Aeonium: Visual Analytics to Support Collaborative Qualitative Coding

Margaret Drouhard, Nan-Chen Chen, Jina Suh, Rafal Kocielnik, Vanessa Peña-Araya, Keting Cen, Xiangyi Zheng, and Cecilia R. Aragon

Abstract— Qualitative coding offers the potential to obtain deep insights into social media, but the technique can be inconsistent and hard to scale. Researchers using qualitative coding impose structure on unstructured data through "codes" that represent categories for analysis. Our visual analytics interface, Aeonium, supports human insight in collaborative coding through visual overviews of codes assigned by multiple researchers and distributions of important keywords and codes. The underlying machine learning model highlights ambiguity and inconsistency. Our goal was not to reduce qualitative coding to a machine-solvable problem, but rather to bolster human understanding gained from coding and reinterpreting the data collaboratively. We conducted an experimental study with 39 participants who coded tweets using our interface. In addition to increased understanding of the topic, participants reported that Aeonium's collaborative coding functionality helped them reflect on their own interpretations. Feedback from participants demonstrates that visual analytics can help facilitate rich qualitative analysis and suggests design implications for future exploration.

Index Terms—Visual analytics, qualitative coding, collaborative coding, social media research, social science, machine learning

1 Introduction

Large-scale computational approaches to social media data collection and analysis offer researchers broad overviews of vast populations, but further probing is often necessary to expose nuanced or deeper relationships and trends in complex, ambiguous human-generated data. Qualitative approaches to data analysis give researchers the opportunity to obtain deep insights but often in a limited context or scope. When researchers qualitatively code data, the humans become the instruments of analysis, drawing from domain knowledge and subtle contextual information to apply expressive "codes" as a means to systematize and interpret unstructured text. Machine learning (ML) techniques may provide support for scaling up to larger datasets, but off-the-shelf ML tools are often inadequate for rich exploratory analysis. Moreover, the goals of qualitative coding often conflict with assumptions of objective ground truth data on which ML classifiers generally rely. Instead, qualitative analysis often necessitates multiple interpretations and re-interpretations of data in order to draw out complicated and sometimes contradictory themes. Rather than attempting only to scale qualitative analysis through machine classification, our coding interface, *Aeonium*, facilitates human insight in collaborative coding through visual overviews of codes assigned by multiple researchers and distributions of codes associated with important keywords extracted from the data. We also support evolving code definitions as exploration of the data shifts code boundaries and reveals new trends. To our knowledge, it is the first visual analytics tool or qualitative coding interface to prioritize drawing attention to potentially ambiguous data for collaborating qualitative coders.

The primary contributions of this work are threefold: (1) Aeonium identifies ambiguous data in order to focus human attention on data that are unclear or merit additional scrutiny; (2) it supports the negotiation of codes and understanding of the context for coding decisions among multiple collaborators; and (3) it supports reflection, reinterpretation, and insight gain in qualitative coding. We conducted formative studies with qualitative researchers to establish key design objectives, and we validated the design decisions and performance of Aeonium through an experimental study and expert review. Aeonium is a novel approach that demonstrates the power of visual analytic techniques to support in-depth qualitative coding by directing human attention toward data that warrant the most interpretation and analysis.

- *Margaret Drouhard, Nan-Chen Chen, Jina Suh, Rafal Kocielnik, Keting Cen, Xiangyi Zheng, and Cecilia R. Aragon are with the University of Washington. Corresponding Author Email: mdrouhard@acm.org.*
- *Vanessa Peña-Araya is with the University of Chile.*

2 Background

2.1 Overview of Qualitative Coding Techniques

Qualitative coding is a fundamental technique in numerous fields including communication, psychology, and other social sciences, and it has proven useful–in combination with quantitative methods–in social media research as a means of synthesizing broad overviews and deep insights about research subjects [1, 10, 23, 32]. Researchers use qualitative coding to impose order and help derive meaning from complex unstructured data generated by humans. The process usually involves arranging data systematically by segregating, grouping, and linking it in order to facilitate sensemaking and explanation of phenomena. Qualitative coding is often used to search for patterns by arranging similarly coded data into categories based on commonly shared characteristics [30]. Coding is critical in qualitative analysis because the data has no intrinsic organizational structure by which to explain the phenomena under study. Researchers must therefore create a structure and impose it on the data to determine how best to facilitate interpretation for their purposes [21]. Therefore, the value of qualitative analysis largely depends on the quality of coding [32].

Although variations of coding methods abound, most of them are rooted in *Grounded Theory*. Fundamentally, grounded theory requires that a theory be developed inductively from data rather than through existing theoretical frameworks [5]. Typically the process consists of the following steps: *open coding*, *focused coding*, *axial coding*, *selective coding* [5, 11, 14, 29, 33]. Often these steps are not carried out sequentially, since insights or realizations of connections can occur any time during the process [8]. *Memo-writing* is another analytical method that can be used throughout the process of grounded theory exploration to help articulate implicit or unstated meanings and to construct theoretical categories [8]. Aeonium primarily supports focused coding, the component of the process in which codes that have been articulated during open coding are applied to new or revisited data [33].

Qualitative coding may be compared to *ground truth labeling* in ML. However, their differences can be characterized in three parts. Firstly, the goals of the two tasks are different. Ground truth labeling presumes that "correct" labels exist for every data point and the task is to identify them, whereas qualitative analysis makes use of subjective codes in the process of meaning-making in a particular context. Secondly, ground

truth categories are typically well-defined before labeling begins, but grounded theory precludes a priori codes, since the process should be based on the data. Finally, quantitative metrics for data in qualitative analysis are not inherently meaningful. In other words, a code that appears a thousand times may hold similar weight in analysis to other codes that appear only a few times. In contrast, the number of data instances for ground truth labels in ML profoundly impacts learning task performance, since ML is statistical in nature. This is not to say that ML places more inherent value on frequent labels, but rather that categories with only few instances may not be learnable by machines, regardless of their importance for a particular analysis context.

Procedures for qualitative coding vary from project to project. In *solo coding* projects, one researcher codes the entirety of the data independently. Solo coding is common for qualitative studies [15], and usually the researcher who collects data also codes it. In other projects, teams of researchers code a dataset collaboratively, which is referred to as *team coding* or *collaborative coding*. In this case, team members need to coordinate their coding tasks and make sure they are coding harmoniously [30]. Aeonium's current design supports focused coding in collaborative settings, addressing the fundamental challenge of coding consistency given constrained codes. Future work should address qualitative coding needs in other contexts.

2.2 Software for Qualitative Coding

Numerous free or commercial tools have been developed to support qualitative coding. Common tools like ATLAS.ti, QDA Miner, and NVivo support coding both text and multimedia data, and they also provide a set of data organization and analysis techniques. Other tools, such as Dedoose, MAXQDA, and SaturateApp, provide more functionality for team coding, but it can be challenging for coders to resolve inconsistencies within these interfaces. Some of these tools emphasize visualization or visual interface in their design. For example, Dedoose includes visualizations of code distribution and correlation, but it primarily facilitates quantitative analysis of qualitative data. Many other tools, like f4analyse, Annotations, and Quirkos, use colors to represent codes. Recently, Blascheck et al. proposed to use visual support for coding interview transcripts and present activity patterns and selected information through visualization [4]. Like these tools, Aeonium uses colors to facilitate coding visually. However, Aeonium's design also explicitly facilitates evolving code definitions and includes a specialized interface for code review in order to better support iteration in coding. Some other tools offer computational tools for inter-coder reliability to identify inconsistency, but Aeonium presents visual overviews for classes of disagreement. It also includes mechanisms for feedback, and its ML model can identify uncoded data for which coders are likely to disagree.

2.3 Machine Learning to Assist in Qualitative Coding

Some prior works showcase progress toward facilitation of qualitative coding through fully- or semi-automatic methods. For instance, Crowston et al. compared the use of automated coding between human-defined rules and machine-learned models [13]. Their results demonstrated that both methods are promising, but the human-defined rules outperformed the ML-based approaches. Yan et al. implemented a system using ML and natural language processing techniques to generate initial codes and have human coders correct false-negative instances [36]. Extending this idea, McCracken et al. built an active learning system that supported an interactive loop between machine annotation and human correction [25]. These automated approaches demonstrate potential, but have not yet achieved sufficient accuracy in label prediction to replace human-defined methods or earn humans' full trust. Rather than leveraging ML to predict codes directly, our system adopts ML as a way to surface ambiguous data, preserving the human-centered nature of qualitative analysis. We do not claim that our ML models outperform other methods in terms of standard ML performance metrics (e.g., ML accuracy) because qualitative analysis and machine learning have different objectives. Previous literature suggests that collaboration between social scientists and computer scientists is required to scale up qualitative data analysis [35]. Thus, a primary focus of our work is to better understand the domain and explore the design space of facilitating qualitative coding through ML-backed visual analytics.

2.4 Visual Support for Text Classification and Feature Engineering

While we emphasized the distinctions between qualitative analysis and ground truth labeling in ML, some ML systems are designed specifically to facilitate human interaction with ML, which can support insight gain. These applications for ML and visual analytics are more closely aligned with our design objectives in Aeonium, but they primarily support human-computer cooperation to improve ML systems. In contrast, we focus on enhancing the human components of analysis with support from ML. Some existing work includes visual support for labeling ground truth or iterating text features. For example, Kulesza et al. proposed a method called *structured labeling* to deal with the issue of *concept evolution*, a phenomenon in which "people change their standard of categorizing things on the way they are labeling" [18]. The visual interface for structured labeling enables the categorization of items into major concept categories (yes-no-could be) and for the creation of subcategories and tags for fine-grained concepts. Kulesza et al. also designed a system called *AutoCoder* to support explanatory debugging of text classifiers [19]. They applied the system in an automatic qualitative coding context, and provided an interface for users to suggest misclassified text. Outside the realm of ground truth labeling, Brooks et al. demonstrated that visual support can be useful in feature ideation [6]. They built a system called *FeatureInsight*, which shows how each feature relates to classifier errors. Also in the sphere of feature engineering, Cheng and Bernstein built a crowd-sourced feature ideation system called *Flock* with an interface to show how user-generated features related to categories [9]. Their study demonstrated that combining system- and user-generated features can increase both ML accuracy and interpretability of ML tools. These techniques have proven useful for drawing out insights (e.g., meaningful features), facilitating labeling, and understanding the data. We incorporate ideas from these prior works, but we aim primarily to support the human qualitative analysis process.

2.5 Inter-rater Reliability Metric for Qualitative Coding

Inter-rater reliability (IRR) is a widely used measure for evaluating consistency between coders. There are various ways to calculate IRR, and one of the most common methods is *Cohen's Kappa* [12]. Although IRR is not universally considered applicable to qualitative coding [3,27], since it is rooted in quantitative analysis, some existing literature suggests that IRR can be used to improve coding quality. For example, Garrison et al. proposed using IRR to measure improvement in consistent application of codes [16]. Hruschka et al. use IRR to determine whether codebooks need to be further modified [17]. Moreover, Burla et al. pointed out that codes with low agreement rate may need clarification [7]. In our study, we use IRR (specifically Cohen's Kappa) as a metric for agreement or consistent application of codes between two coders in order to measure improvement in consistency under various conditions.

3 Formative Studies & Design Objectives

In the early stages of designing Aeonium, we conducted semi-structured interviews with five qualitative researchers to establish and validate our guiding design principles. One interviewee was a faculty member, and the remaining four were PhD students. Two of these researchers have a background in Human-Computer Interaction, while the other three have backgrounds in Communication. All of the researchers interviewed use qualitative coding extensively in their research. Each interview lasted approximately one hour, and members of our research team coded and analyzed the data from interview notes and audio recordings. Several of the authors have experience with qualitative coding, but we sought insight from researchers who rely on qualitative analysis as either a primary or exclusive method for their research. We asked interviewees to reflect on their experiences with qualitative coding, and in particular,

to identify key challenges encountered when coding collaboratively. We also asked our interviewees to describe how they evaluate the quality of coding and to what extent inconsistency between different coders and ambiguity in the data impacted their analysis. Time-permitting, we asked interviewees about the software tools they use for coding and their perspectives on the application of ML in qualitative analysis. From our own experiences with qualitative coding and through these interviews, we distilled three principle design objectives for Aeonium:

Design Objective 1 *Draw out ambiguous subsets of data–or data where coders are likely to code inconsistently–in order to focus human attention on challenging areas of data.*

Design Objective 2 *Keep code definitions prominent and facilitate clarification of the explicit or implicit basis for coding decisions.*

Design Objective 3 *Support the iterative, reflective work of qualitative analysis.*

We elaborate on each design principle in the following sections.

3.1 Design Objective 1: Draw Out Ambiguity

Given that the subject of most qualitative coding is human behavior, ambiguity is an inherent characteristic of this data. Interviewees emphasized that coding is always subjective, so rather than valuing "accuracy" of codes, it may be more useful to focus on coding in a manner that helps explain phenomena meaningfully. Some of the qualitative researchers in our interviews were worried that the application of ML techniques to qualitative analysis may inhibit coders' ability to see connections in data because ML models cannot capture implicit knowledge "at the fringes of observation." They also expressed concern that ML might oversimplify analysis by implying that an objective label exists for what is inherently a subjective interpretation. One interviewee stated, "If someone or some program says there is one answer [in qualitative analysis], that's the wrong answer." However, almost all the researchers expressed interest in ML's potential to draw out inconsistencies in coding or identify ambiguous, difficult-to-code data.

Beyond the challenges of particular data, factors such as mood changes, attention, and memory can impact consistency between different coders, or *inter-rater reliability*, as well as consistency of the same coder at different points in time. Saldana urges qualitative coders to consider throughout the coding process how their personalities, perspectives, and subjective decisions impact the analysis [30]. Human fallibility inevitably leads to some inconsistency , which can delay or change the overall analysis of the data. ML systems are not susceptible to such factors , but their application in facilitating qualitative coding is not straightforward. Complex features that humans can intuitively identify and interpret are more valuable for rich insights , but their complexity often results in an insufficient amount of labeled data for ML algorithms. More importantly, most ML tools function as black boxes, so their results are not readily interpretable or controllable by humans. Inversely, humans may not be able to express their ideas to the ML model in an effective way, or conflicts between coders may impact the performance of the machine learner.

Aeonium allows coders to explicitly flag ambiguous data through the interface, and it also infers ambiguity from disagreement between collaborators. The combination of explicit ambiguity labeling and implicit recognition of ambiguity through disagreement supports qualitative coders' analysis of challenging data. Aeonium helps draw out ambiguous data that would otherwise require more extensive exploration and negotiation by humans. Rather than try to predict how humans will code, Aeonium helps focus coder attention on data that may require more careful consideration.

3.2 Design Objective 2: Highlight Code Definitions and Context for Decisions

Along with ambiguous data, researchers identified complex codebooks (sets of codes) and vague boundaries between codes as challenges that should be addressed. As they process "chunks" of data, coders are generally working with multiple codes at the same time, even if the codes are mutually exclusive or only one may apply to a given chunk. It can be difficult to remember the intricacies of each code or the boundaries between codes, so highly visible code definitions may support more efficient coding. Researchers in our interviews also pointed out that different perspectives of multiple coders yield different insights. Particularly for ambiguous data, these different perspectives may lead to inconsistent coding or decisions that are unclear to other coders. Coders have to explain the basis of their coding decisions either through explicit content–such as keywords–or implicit background knowledge. Similar to how externalization can increase awareness and lead to new insights [24], identifying features that were the basis of coding decisions, whether explicit or implicit, may help coders iterate on coding decisions and find new insights in the data.

This design objective also requires that coders have the ability to elaborate on code definitions or provide context to explain the basis for their coding decisions over time. The potential for code definitions to evolve presents an additional challenge in applying ML to qualitative coding: conflicting assumptions and objectives for qualitative codes as opposed to ML labels. In fact, the goals of qualitative coding may often conflict with the presumption of a singular, "correct" label. We heard multiple researchers explain that coding always involves some analytical "messiness." "[Codes] are never defined enough." As one interviewee described it, coding is not just about figuring out what's happening, but rather about figuring out how to tell the story in a way that helps people understand it meaningfully. Qualitative coders recognize the subjectivity of decision boundaries of codes and the exact circumstances in which they apply, so they have to negotiate the boundaries together.

Emphasizing code definitions and the basis for coding decisions also helps fulfill design objective 1. In some cases, the definition of codes can lead to inconsistent application. When definitions are vague, they may be applied differently by different coders or even be completely misunderstood. Supporting evolution of definitions over time ensures that codes adapt to changing data and facilitates coders' discoveries of new dimensions in the concept space. Limited context is a frequent source of inconsistency in coding. The scope of a topic may be unclear, or it may need to be adjusted over time. In other cases, only partial data is available, so coders must either guess the missing context or ignore the data points. Coders can alleviate the risks of inconsistency by explaining the reasoning behind their coding decisions. Humans often know or presume context that is external to the data but may affect analysis. This external context or background knowledge varies from person to person and changes over time based on experiences. Tools that enable coders to share the basis for their coding decisions can help a team take advantage of the wealth of perspectives and knowledge among researchers to promote deeper understanding of complex data.

Aeonium supports both the evolution of code definitions and improved awareness of definition boundaries with prominent, mutable code definitions in the coding interface. Coders may also select words in a given data instance and add them as keywords aligned with particular codes to explain the basis for the coding decisions. Keyword highlighting can mitigate the risks of inattentive coding decisions. These functionalities also support deeper analysis of emerging concepts from the data by encouraging coders to explicitly consider code bounds and context for coding choices.

3.3 Design Objective 3: Support Iteration

Qualitative researchers in our interviews emphasized that the coding process is iterative, and that tools need to support shifting understandings, broadening of codes, and re-examination of old data and code decisions. Often throughout the coding process, interviewees informed us, coders review previously coded data as emerging information casts it in a new light. Alternatively, conflicting interpretations from various coders, which differ due to distinct backgrounds or experiences, may affect review and reinterpretation. According to our interviewees, these reflections and reinterpretations frequently serve to help coders clarify code definitions and boundaries, which Aeonium supports through modifiable definitions. Interviewees reported frustration with tools that

did not adequately facilitate exploration of relationships between previously coded and new data, as well as with those that did not support evolving codes.

Researchers we interviewed also described the different roles iteration may play in outcomes of qualitative analysis depending on overall objectives. In some cases, researchers may be interested in precisely defining the themes observed in a particular context, so evolving code definitions as described in design objective 2 can become the priority. In other instances, the objective may be to ensure that researchers have reached "saturation" in their understanding of the context. In other words, the priority is on collecting more data and repeatedly reviewing it until no new concepts emerge. Regardless of the objective for a particular project, iteration and reflection on the data and codes are critical, so they must be supported to facilitate qualitative coding appropriately.

Aeonium facilitates various levels of iteration and deliberation on the data. The coding interface helps link the discovery of new concepts and ideas to previously coded data through code definitions, example data, and highlighted keywords. The review interface supports resolution of disagreements, evaluation of ambiguity, and evolving code definitions. Shifting between the interfaces is also closely aligned with the qualitative coding practice of alternating between coding and reflecting.

3.4 Simple Visual Encodings to Support Design Objectives

Our choice of relatively simple visual encodings in Aeonium was deliberate and grounded in the needs of qualitative analysis. Qualitative coding is a very text-heavy task. From our formative studies and our own experiences in qualitative coding, we realized that showing raw text would be desirable. When researchers cannot easily access the text in a coding tool, they often prefer to "regress" to plain text spreadsheets. In designing Aeonium, we tried to ensure that researchers could interact with text easily, which led to simple visual encodings. Many existing text visualizations with more complex visual encodings focus on word counts (e.g., Wordle [34]), co-occurrence frequency networks, or topics (e.g., [22]). As we mentioned in Section 2.1, quantitative metrics are not necessarily valuable in qualitative analysis. Thus, we did not consider those complex visualization encoding methods in Aeonium's design, but rather made efforts to support qualitative analysis of data, and the resulting visual encodings were simple. Prior work in visualization for decision support [2] has shown simpler visualizations to be more effective when users need to focus attention on information not represented in the visualization. Qualitative researchers constantly rely on background contextual knowledge and information "at the fringes of observation," so simple visualizations are well suited to provide valuable context without distracting attention from other areas of focus.

4 Aeonium System

Aeonium supports qualitative focused coding, facilitating the review of coded data between coding partners. In its current iteration, codes are defined by "master coders" who lead the qualitative coding and are usually researchers with domain expertise. As we describe in Section 7, more flexible additions and merging of codes will be supported in our next iteration. However, we presume that coders who wish to define new codes or hierarchies of coding concepts will have deep familiarity with the data and research goals. For the purposes of our initial design and validation studies, master coders were two graduate students with training in qualitative analysis methods and experienced in qualitatively coding social media data. The master coders also collaborated with two undergraduate student members of the research team in the initial coding and defining of code boundaries. Aeonium may be used for qualitative coding of any short text documents, but we have conducted our initial studies with a tweet dataset.

Aeonium trains a support vector machine (SVM) classifier for each user based on their labels (i.e., coded tweets) and features (or keywords). Keyword features consist of system keywords (i.e., bag-of-words unigram features) extracted from the data set and user-defined keywords (i.e., one or more selected words that are not necessarily contiguous) extracted from coded tweets that are relevant to explicitly explaining the code decision. The feature value is computed by matching the words in the tweet to the keywords. The classifiers are used only for the purpose of determining which tweets to label based on predicted ambiguity. Aeonium has two interfaces: one for coding data, shown in Figure 1, and another for reviewing codes, keywords, and definitions, shown in Figure 2. The coding interface supports efficient coding decisions by locating the color-coded definitions centrally on the screen and by showing keywords (if present within a given tweet) highlighted in the color used to represent the associated code. The coding interface also affords more in-depth analysis by showing extended definitions and visual overviews of keywords for a selected code in the lower panel. The review interface facilitates negotiation of assigned codes between a pair of coders, as well as reinterpretation of data given evolving code definitions and new insights from data.

In the following subsections, we describe our system in terms of its alignment to our design objectives.

4.1 ML Model and Flags to Draw Out Ambiguity

As described in design objective 1, qualitative researchers are interested in ways to draw out ambiguous data or inconsistent codes in order to better shape the definitions and negotiate code boundaries. Aeonium supports this aim in two ways: the ML model predicts tweets for which partners may disagree, and users may explicitly flag ambiguous tweets during coding. Showing tweets that are likely to be ambiguous or inconsistent between coders draws users' attention to these tweets and encourages a dialogue between coders to improve mutual understanding and consistent coding. The ambiguity flag is also useful for exploring sources of confusion or uncertainty in coding.

From our preliminary interviews with qualitative researchers, we determined that when coders initially disagree on appropriate codes or initially identify multiple mutually exclusive codes that may be applicable, these data require additional attention. For our primary evaluation of Aeonium, the metric we used to predict "ambiguous" data was disagreement between partners on prior coding decisions. Since partners may change their codes through the feedback dropdown menu in the review interface, Aeonium's ML model predicted ambiguity based on tweets for which partners continued to disagree after reviewing. After a pair completes the review stage, the system will train a classifier for each user based on their existing coded tweets after review. Then for the remaining tweets that have not been coded, the system predicts labels for both partners. For a stage of coding focused on ambiguous data, the system will sort uncoded tweets based on level of predicted disagreement (i.e., tweets for which partners are predicted to disagree with the highest confidence), and the dataset for the ambiguous stage will include tweets for which partners are most most strongly predicted to apply inconsistent codes. In order to better surface ambiguity in the future, Aeonium's ML models will soon incorporate as features the explicit flagging of ambiguous tweets and the "Unsure" responses from the disagreement feedback dropdown menu, which indicate uncertainty about the decision.

4.2 Pairwise Comparison and Feedback on Disagreement

In addition to drawing out ambiguity, Aeonium facilitates reviewing and resolving disagreements. In the review interface, the code comparison table summarizes how users and their partners agree or disagree with each other. For instance, in Figure 2.7, the interface shows that among the tweets coded with the "Support " label by the current user, his/her partner has disagreed twice, with one tweet coded as "Rejection" and another as "Uncodable." By clicking on each row, users can filter to tweets that belong to the selected code combination and focus on analyzing, providing feedback, and resolving inconsistency. Currently, for each tweet on which partners' codes disagreed, Aeonium provides three response options for the disagreement: "My code is correct", "My partner's code is correct", or "Unsure." "Unsure" may indicate that either code might apply or that there is insufficient context to make a coding judgment. When a user indicates that his/her partner's original code is correct, the system will ask that user whether or not to change the assigned code to match the partner's code. This feedback loop can help coders become more consistent with each other over time, and

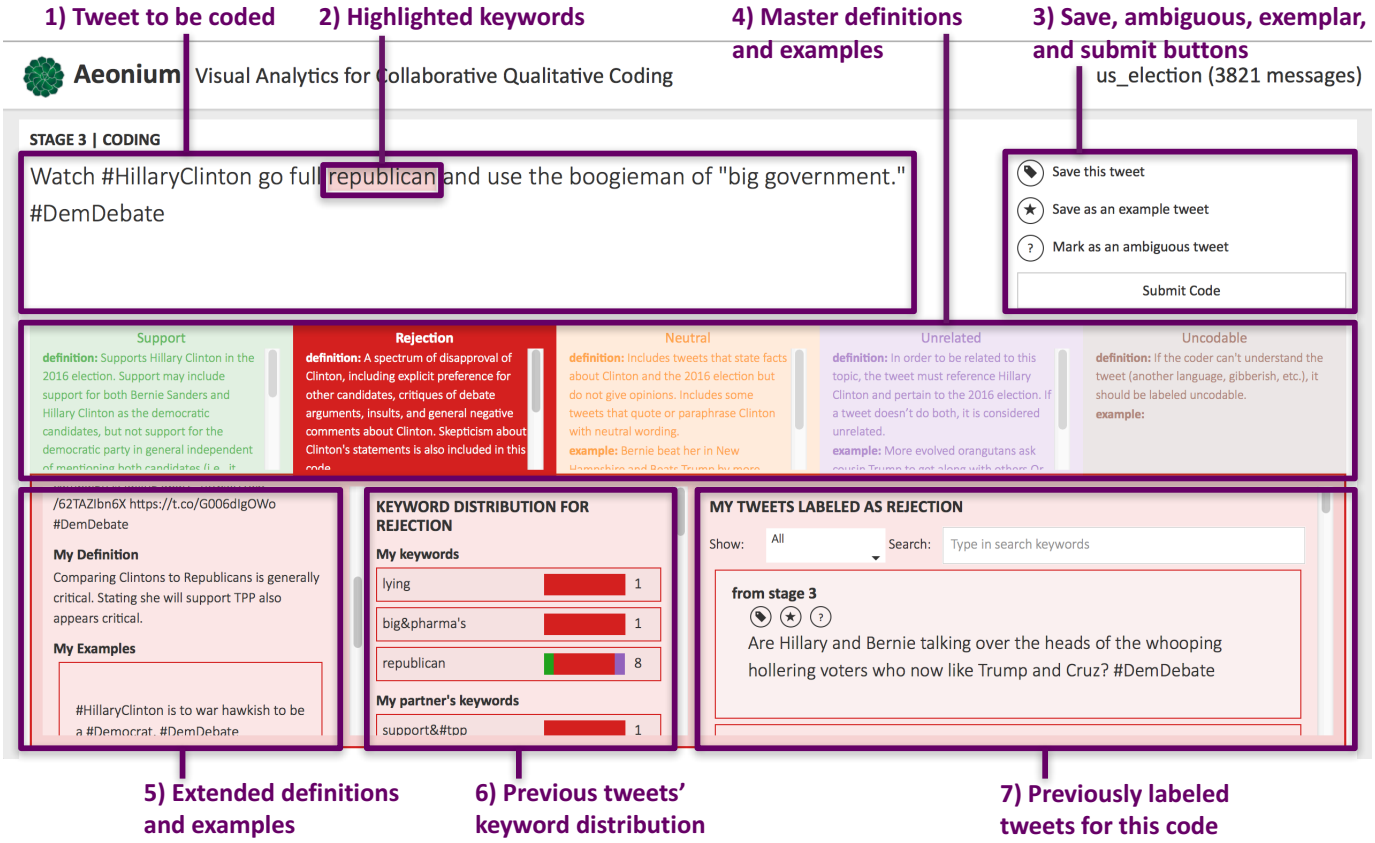

Fig. 1. Aeonium's coding interface consists of two panels: The top panel primarily supports the coding task and displays a single tweet to code (1) with keywords that the model recognizes highlighted (2), buttons to flag that tweet as ambiguous, save it, or make it an exemplar tweet for the selected code (3), and a row of color-coded buttons representing codes in the coding schema with the code definition and an exemplar tweet from master coders (4). The bottom panel enhances the coding task by providing additional information such as the code definitions and examples from the master, user, and partner (5) to highlight discrepancies, distribution of coded tweets for the system, user, and partner keywords (6), helping to illustrate keyword relevance. The bottom panel also includes a list of the user's previously coded tweets (7) for additional context.

it implicitly supports the iterative nature of qualitative coding (design objective 3). Additionally, since disagreement can be an indication of ambiguity, having pairwise comparison also contributes to our design objective 1 to draw out ambiguous data.

4.3 Keyword Highlighting to Provide Context and Support Iteration

Keyword highlighting works in three ways in Aeonium. During coding, if a user-extracted keyword exists in the tweet text, it will be highlighted in the color of its associated code, drawing attention to key information and mitigating the risks of inattentive coding. Secondly, when reviewing coded tweets, a user can add new keywords from a tweet's text to explain the context for choosing a particular code. Finally, after adding a new keyword, users can see how this keyword is distributed over codes. If a keyword is not very predictive or informative (i.e., it is not particularly well aligned with a specific code), users will be able to recognize that it may not be a useful keyword to explain a coding decision. As outlined in describing design objective 2, researchers want to understand context around coding decisions, and highlighting keywords identifies some of the explicit context. In addition, when keywords are shown highlighted according to code color in the coding interface, users can assess how well keywords align with their assigned codes. Since the coding process is iterative as explained in design objective 3, users can reflect on the association between keywords and codes. For example, when a user sees a keyword highlighted in green as "Support" in the text, but he/she thinks the tweet should be coded "Rejection," the user can reevaluate how well the keyword aligns with the "Support" code.

4.4 Code Definitions for Context Awareness and Iterative Negotiation of Code boundaries

Aeonium's coding interface shows codes in a given codebook, along with their definitions and examples just below the tweet. By placing these definitions centrally within interface, users always have at least peripheral awareness of the definitions. This functionality serves design objective 2. Researchers in our interviews indicated that codes tend have vague boundaries, so centralizing code definitions can help coders make more efficient, better informed coding decisions. Additionally, since each user likely has a slightly different understanding of code meanings, explicit support for negotiation can improve consistency and facilitate iteration (design objective 3). The review interface in Aeonium lets users provide expanded definitions based on how code meanings are evolving, and this information is displayed to collaborators in both the coding and review interfaces, in which users see their partners' definitions and their own extended definitions in addition to the master coder definitions.

5 Validation

5.1 Validation Methods

To validate the contributions presented in this work we relied on Munzner's nested model [26]. The core components of our validation methods include: interviews with qualitative researchers (Section 3); a study of ambiguity and coding confidence with Amazon Mechanical Turk master workers (Section 5.4); an experimental Aeonium evaluation study with undergraduate and graduate students (Section 5.5); and

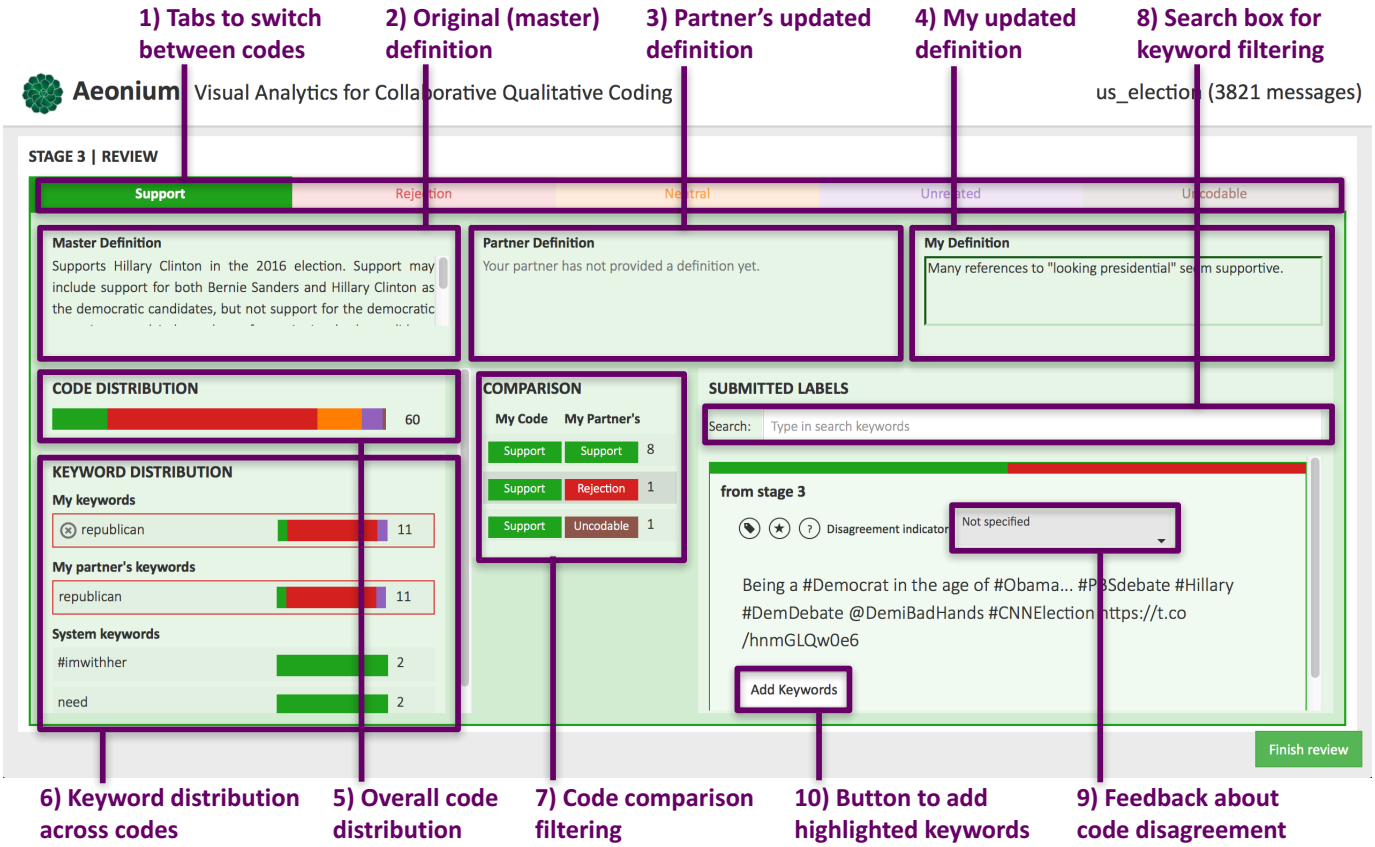

Fig. 2. The review interface supports understanding and negotiation of code boundaries and discrepancies between coders. Its tab-based view allows for switching between the detail view of each code (1). Each code tab provides comparison and edit capability of code definitions (2-4), overview of the data distribution (5), and distribution of coded tweets for keywords extracted (6), comparison of codes (7) to summarize agreement or disagreement between coders, and the list of coded tweets for analysis. Tweets can be filtered with search terms (8) or by selecting a keyword (from 6) or a code pair (from 7). Users can reevaluate codes and provide feedback about a disagreement through a dropdown menu (9) or provide more context for their decisions by adding keywords extracted directly from the tweets (10).

an expert review of Aeonium with qualitative researchers (Section 5.6).

Our interviews with qualitative researchers were the primary means of validation for domain characterization, the outermost level of Munzner's model. These interviews, in addition to helping us shape our design objectives, supported our articulation of the qualitative coding tasks that may benefit from visual analytics tools. Some of the tasks identified were exploring data, identifying areas of disagreement between coders, and refining coding context and definitions. Also at the domain characterization level, the Mechanical Turk (MT) study of ambiguity and coding confidence served to validate our claims that ambiguous data merits additional human attention. For the second layer in the nested model, we utilized the experimental evaluation study and the expert review to validate Aeonium's support for sorting, filtering, predicting ambiguous data, and other operation-level ("generic") tasks. We used both quantitative and qualitative results from the experimental study, along with qualitative feedback from the expert review, to validate our visual encodings and interaction techniques in the third level of Munzner's model. Techniques evaluated include: color highlighting of keywords, overviews of code distributions, and functionalities for negotiation between collaborators.

In addition to Munzner's model, our studies align with the scenario of Evaluating Communication through Visualization (CTV) suggested by Lam et al. [20]. Both of the studies applied quantitative and qualitative methods, and we garnered additional qualitative feedback about Aeonium from the expert review. Given that the controlled experiments may not necessarily reflect coding practices well, we placed higher value on qualitative analysis of insights users gained through coding with Aeonium. As a component of our evaluation, we qualitatively coded insights participants reported gaining, as prior work has proposed for assessing visualizations [28].

5.2 Research Questions and Hypotheses

For our evaluation studies, we first explored the effect ambiguity has on users' perceptions of coding quality and metrics for coding consistency. Thus we first investigated the following research question:

RQ 1 ***What are the differences in coding confidence and inter-rater reliability (IRR) when coding ambiguous versus unambiguous data?***

We expect that coders labeling ambiguous data would be less certain of their coding decisions. We expect that if coders are less confident in applying codes to particular subsets of data that these subsets require additional attention from coders and probably negotiation among multiple coders. Specifically, we hypothesize:

Hypothesis 1a ***Confidence in Coding with Ambiguous Data:*** *Participants labeling ambiguous data will have lower confidence in their labeling accuracy than those labeling unambiguous data.*

In order to evaluate how well the ML model in Aeonium identifies ambiguous data, we measure IRR. We expect that coding ambiguous data will lead to lower IRR between coders than will coding random data. Therefore to evaluate how well our ML model could identify data for which coders were likely to disagree we hypothesize that:

Hypothesis 1b ***Identification of Ambiguous Data:*** *Coding ML-predicted ambiguous data will lead to lower IRR between coders than coding randomly selected data.*

We were also interested in investigating how ambiguity affects the process of formulating code definitions, as well as how it impacts the updates coders make to the keywords aligned with codes. Accordingly, we articulated the following research question:

RQ 2 ***How does ambiguity affect the negotiation and coding consistency between collaborators?***

We expected that coders would negotiate more around code definitions and coding decisions for ambiguous data. Updating code definitions indicates recognition that evolving definitions are necessary as new data are explored. Therefore we hypothesize:

Hypothesis 2a ***Definition Updates:*** *Participants will make more refinements to code definitions when data is ambiguous than when data is random.*

We operationalize "more refinements" as updating and saving code definitions more. Negotiation around coding decisions in Aeonium is accomplished through modifying keywords. Adding and removing keywords from explicit tweet content is one way for participants to explain the basis of their decisions to collaborators. We therefore hypothesize that:

Hypothesis 2b ***Keyword Updates:*** *Participants will update more keywords when data is ambiguous than when data is random.*

We operationalize "update more keywords" as either adding or removing keywords. We further expect that increased negotiation will improve IRR, as coders will be able to better align their coding strategies. As explained in hypotheses 2a and 2b, we anticipate that participants who never coded ambiguous data will negotiate less around code definitions and the context for coding decisions (through keywords), since they are not coding targeted ambiguous data. We expect that the more communication participants have around code definitions and basis for coding decisions, the more their IRR will improve. We therefore hypothesize:

Hypothesis 2c ***Negotiation of Ambiguity Improves Inter-rater Reliability:*** *Participants who coded ambiguous data will show greater improvement in IRR than those who did not code any ambiguous data.*

We further expect that the earlier the participants negotiate code boundaries and context for coding decisions, the more improvement they will have in IRR later. We therefore hypothesize that:

Hypothesis 2d ***Earlier Negotiation of Ambiguity Improves Inter-rater Reliability More:*** *The participants who negotiate code boundaries earlier (encounter ambiguous data earlier) will show greater improvement in IRR than those who negotiate code boundaries later (encounter ambiguous data later).*

5.3 Data and Codes

Aeonium's visual coding interface supports labeling of tweets with codes defined by master coders. For the validation studies, tweets were collected from twitter's streaming API for a period of a week in February 2016 and filtered for references to the 2016 election. A subset of election-related tweets, those which reference Hillary Clinton and the Democratic Debate, were selected as the dataset for our study. Master coders identified the following five mutually exclusive codes:

- **Support:** Explicit or implicit support for Hillary Clinton (regardless of opinions about other candidates)
- **Rejection:** Explicit or implicit rejection, criticism, or skepticism about Hillary Clinton, regardless of opinions about others.
- **Neutral:** Statement of fact or quotation from a candidate without explicit or implicit expression of opinion.
- **Unrelated:** Statements that do not reference Hillary Clinton or do not pertain to the 2016 U.S. election (Note: Although all tweets in the dataset referenced Clinton and the election in the tweet text or metadata, not all of the tweets actually pertained to this topic, so the unrelated code was used to identify such tweets.)
- **Uncodable:** Statements in which meaning cannot be deciphered (e.g., in a language other than English, gibberish, etc.)

5.4 Study of Ambiguity and Coding Confidence

Our first study was designed to further validate our claim at the domain characterization level that ambiguous data merits extra human attention.

5.4.1 Pre-test of Ambiguity

Prior to the study itself, we conducted a pre-study to confirm distinctions between ambiguous and unambiguous tweets. We pre-tested 116 tweets from the full dataset. Approximately half of the pre-tested tweets had been labeled ambiguous by master coders, and the distribution of tweets among the five mutually exclusive codes described in Section 5.3 was proportional to the distribution coders had seen in the data analyzed up to that point. Two Amazon Mechanical Turk (MT) master workers were asked to categorize the tweets as "ambiguous" if multiple of the five mutually exclusive codes could potentially apply. The workers were to label the tweets as "unambiguous" and apply one of the five codes if they believed that the statement clearly belonged in one code category.

Of the data categorized in this pre-test, master workers agreed on categories (and sub-categories if applicable) for only 44 items, underscoring the highly subjective nature of qualitative coding. For the ambiguity study, we selected 20 tweets determined to be ambiguous and 20 determined to be unambiguous by the pre-test. These tweets became the dataset for the study of ambiguity. They were distributed primarily among support and rejection codes, but some neutral and unrelated tweets were also included.

5.4.2 Confidence in Coding with Ambiguous Data

This study was designed to validate our initial understanding that coding ambiguous data requires more human attention than unambiguous data, in order to support our design choices for drawing out ambiguity in design objective 1. As in the pre-test, participants in the coding ambiguous data study were MT master workers. The study was between two groups, each of which contained 15 participants. Participants in the ambiguous condition coded tweets determined to be ambiguous in the pre-test, and participants in the unambiguous condition coded tweets determined to be unambiguous in the pre-test. All participants completed a survey in which they labeled 20 tweets. Participants were provided the definitions for the codes given in Section 5.3, and they were required to provide a code for each tweet *without* the option to label tweets as ambiguous. After labeling the tweets in the survey, participants were asked to rate their confidence in the codes they had applied to the tweets on a scale of 1 to 5.

5.5 Aeonium Evaluation Study

We designed the Aeonium evaluation study to validate our operation-level abstractions for domain tasks (including sorting, filtering, predicting ambiguity) and to evaluate the visual encoding and interaction techniques (such as color keyword highlighting, overviews of code distributions, and mechanisms for negotiation between collaborating coders). We collected quantitative and qualitative metrics from this study, and our qualitative results particularly highlight Aeonium's contributions toward facilitating qualitative coding.

5.5.1 Participants

39 participants for this study were recruited from undergraduate and graduate students. Of those who chose to disclose demographic data, 14 identified as female and 21 as male. Eight were adults under age 20, 23 were in the age range 21-30, and 4 in the range 31-40. The highest completed educational degrees reported were: high school diploma

Table 1. Table of Experimental Conditions for Aeonium Evaluation Study. Random tweets may be any tweets in the dataset not already labeled by a coding pair. Ambiguous tweets are those for which the ML model predicts partners will disagree based on their previously coded tweets.

	Stage 1	Stage 2	Stage 3	Stage 4
Condition 1	Random	Random	Random	Random
Condition 2	Random	Ambiguous	Random	Random
Condition 3	Random	Random	Ambiguous	Random

(11), associate's degree (3), bachelor's degree (8), or master's degree (10). Participants also reported their prior experience with qualitative coding through a self-reported Likert scale measure in a survey after our experiment. In a range of 0 to 5 with 0 representing no experience with qualitative coding and 5 representing regular use of qualitative coding in their research, 14 marked 0 experience, while the remainder indicated the following levels of qualitative coding experience: 1 (6), 2 (6), 3 (8), 4 (5), and 5 (2).

5.5.2 Study Design

In this study, participants coded tweets using Aeonium in collaboration with remote partners. Remote partners and master coders were members of our research team who were familiar with the dataset and utilized collaborative functionalities of the tool to communicate insights with participant partners. We adopted a between-group study design in which participants were assigned randomly to one of three conditions. Within-group measures comparing ambiguous stages to random stages were also collected. Our experiment consisted of four stages and usually lasted around 90 minutes. For each condition, the first stage was a baseline stage in which data was randomly selected from the dataset, while the remaining stages could include either random or ML-predicted ambiguous data.

Aeonium functionalities and code definitions were explained in a tutorial at the beginning of the study, and definitions were available throughout the experiment via the definition panel. When explaining code definitions, researchers also informed participants explicitly that they would have the opportunity to elaborate on code definitions if they thought it appropriate. Exemplar "gold standard" tweets were provided as guidance for application of codes. This process simulates qualitative focused coding, in which codes have already been defined. In our version of focused coding, code definitions could evolve as coders explored the data, but the codes themselves could not be modified, and new codes could not be added. Coders could communicate emerging definitions of codes explicitly in the code definition panel. For this experiment, codes were mutually exclusive, so coders could only label a tweet with one code.

5.5.3 Procedure

Each stage is composed of two sub-stages: first coding (using the coding interface shown in Figure 1) and then review (interface shown in Figure 2). In the coding stage, participants received 20 tweets to label with the specified codes. Participants additionally had the ability to explicitly mark tweets as ambiguous or as exemplar tweets for the code. After both partners finished coding the set of tweets, they began the review sub-stage, in which they could view the coded tweets with both partners' labels. Participants and remote partners could review codes and provide feedback about a disagreement, including highlighting keywords to explain a coding decision and explicitly updating the code definitions to include emerging phenomena. Researchers conducting the study instructed participants explicitly "not to worry about accuracy" in coding, but rather to simply choose the code they thought most appropriate. For additional guidance, we provided for reference (at any point) exemplar "gold standard" tweets as defined by master coders. To view other tweets in the dataset, participants could filter tweets by code (or filter for unlabeled tweets), and they could search by keyword.

The four stages of our experiment were distinct in only a few ways. Stage 1 (S1) was a baseline stage, establishing how participants coded in general, as well as how consistent they were with gold standard and partner codes. Additionally, we gathered baseline measures from the review portion of S1. Stages 2 (S2), 3 (S3) & 4 (S4) were identical to each other except in the data. The conditions for the data available in each stage are shown in Table 1. For the ambiguous stage, pairs coded and reviewed tweets for which the ML model predicts that the pair will disagree, based on disagreement in the stage immediately prior to the ambiguous stage. In stages S2, S3, and S4, users also had visual overviews of code distributions, keywords, and definitions, as described in Section 4.

Following the coding and review tasks, participants completed a brief survey including demographic information and Likert score measures of experience with qualitative coding and of knowledge about Hillary Clinton and the 2016 election before and after the coding exercise. The survey also included open-ended questions about insights gained through the process and tool functionalities that supported insight gain. Three of the authors qualitatively coded responses to these open-ended questions. Average pairwise inter-rater agreement between the three coders ranged from 82.02 to 91.65%, and we resolved coding disagreements using the majority opinion of all three.

5.6 Expert Evaluation

In order to get more targeted feedback from qualitative researchers, we ran an abbreviated version of the Aeonium evaluation study with four graduate-level qualitative researchers who had not participated in our formative studies. This evaluation included a brief initial interview about the researcher's collaborative coding process and challenges encountered. Then we proceeded through two stages of coding tweets. As with the Aeonium evaluation study, stage 1 was a baseline and learning period for the tool, and participants coded random tweets. For stage 2 in the expert evaluation, participants coded tweets predicted to be ambiguous. In the last stage, we asked participants to think aloud while coding and reviewing. Afterwards, we asked participants to provide feedback about qualitative coding challenges, Aeonium functionalities, and the application of ML in qualitative analysis. Following the study, we open-coded our notes and audio from the expert evaluation, capturing feedback from the participants about general issues with qualitative coding, helpful or valuable functionalities of Aeonium, limitations and suggestions for future work, and perspectives on the utility of ML support for qualitative coding.

5.7 Potential Threats to Validity

For our MT study, there were potential threats to internal validity such as selection bias since we did not have background information for the MT workers. Similarly, since the results were drawn from a small group of MT workers, they may not be generalizable to other populations. As for the potential threats to the internal validity of Aeonium's evaluation study, first, since participants were paired with a random master coder, there may be differences between master coders that lead to issues such as maturation effect and instrumentation errors. In addition, since participants were told they would be testing a tool, this knowledge might have affected their behavior, perhaps encouraging them to use all the functionality we provided. In terms of threats to external validity, since our participants were mostly students at a university, our results might be biased towards a limited population.

6 Results

6.1 Drawing Out Ambiguous Data

One of our highest priorities in designing Aeonium was to draw out ambiguous data as described in design objective 1 in order to focus human resources on the most challenging work. To justify this design priority, we first had to validate our premise that ambiguous data requires additional human effort. Results from multiple components of our study supported this proposition. In our study of ambiguity with MT workers, participants' responses to the follow-up question indicated that coders had lower confidence in their coding decisions for ambiguous data. Levene's test showed that the equality of variances held between two groups ($F(1,13) = 0.65, p = 0.43$). A one-tailed

independent-samples t-test of coding confidence indicated a significant difference ($t = -2.977, p < 0.01$) between the group coding ambiguous data (M = 3.93, SD = 0.7) and the group coding unambiguous data (M = 4.6, SD = 0.51), so **hypothesis 1a** was supported. Qualitative analysis reinforced this finding as well. In the feedback from the Aeonium evaluation study, a majority of participants cited ambiguity or uncertainty as a meaningful challenge in their coding experience. Feedback from qualitative researchers in the pre-interviews and expert evaluation confirmed that ambiguous data and lack of context are significant issues in the coding process in general.

Disagreement between collaborating coders is a strong indicator of ambiguous data. In **hypothesis 1b**, we proposed that disagreement in the ML-predicted ambiguous data stage could be used to evaluate how well Aeonium identified ambiguous data points. We performed one-tailed paired-samples t-test on the difference in IRR for participants in both conditions 2 and 3 between the random and ambiguous stages 2 and 3, ordered according to Table 1. As we expected, IRR was lower for the ambiguous stage ($M = 0.73, SD = 0.16$) than the random stage ($M = 0.77, SD = 0.12$), though the difference was not statistically significant ($t = -1.115, p = 0.138$). Cohen's d showed effect size to be low ($d = 0.255$), which may indicate insufficient power. We anticipate that expanding the metric of ambiguity to include user-flagged ambiguous tweets and feedback about partner disagreement could further widen the gap between ambiguous and random data. The model may also require more training data from participants, since in this case predictions were based on disagreement during a single stage (20 tweets). Both experts and participants in the Aeonium evaluation study stated that they value being able to label ambiguity explicitly, and experts said they appreciated the ML predictions of ambiguity. Given the broad acknowledgement that ambiguous data is particularly difficult to code qualitatively, we expected that coders would make more effort to negotiate code definitions and explain the basis for their coding decisions when they coded ambiguous data. Results for definition updating in the Aeonium base evaluation study trended in the direction of **hypothesis 2a**, but differences between the number of definition refinements in random ($M = 0.33, SD = 0.83$) versus ambiguous data stages ($M = 0.17, SD = 0.58$) were not statistically significant ($t = 1.112, p = 0.136$) in a paired-samples t-test. Counter to our proposition in **hypothesis 2b**, participants updated keywords more during the random data stage ($M = 1.71, SD = 2.67$) than the ambiguous stage ($M = 1.67, SD = 2.33$). Although the difference was not significant under an one-tailed paired-samples t-test ($t = 0.103, p = 0.46$), this result suggests that ambiguity may be related to missing context from the tweet, requiring coders to use implicit knowledge to make a coding decision. We describe Aeonium's support for elaborating on the context of coding decisions in the following section.

6.2 Support for Negotiating of Code Definitions and Context around Coding Decisions

In service of design objectives 2 and 3, Aeonium supports negotiation of code definitions and communication around the context for coding decisions. In particular, the prominent, modifiable code definitions, color highlighting of user-specified keywords, and flags for ambiguity serve to facilitate this communication. Participants cited the keyword highlighting as one of the most valuable functionalities of Aeonium. Many participants described adding keywords to clarify their reasoning around coding decisions and code boundaries.

Experts and participants in the evaluation study reported that they valued the view of the partner's updated definitions as it helped them better understand the code boundaries, but they were reluctant to modify the definitions themselves. Participants stated that they preferred to defer to the definitions provided by the master coders because they felt that the master coders were more familiar with the data and project goals. Experts explained, however, that they would want to evolve code definitions over time for their own research. In order to better evaluate the support for updating code definitions, in future work we will conduct longitudinal studies with qualitative researchers coding their own data.

Regardless of the dataset, we expected that ambiguous data would elicit more explanation and negotiation of code boundaries and decisions. In **hypothesis 2c**, we reasoned that ambiguous data would encourage the negotiation process since it necessitates added human effort. We hypothesized that the increased communication would result in greater improvements of inter-rater reliability for participants who coded ambiguous data than for those who did not. We further proposed in **hypothesis 2d** that the earlier the coders initiated the discussion around code definitions and context, the more they could improve IRR. Levene's Test for homogeneity of variance indicated no significant difference between groups ($F = 0.166, p = 0.847$). One-way ANOVA analysis showed a significant difference ($F = 3.794, p < 0.05, \eta^2 = 0.174$) between the conditions, but the post-hoc two-tailed t-tests with Bonferroni correction ($p < 0.167$ to be significant) did not show a significant difference ($t_{12} = 0.305, p_{12} = 0.763; t_{13} = 2.365, p_{13} = 0.026; t_{23} = 2.098, p_{23} = 0.047$). The observed power from power analysis is 0.654, indicating that our results may be impacted by inadequate sample size. As we discussed in Section 6.1, it is also possible that collaborators may need to code more data together to train a satisfactory ML model for ambiguity.

In the context of negotiation, another key finding that emerged from qualitative feedback is that coders wished to provide explanations for coding decisions based on implicit knowledge. Three of the expert evaluators indicated that they need functionality to provide explanations about implicit context to adequately explain the basis of coding decisions. All of the experts agreed that missing context is a fundamental source of ambiguity, so explanations of implicit knowledge or decisions based on what *is not* in the text may be particularly meaningful for ambiguous data. Future iterations of Aeonium will provide extended functionality for explaining coding decisions in order to capture implicit context.

6.3 Support for Reflection, Reinterpretation, and Insight Gain

Aeonium's design prioritizes one of the most profound goals in qualitative coding: facilitating the expansion of understanding and reflection. As outlined in our description of design objective 3, qualitative coding is intended to iteratively build up insight about the coding context. Different Aeonium functionalities supported insight gain for different participants. Many participants identified code and keyword distribution visual overviews as helping them get a bigger picture view of the data. Expert evaluators cited central views of code definitions and overviews of user and system keywords as useful for supporting their understanding of the codes. Most participants relied heavily on the visual overview and filtering using the code comparison table to resolve inconsistencies, with many stating that these functionalities stimulated reflection and deeper thought about the data and the codes. Experts also indicated that the highlighting of inconsistency through the code comparison table and feedback to partners could help resolve "information asymmetry" in order to build knowledge and reach consensus among collaborators with different perspectives and backgrounds.

7 Limitations and Future Work

In this work, we have focused on ambiguity in coding informal text such as tweets, and we used ML prediction of ambiguity to direct coders to data that may require more human attention. Although the current interface design only supports a limited number of codes and pairs of coders, the design simplicity helped us evaluate the concept of drawing out ambiguous data in qualitative coding. As we evolve Aeonium, we are exploring ways to extend the interface to include additional codes and ways to support an arbitrary number of coders who code and review results collaboratively. We are also working to conduct a longitudinal study with qualitative researchers using their own data to explore more realistic usage scenarios, especially collaboration cases such as sharing the work. The current Aeonium system supports iteration through functionalities for updating definitions, identifying keywords, and providing feedback on disagreement. In addition to these mechanisms for negotiation, we are adding functionality for

per-instance comments to allow explanation of implicit context and to support in-place memoing. We are also working to support more flexible codebook iterations (e.g., adding or merging codes; hierarchical codes). Finally, though the current ML methods supporting Aeonium are preliminary and results may be limited due to the small number of coded items, we view our planned longitudinal study as an opportunity to incorporate other methods of exploration such as active learning [31]. We are also working to improve our ML models by exploring different featuring and sampling methods for drawing out potentially ambiguous data points. Our tool may be extended to support coding interview scripts or other types of textual media, but since our focus is ambiguity, informal short text with inherently limited context is of particular interest to us.

8 Conclusion

We designed Aeonium to support the collaborative, iterative process of qualitative coding by highlighting ambiguity and facilitating the evolution of code definitions, as well as supporting explanations of the basis for coding decisions. Through interviews with qualitative researchers and three studies, we validated and demonstrated how Aeonium fulfills our three design objectives. Participants in our evaluation studies reported that the collaborative coding functionalities for communicating between partners gave them new insight about the topic and helped them reflect on their own interpretations of the tweets. Unprompted, two of the expert evaluators stated that they would like to use Aeonium in their own research.

We believe that this approach for ML-supported visual analytics can facilitate qualitative coding by improving consistency and accelerating exploration of large text datasets. Our interface provides clean, intuitive interactions that allow coders to enhance their analysis by identifying keywords they consider significant when applying codes and broadening code definitions as new understandings emerge. This work demonstrates the potential of visual analytics techniques to support drawing coders' attention to issues that most depend on human insight and interpretation.

Acknowledgments

The authors wish to thank the researchers and students who participated in our evaluations. This work was supported in part by Washington Research Foundation Fund for Innovation in Data-Intensive Discovery and by the Moore/Sloan Data Science Environments Project at the University of Washington.

References

[1] C. Andrews, E. Fichet, Y. Ding, E. S. Spiro, and K. Starbird. Keeping up with the tweet-dashians: The impact of officialaccounts on online rumoring. 2016.

[2] C. R. Aragon and M. A. Hearst. Improving Aviation Safety with Information Visualization: A Flight Simulation Study, 2005.

[3] D. Armstrong, A. Gosling, J. Weinman, and T. Marteau. The Place of Inter-rater Reliability in Qualitative Research: An Empirical Study. *Sociology*, 31(3):597–606, 1997.

[4] T. Blascheck, F. Beck, S. Baltes, T. Ertl, and D. Weiskopf. Visual analysis and coding of data-rich user behavior. *IEEE TVCG*, 2017.

[5] S. Borgatti. Introduction to Grounded Theory. *http://www.analytictech.com/mb870/introtogt.htm*.

[6] M. Brooks, S. Amershi, B. Lee, S. M. Drucker, A. Kapoor, and P. Simard. Featureinsight: Visual support for error-driven feature ideation in text classification, 2015.

[7] L. Burla, B. Knierim, J. Barth, K. Liewald, M. Duetz, and T. Abel. From text to codings: intercoder reliability assessment in qualitative content analysis. *Nursing research*, 57(2):113–117, 2008.

[8] K. Charmaz and J. Smith. Grounded theory. *Qualitative psychology: A practical guide to research methods*, pp. 81–110, 2003.

[9] J. Cheng and M. S. Bernstein. Flock: Hybrid crowd-machine learning classifiers, 2015.

[10] C. Chew and G. Eysenbach. Pandemics in the age of twitter: content analysis of tweets during the 2009 h1n1 outbreak. *PloS one*, 5(11):e14118, 2010.

[11] A. Coffey and P. Atkinson. *Making sense of qualitative data: complementary research strategies*. Sage Publications, Inc, 1996.

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

[13] K. Crowston, X. Liu, and E. E. Allen. Machine Learning and Rule-based Automated Coding of Qualitative Data. *Proc. of the American Society for Information Science and Technology*, 47(1):1–2, 2010.

[14] T. D. Gallicano. An example of how to perform open coding, axial coding and selective coding. *https://prpost.wordpress.com/2013/07/22/an-example-of-how-to-perform-open-coding-axial-coding-and-selective-coding/*, July 2013.

[15] S. C. Galman. *Shane, the lone ethnographer: a beginner's guide to ethnography*. Rowman Altamira, 2007.

[16] D. R. Garrison, M. Cleveland-Innes, M. Koole, and J. Kappelman. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education*, 9(1):1–8, 2006.

[17] D. J. Hruschka, D. Schwartz, D. C. S. John, E. Picone-Decaro, R. A. Jenkins, and J. W. Carey. Reliability in coding open-ended data: Lessons learned from hiv behavioral research. *Field Methods*, 16(3):307–331, 2004.

[18] T. Kulesza, S. Amershi, R. Caruana, D. Fisher, and D. Charles. Structured labeling for facilitating concept evolution in machine learning. In *Proc. of CHI*, pp. 3075–3084. ACM, 2014.

[19] T. Kulesza, S. Stumpf, M. Burnett, W.-K. Wong, Y. Riche, T. Moore, I. Oberst, A. Shinsel, and K. McIntosh. Explanatory debugging: Supporting end-user debugging of machine-learned programs, 2010.

[20] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale. Seven guiding scenarios for information visualization evaluation. 2011.

[21] M. D. LeCompte. Analyzing qualitative data. *Theory into practice*, 39(3):146–154, 2000.

[22] H. Lee, J. Kihm, J. Choo, J. Stasko, and H. Park. ivisclustering: An interactive visual document clustering via topic modeling. *Computer Graphics Forum*, 31(3pt3):1155–1164, 2012.

[23] J. Maddock, K. Starbird, H. J. Al-Hassani, D. E. Sandoval, M. Orand, and R. M. Mason. Characterizing online rumoring behavior using multidimensional signatures, 2015.

[24] N. Mahyar and M. Tory. Supporting communication and coordination in collaborative sensemaking. *IEEE TVCG*, 20(12):1633–1642, 2014.

[25] N. McCracken, J. L. S. Yan, and K. Crowston. Design of an Active Learning System with Human Correction for Content Analysis. *Sponsor: Idibon*, p. 59, 2014.

[26] T. Munzner. A Nested Model for Visualization Design and Validation. *IEEE TVCG*, 15(6):921–928, 2009.

[27] K. A. Neuendorf. *The content analysis guidebook*. Sage, 2002.

[28] C. North. Toward measuring visualization insight. *Computer Graphics and Applications, IEEE*, 26(3):6–9, 2006.

[29] N. Ralph, M. Birks, and Y. Chapman. The methodological dynamism of grounded theory. *International Journal of Qualitative Methods*, 14(4):1609406915611576, 2015.

[30] J. Saldaña. *The Coding Manual for Qualitative Researchers*. Sage, 2015.

[31] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11.

[32] A. L. Strauss. *Qualitative Analysis for Social Scientists*. Cambridge University Press, 1987.

[33] R. Tesch. *Qualitative research: Analysis types and software*. Routledge, 2013.

[34] F. Viegas, M. Wattenberg, and J. Feinberg. Participatory visualization with wordle. *IEEE TVCG*, 15(6):1137–1144, Nov 2009.

[35] G. Wiedemann. Opening Up to Big Data: Computer-assisted Analysis of Textual Data in Social Sciences. *Historical Social Research/Historische Sozialforschung*, pp. 332–357, 2013.

[36] J. L. S. Yan, N. McCracken, S. Zhou, and K. Crowston. Optimizing Features in Active Machine Learning for Complex Qualitative Content Analysis. *ACL 2014*, p. 44, 2014.