Using ChatGPT for Software Security: Perception vs. Practicality. arXiv:2408.00435 [cs] version: 1. (Aug. 2024). DOI: 10.48550/arXiv.2408.00435.

[6] 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. en. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Hamburg Germany, (Apr. 2023), 1–21. ISBN: 978-1-4503-9421-5. DOI: 10.1145/3544548.3580766.

[7] Daniel Votipka, Kelsey R Fulton, James Parker, Matthew Hou, Michelle L Mazurek, and Michael Hicks. [n. d.] Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It. en.

A Codebook

We include the final version of the codebook in Tables 5, 6, and 7.

Received 11 December 2024

Table 5: Final codebook: definitions for codes in the Sentiment category

Code	Definition
Positive	If the tweet describes the influence of ChatGPT or some other AI-based tool as beneficial towards the overall state of security or security in a particular case, then it should be classified as positive. If the tweet points out benefits as well as drawbacks on this matter, then the tweet should be considered positive if it portrays the benefits as outweighing the drawbacks or if it emphasizes the benefits over the drawbacks. Note that security here is referring to security as understood in computer science as well as the ability to cause/prevent harm to people.
Negative	The tweet refers to the use of ChatGPT or AI in a bleak manner (e.g. complaints or warnings) in the context of security. In this case, the dangers posed could outweigh the benefits provided by using these tools. Note that security here is referring to security as understood in computer science as well as the ability to cause/prevent harm to people.
Neutral	The tweet does not provide an opinionated view on using ChatGPT or AI for security purposes. This means that the intended audience of the tweet would not be swayed one way or another after reading the tweet about using ChatGPT for security. Examples of this include facts, research findings, and insights. If the tweet does not pertain to ChatGPT or AI in light of security (as understood in computer science or as it relates to misinformation/ cheating/ deception), then it should be considered neutral. Note that, if a tweet is simply the title of a resource such as a paper, video, etc, even if said title is positive or negative about ChatGPT/AI in relation to security, the tweet should still be classified as neutral.

Table 6: Final codebook: definitions for codes in the Discussion category

Code	Definition
Advertisement	The tweet's main goal is to promote links, papers, podcasts, products, websites, or videos, especially in the area of security though they do not have to be. For example, the tweet could highlight that a panel discussion took place about how ChatGPT will influence the security landscape and include a link to a recording of it. Note that a tweet should not fall into the advertisement discussion type category if the resource is not the primary focus of the tweet but rather is a supporting piece of evidence to the main idea of the tweet.
Complaint	The tweet criticizes aspects of using ChatGPT in relation to security (or whatever the subject of the tweet is) and could contain examples of problems caused. The focus of the tweet is more about the annoyance and dissatisfaction caused by the subject of the tweet than the various problems or risks posed by it. A complaint often mentions a specific scenario in which the user is clearly dissatisfied, regardless of whether or not the claims are warranted. The tweet may include other links or resources if they are meant to support the complaint being made.
Warning	The tweet contains various problems or risks posed by ChatGPT in the context of security (or whatever the subject of the tweet is). The focus of the tweet is more about the aforementioned topic than the annoyance and dissatisfaction caused by the subject of the tweet. Unlike a complaint, a warning must inform users about a prediction that could prove detrimental. The tweet may include other links or resources if they are meant to support the warning being made.
Joke	The tweet is humorous in nature, for example, through the use of sarcasm or funny examples. The tweet may provide actual information about ChatGPT, but if it is intended to create laughs, then it should still be classified as a joke. The tweet may include other links or resources if they are meant to support the joke being made.
Informative Discourse	The tweet contains undeniably true information, or it provides some information that makes a topic easier to understand. The overall result of the tweet is enlightening the user, particularly about ChatGPT as it pertains to security or as it relates to its behavior when performing a specific task. The tweet may include other links, resources, or ChatGPT-generated quotes if they are meant to support the discourse being made.
Endorsement	The tweet contains praise about ChatGPT as it pertains to security (or whatever the subject of the tweet is), or it endorses its usage in a particular area of interest. For example, a tweet whose main focus is on recommending ChatGPT as a tool to look over code for vulnerabilities should be classified as an endorsement. The tweet may include other links or resources if they are meant to support the endorsement being made.

Table 7: Final codebook: definitions for codes in the Topic category

Code	**Definition**
ChatGPT/AI Vulnerability	The tweet contains information about ways, within the context of security, that ChatGPT/AI can be exploited, either maliciously or accidentally, can result in output that can be exploited, or fail to properly do the task assigned to it. Specifically, the code refers to ways in which the average user can have negative interactions with the tools, or ways in which attackers can take advantage of the tools' weaknesses. For example, if a tweet discusses how ChatGPT's code often contains security flaws, then it should be classified as a ChatGPT/AI vulnerability. If the tweet itself is not descriptive but it includes a link or resource that describes a ChatGPT/AI vulnerability, then the tweet's topic should still be classified as ChatGPT/AI vulnerability.
ChatGPT/AI Application	The tweet describes ways in which ChatGPT/AI can be leveraged to perform security-related tasks. This task is not about solving a particular security need or problem but rather doing a specific function needed by user/s. If the tweet itself is not descriptive but it includes a link or resource that describes how ChatGPT/AI could be used in such a manner, then the tweet's topic should still be classified as ChatGPT/AI application.
Finding	The tweet's primary focus is on research regarding the influence and/or implications of AI on the overall state of security or some subfield of it, or it encourages such research to be performed. This is primarily in the form of descriptions/conclusions drawn from research papers, but there can be descriptions/conclusions from other sources like articles. If the tweet itself is not descriptive but it includes a link or resource that describes a finding pertaining to AI, then the tweet's topic should still be classified as a finding. Note that if the main purpose of the information stated from a paper, article, or some other resource is to highlight a vulnerability of, application of, or solution involving ChatGPT/AI, then the tweet should be classified as a ChatGPT/AI Vulnerability, ChatGPT/AI Application, or Security Solution respectively.
Security Solution	The tweet describes ways in which ChatGPT/AI is the primary engine for a large portion or entirety of a security solution. This differs from a ChatGPT/AI application in the sense that a security solution involves a whole idea or methodology to address a security need or problem (e.g. ensuring mobile app security) and whereas a ChatGPT/AI application involves ChatGPT/AI being employed to address an individual task (e.g. checking for a bug in a piece of code) that could be a constituent part of a larger security need or problem. If the tweet itself is not descriptive but it includes a link or resource that describes a security solution, then the tweet's topic should still be classified as a security solution.
Other	There are cases in which the tweet does not focus on the correct meaning of security because we are looking only for instances of security as understood in computer science or as it relates to misinformation/cheating/deception. If the tweet is not related to using ChatGPT or AI in security, it should be classified as other. Tweets that are about ChatGPT's output in areas outside of security and tweets whose main subject is the economic viability of cryptocurrencies and/or NFTs should also be classified as other.