# Cody: An AI-Based System to Semi-Automate Coding for Qualitative Research

Tim Rietz
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
tim.rietz@kit.edu

Alexander Maedche
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
alexander.maedche@kit.edu

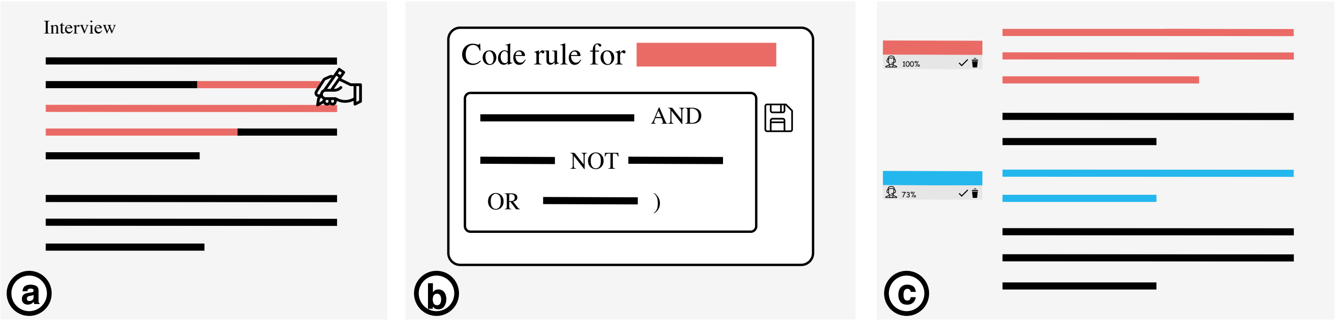


**Figure 1: Cody used to extend qualitative coding to unseen data. (a) The user makes an annotation in a text document. (b) The user revises a rule suggestion to define the created code. (c) Cody searches text for other occurrences (red), and trains a supervised machine learning model to extend manual coding to seen and unseen data (blue).**

## ABSTRACT

Qualitative research can produce a rich understanding of a phenomenon but requires an essential and strenuous data annotation process known as coding. Coding can be repetitive and time-consuming, particularly for large datasets. Existing AI-based approaches for partially automating coding, like supervised machine learning (ML) or explicit knowledge represented in code rules, require high technical literacy and lack transparency. Further, little is known about the interaction of researchers with AI-based coding assistance. We introduce Cody, an AI-based system that semi-automates coding through code rules and supervised ML. Cody supports researchers with interactively (re)defining code rules and uses ML to extend coding to unseen data. In two studies with qualitative researchers, we found that (1) code rules provide structure and transparency, (2) explanations are commonly desired but rarely used, (3) suggestions benefit coding quality rather than coding speed, increasing the intercoder reliability, calculated with Krippendorff's Alpha, from 0.085 (MAXQDA) to 0.33 (Cody).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CHI '21, May 8–13, 2021, Yokohama, Japan*
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8096-6/21/05…$15.00
https://doi.org/10.1145/3411764.3445591

## CCS CONCEPTS

• **Computing methodologies** → *Supervised learning by classification*; • **Human-centered computing** → *User interface design*; • **Information systems** → **Clustering and classification**; *Structured text search*.

## KEYWORDS

Qualitative research; Qualitative coding; Rule-based coding; Supervised machine learning; User-centered design; Artifact design

**ACM Reference Format:**
Tim Rietz and Alexander Maedche. 2021. Cody: An AI-Based System to Semi-Automate Coding for Qualitative Research. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3411764.3445591

## 1 INTRODUCTION

Qualitative research is valued not only in the human-computer interaction (HCI) community to produce detailed descriptions and rounded understandings, allowing researchers to answer *what is*, *how*, and *why* questions [38]. It relies heavily on primary data in the form of unstructured text, transcribed from sources such as recordings from interviews or focus groups. The annotation of transcripts with descriptive or inferential labels, referred to as *coding*, is an essential step for making sense of the text to drive the development of concepts or theory [7]. Within qualitative data analysis (QDA), coding is iterative. It goes from identifying initial categories in data during first-pass coding to assigning and revising labels to identify categories and themes. While qualitative researchers cherish good coding as a mix of science and art [38], detailed and extensive texts make coding highly time-consuming and error-prone. Much of the process can be painstaking and repetitive [46]. This challenge is further aggravated with access to more massive datasets with new possibilities for scalable data collection [36, 42], causing coding to lose reliability and become intractable [1, 6].

QDA systems (QDAS) aim to support researchers during qualitative coding and analysis, with MAXQDA, Nvivo, Atlas.ti, Dedoose, WebQDA, and QDAMiner being commonly used [16]. Some of these systems incorporate machine learning (ML) to accelerate qualitative coding based on human annotations [12, 31, 48]. However, recent user studies demonstrated two critical shortcomings that impede the utility of available systems for enabling qualitative coding at scale [7, 13, 26]: (*i*) QDAS do not integrate ML as an interactive process that involves refining automated suggestions. The system mostly restricts the interaction between the user and the ML model to accepting and rejecting codes without insight into underlying coding rules. (*ii*) Therefore, code suggestions lack transparency, causing qualitative researchers the be reluctant to adopt ML-based support for qualitative coding.

This paper addresses these gaps by designing and evaluating a novel interactive AI-based ML system to support qualitative coding. Building on the recent work of the HCI and the interactive machine learning (IML) communities, we present Cody, a user-facing system for semi-automating coding. We present the results of two evaluations: Firstly, a formative evaluation to understand *how qualitative researchers interact with and whether they would trust an IML system to support coding?* Secondly, a summative evaluation, investigating *how qualitative researchers use Cody compared to the commercial and well-established QDAS MAXQDA?*

Our novel contributions include the following: We explain the design of the AI-based system Cody, which allows end-users to define and apply code rules (Figure 1b) while training a supervised ML model to extend coding to seen and unseen data (Figure 1c). Therein, we propose ideas for tackling challenges such as generating suggestions for code rules and cold start training of the ML model. Through interviews with qualitative researchers, after having used Cody for one week, we found that compared to MAXQDA, automated suggestions increased coding quality rather than coding speed. Further, while working with suggestions introduces an extra step to coding, this step is beneficial for researchers to get a better overview of the documents and to reduce the workload in the long run. Additionally, researchers desired explanations, particularly for ML-based suggestions, but rarely worked with them during the coding process. Finally, we discuss gains in intercoder reliability when using Cody; implications for designing suggestions to be *less* precise but *more engaging*; meta-issues around automated suggestions for qualitative research; and suggestions for future work.

## 2 RELATED WORK

### 2.1 Coding in Qualitative Data Analysis

Multiple disciplinary origins, such as sociology, psychology, and anthropology, shape the research traditions of qualitative research [33], including Ethnography, Phenomenology, and Critical theory, each with distinct aims. Approaches to analyzing qualitative data, such as content analysis or grounded theory, differ between traditions in terms of the main focus and aims of the analytical process [38]. While a comprehensive review of traditions and approaches is out-of-scope for this paper, we refer to Ritchie & Lewis [38] for in-depth information about qualitative research practices.

What unites many approaches to qualitative analysis is that they involve some sort of coding, where researchers aggregate information about the content of data by assigning short labels or *codes* – typically single words, sentences, or paragraphs [3, 14, 17–19, 24, 26, 35, 45]. Depending on the epistemological assumptions, researchers take two approaches to coding: *deductive* (codes are derived a priori from scientific theories) or *inductive* (codes emerge from the analytical process). Oftentimes, coding involves both deduction and induction at different stages of the research process [33, 38]. Codes themselves can constitute various levels of information depending on the researcher's needs but are commonly created either in a *descriptive* fashion, explaining higher-level concepts, or *in-vivo*, were responses are used directly to create codes and highlight themes. Coding allows researchers to make sense of the vast amounts of data typically created through interviews, field notes, and other qualitative data collection approaches.

The iterative, creative, and human-centered nature of coding [6, 35] makes it a time-consuming and error-prone task [7, 26, 46]. Code development and application takes hours of concentrated work, which is hard to perform reliably at scale [10], even for moderately sized datasets. With access to larger datasets and advances in computer-supported analysis, the adoption of qualitative data analysis systems (QDAS) has increased substantially [14, 16].

### 2.2 Qualitative Data Analysis Systems

QDAS offer a magnitude of features for organizing, structuring, coding, and analyzing texts and other digital data types such as audio or video to improve upon the traditional paper-based coding procedures [14]. Often, the institutional environment determines which systems researchers use, due to funding and access to training and support. Prominent examples of QDAS are Nvivo, Atlas.ti, and MAXQDA, with a similar feature set[1].

Despite the importance of coding for the entirety of data analysis, support to accelerate qualitative coding with automated procedures is limited [26]. With recent builds, Nvivo, Atlas.ti, and MAXQDA allow users to search for keywords and auto-code all occurrences [20, 27, 31]. Nvivo additionally includes an experimental feature that uses machine learning to automatically assign codes using existing coding patterns. The past five years have also seen the rise of various open-source QDAS. *INCEpTION* [21] and, more recently, *TEXTANNOTATOR* [2] provide web-based systems specializing in semantic annotation coding. Both systems aim to speed up semantic annotation by integrating active learning from human code examples (INCEpTION) or by providing automated pre-processing of data through named entity recognition, sentiment scores, and topic models (TEXTANNOTATOR). Tietz et al. [43] specifically evaluate the user interface of their semantic annotation system *refer* which combines manual and automated *annotations* in documents to improve coding quality. They find that a combination of manual and automated annotations achieves the most complete and accurate results [43]. As above, the evaluation of user-facing systems so far has focused on enabling users to annotate large-scale datasets for a range of NLP tasks without systematic attention to HCI and

[1] For a detailed overview of systems and capabilities, see, e.g., [12, 16]

qualitative data analysis [6, 26]. Focusing on qualitative coding, *Aeonium* uses ML not to speed up coding, but to draw the attention of collaborating qualitative coders to potentially ambiguous data [13].

Overall, features to accelerate coding in established tools are still at an experimental state and lack transparency, making them hard or sometimes impossible to validate [18]. With a user-centered inquiry, Marathe and Toyama [26] demonstrate that available QDAS remain "electronic filing cabinets" due to insufficient catering to qualitative researchers' needs. Issues with the quality of and trust in automated code suggestions and a lack of integration in the coding process have led to reluctance in adopting ML-based features [26]. Simultaneously, the focus of technologically advanced coding tools lies in supporting corpora creation for NLP tasks. Available systems are not designed to build trust in suggestions through an interactive coding workflow that combines manual and automated annotations [6, 26].

### 2.3 AI-based Qualitative Coding

Two approaches in the context of artificial intelligence (AI) are prevalent for accelerating qualitative coding: Natural Language Processing (NLP) and Machine Learning (ML). Crowston, Liu, and Allen [11] gave a prime example of both approaches by comparing human-created NLP rules against rules inferred with supervised ML. While both approaches offer promise for coding, manual development of NLP rules requires an expert, while ML-based rule development needs many examples. Crowston, Allen, and Heckman [10] extended their work focusing on rule-based coding support for content analysis and achieved commendable recall and precision of 74% and 75%, respectively, for some codes. However, creating NLP rules was time-consuming and difficult for rich codes, even for experts that defined rules ex-post from a coded dataset. Meanwhile, the open-source text analysis software *Cassandre* allowed users to define (multiple) single word rules by highlighting *markers* in a text [23], which could be grouped under one single label, forming a *register*. Cassandre then gathers all passages that include the marker. Lejeune [23] referred to the process of iteratively revising markers to improve registers as the *bounce technique*. Shortly after, scholars turned to supervised ML as one way to circumvent the definition of explicit NLP rules and have systems learn directly from manual coding [18, 24]. Yan et al. [47] developed a system for content analysis using a support vector machine and active learning principles for the multi-label classification of emails. While training multiple individual models for each label, they achieved a mean recall of 87% at the expense of precision (7%). At the same time, users lacked the technical skills to improve ML models through feature selection and required interactive and adaptive interfaces to understand ML outputs [47]. Along these lines, Chen et al. [7] called for research on interactive ML approaches, reimagining the use of ML in coding to make ML human-understandable. With *Aeonium*, Drouhard et al. [13] answered the call by giving an example of interactive ML with a system that does not utilize ML to suggest codes but to identify ambiguities. Finally, Marathe and Toyama [26] reported from an inquiry with qualitative researchers that while researchers desire automation, automation needs to be transparent and part of the coding process. They propose a novel spin at NLP rules by following a search-style querying approach that achieved a commendable 88% precision and 82% recall on average. Compared to the NLP rules used by Crowston, Allen, and Heckman [10], search-style rules are more accessible and might force researchers to develop coherent definitions for labels [18]. However, previous work on code rules had experts define rules ex-post, rather than following an interactive approach that enabled end-users to define rules as part of the coding process.

Overall, this short review indicates interest and promise in applying code rules and ML to support qualitative coding. Social scientists and HCI researchers alike (e.g. [6, 25, 26]) have called for research on designing interactive AI-based systems that integrate rule definition and ML model training into the process of qualitative coding while providing trustworthy suggestions. This paper presents an interactive AI-based system to bridge this gap and demonstrates results from two evaluations with qualitative researchers.

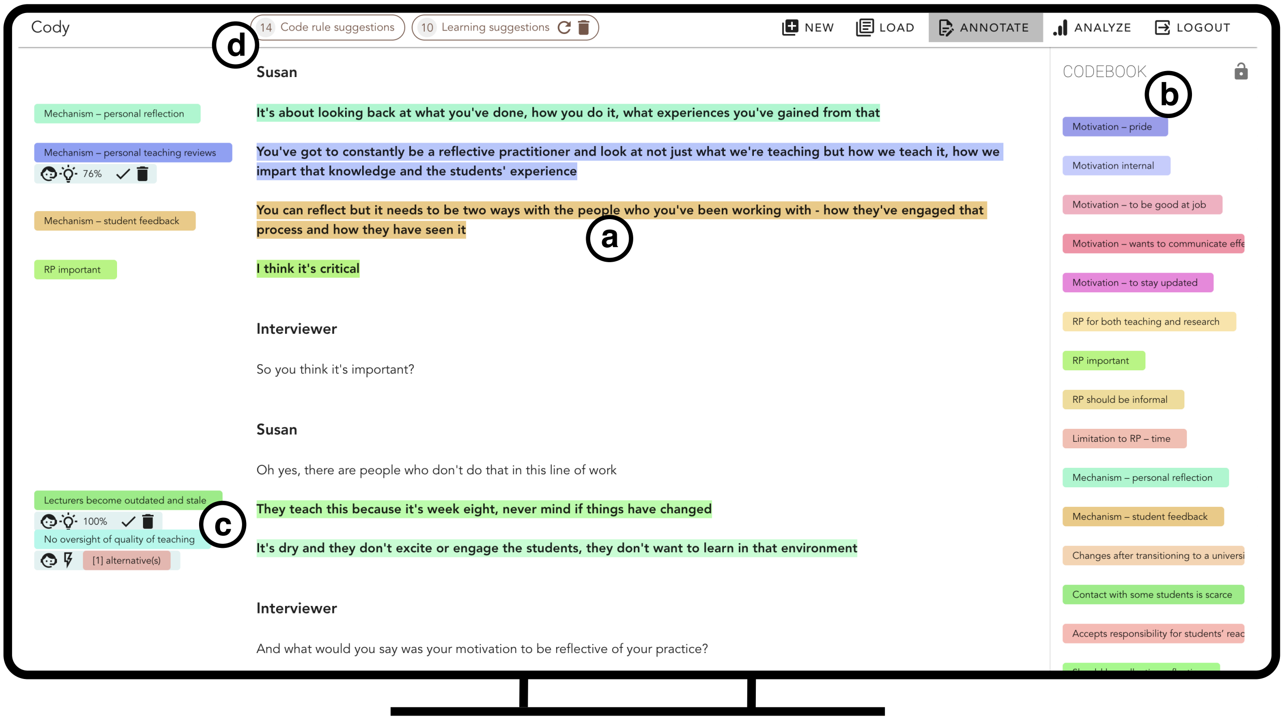
## 3 CODY

Cody emphasizes an interactive AI-supported coding process. Users can specify their desired unit-of-analysis, add annotations and codes, define coding rules, react to suggestions, and access a rudimentary statistics page. Figure 2 shows the interface of Cody during the coding process. This section details the requirements for Cody to support the coding process successfully.

### 3.1 System Requirements

We defined six requirements to build an assistive tool for qualitative coding that pays attention to the HCI and AI challenges posed by qualitative data analysis [45]. The requirements are inspired by the excellent user-centered study presented by Marathe and Toyama [26] and other related work [37]. By satisfying the following requirements, we build a system that may act as a stepping-stone towards Wiedermann's vision for qualitative research: "*In combination with pattern-based approaches, powerful visualizations and user-friendly browsers, [machine-learning algorithms] are capable to extend traditional qualitative research designs and open them up to large document collections.*" [45]

- *R1 Unit-of-analysis.* The unit-of-analysis (UoA) defines the level at which annotations are made to the text (e.g. flexible or sentence-level). The system should allow users to set a UoA for a document to improve consistency between multiple coders [10, 26].
- *R2 (Re)Define code rules.* Code rules can urge coders to combine keywords to form precise coding instructions [17]. Thereby, researchers might increase their understanding of the data [18]. During the coding process, coders encounter unexpected responses that effect previously defined code rules. As such, the system should enable coders to define and iteratively adjust code rules, applying the bounce technique [32] (Figure 3d).
- *R3 Seamless training of ML model.* Qualitative researchers' primary goal is not to train an ML model but to identify meaningful instances in data [6]. The system should require the user to be responsible for reviewing ML suggestions while hiding model and training complexity [3] (Figure 3f).


Cody
14 Code rule suggestions
10 Learning suggestions
d
NEW
LOAD
ANNOTATE
ANALYZE
LOGOUT
CODEBOOK
b
a
c
Mechanism – personal reflection
Mechanism – personal teaching reviews
76%
Mechanism – student feedback
RP important
Lecturers become outdated and stale
100%
No oversight of quality of teaching
[1] alternative(s)
Motivation – pride
Motivation internal
Motivation – to be good at job
Motivation – wants to communicate effe
Motivation – to stay updated
RP for both teaching and research
RP important
RP should be informal
Limitation to RP – time
Mechanism – personal reflection
Mechanism – student feedback
Changes after transitioning to a universi
Contact with some students is scarce
Accepts responsibility for students' reac

Susan

It's about looking back at what you've done, how you do it, what experiences you've gained from that

You've got to constantly be a reflective practitioner and look at not just what we're teaching but how we teach it, how we impart that knowledge and the students' experience

You can reflect but it needs to be two ways with the people who you've been working with - how they've engaged that process and how they have seen it

I think it's critical

Interviewer

So you think it's important?

Susan

Oh yes, there are people who don't do that in this line of work

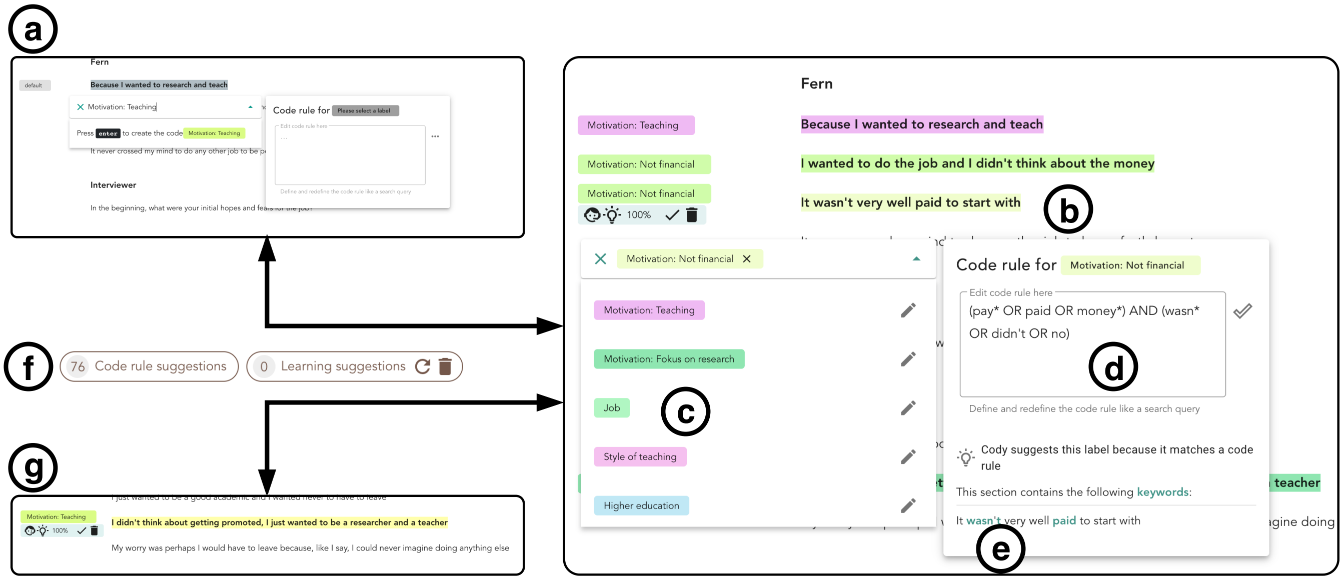
They teach this because it's week eight, never mind if things have changed

It's dry and they don't excite or engage the students, they don't want to learn in that environment

Interviewer

And what would you say was your motivation to be reflective of your practice?

**Figure 2: Final User Interface of Cody. (a) main annotation view, (b) codebook sortable via drag-and-drop, (c) Code suggestion with confidence and accept/reject buttons. Below, Cody highlights multiple alternative suggestions for a section, (d) Number of rule- and ML-based suggestions**


a
default
Motivation: Teaching
Press enter to create the code Motivation: Teaching
Code rule for Please select a label
Edit code rule here
...
Define and redefine the code rule like a search query
f
76 Code rule suggestions
0 Learning suggestions
g
Motivation: Teaching
100%
b
Motivation: Teaching
Motivation: Not financial
Motivation: Not financial
100%
Motivation: Not financial
Motivation: Teaching
Motivation: Fokus on research
Job
Style of teaching
Higher education
c
Code rule for Motivation: Not financial
Edit code rule here
Define and redefine the code rule like a search query
d
e

Fern

Because I wanted to research and teach

It never crossed my mind to do any other job to be p

Interviewer

In the beginning, what were your initial hopes and fears for the job?

I just wanted to be a good academic and I wanted never to have to leave

I didn't think about getting promoted, I just wanted to be a researcher and a teacher

My worry was perhaps I would have to leave because, like I say, I could never imagine doing anything else

Fern

Because I wanted to research and teach

I wanted to do the job and I didn't think about the money

It wasn't very well paid to start with

(pay* OR paid OR money*) AND (wasn* OR didn't OR no)

Cody suggests this label because it matches a code rule

This section contains the following keywords:

It wasn't very well paid to start with

**Figure 3: Coding workflow with Cody. Users make an new annotation and define a new code (a) which opens the code menu (b). Users may add codes to or delete codes from an annotation, or edit a code (c). Cody suggests a possible code rule that users can edit (d). When clicking on suggestions to open the label menu, Cody shows explanations (e). Code rules are applied on saving to create suggestions and can be accepted/rejected by clicking the respective icons (g). The number of available suggestions is shown in the menu bar (f), where users can trigger ML model retraining (refresh icon) or delete all ML-based suggestions (trashcan icon).**

- *R4 Iterative suggestions based on manual annotations.* As researchers value coding parts of their data to familiarize themselves with the material while desiring recommendations to reduce repetitiveness, the system needs to incorporate manual annotations and update accordingly [26].
- *R5 Foster reflection.* In qualitative coding, imprecise codes become apparent as data is re-coded by a second coder, triggering an iterative code revision process [35]. Code suggestions might act as a proxy for a second coder, as immature code rules help coders identify potential coding errors and enforce coding rigor [7, 26]. The system needs to enable researchers to spot potential issues to reflect and iterate on coding rules (Figure 2c).
- *R6 Include explanations.* Suggestions need to be easily understandable to enable coders to predict how changes affect suggestions, without requiring technical literacy [6, 8]. Without understanding the source of suggestions, coders not trained in ML techniques might reject suggestions altogether, while novice coders might accept suggestions too easily. The system should explain suggestions by referencing code rules or highlighting relevant keywords and providing a certainty factor (Figure 3e).

## 3.2 Coding Process with Cody

We developed Cody as a web-based system running on *Vue.js* (front end) and *Flask* (back end). Cody asks users to choose a UoA once a document is uploaded, which determines whether Cody automatically adjusts annotations to encompass an entire sentence (*R1*, Figure 2a). When applying a label to a selection, the user can use the label menu to review and adjust code rules by editing the rule in the text area (*R2*, Figure 3d). Upon saving changes to rules, the new rule is applied to the entire document to create new suggestions. Users can review suggestions by clicking on either the label or the annotation, e.g., to revise conflicting code rules (*R2, R5*, Figure 2c) or to view explanations for suggestions (*R6*, Figure 3e). ML-based suggestions are updated automatically after ten manual changes to annotations (adding, editing, deleting) or whenever the user clicks the *refresh* button (*R3, R4*, Figure 3f).

## 3.3 Suggesting Labels with Code Rules

When a user creates a new code, the system generates an initial code rule suggestion. Therefore, the system compares the new code with the words of the respective annotation using *similarity scores* (SiS) and *Levenshtein distance*[2] (LD). We use *spaCy*, a Python library for natural language processing (NLP), to calculate SiS. Initially, we remove stopwords[3], spaces, and punctuations from the annotation. Depending on the text's language, the system then uses a pre-trained model in German or English. It compares the context-sensitive tensors of each word in the code with the lemmatized remaining words in the annotation to identify potential synonyms for codes that exceed an arbitrary cut-off value (similarity > 0.45). We use the LD to additionally include words in the rule that have a close enough match (relative LD > 0.3)[4] to the given code. Rule suggestions are lower cased and no word can be contained twice. Initial code rule suggestions have the following form:

$$rule \rightarrow\ lemmatized(LD\ 1) * \ AND\ lemmatized(LD\ n)* \\ AND\ [\ lemmatized(SIS\ 1) * \ OR\ lemmatized(SIS\ n)*\ ]$$

Whenever Cody generates a new rule, or when a user changes a rule, Cody applies it to the entire document upon saving (Figure 4). We use the Python library *whoosh* [5] to search documents and identify occurrences [26]. We structure every document in sections to make code suggestions. In a typical interview transcript, each sentence will form one section. When a rule changes, whoosh parses the code rule into a search query and applies it to the indexed document, returning the IDs of matching sections. Cody relies on section IDs to update (add & remove) annotation suggestions on the front end. Thus, the system makes suggestions on the sentence level. Currently, code rules will not automatically account for syntax or spelling errors in the underlying data (e.g., interview transcripts). Users may include wildcards in code rules which allow for *softer* matches to handle noise. Further, Cody highlights matching keywords for a suggestion in the label menu, below the rule input text area. For rule-based suggestions, Cody highlights matched words in an excerpt from the current annotation (*R6*).

## 3.4 Suggesting Labels with Supervised ML

One crucial challenge to making code suggestions through supervised ML is the availability of labeled examples (cold start problem). Cody utilizes both manual annotations and rule-based suggestions to kick-start training the ML model (*R4*). As supervised ML algorithm, Cody trains a *logistic regression with stochastic gradient descent (SGD) learning*[5] to classify unseen data based on the available annotations (positive examples) using *scikit-learn* [34] (Figure 4). We use the words in annotations as features for training while removing language-depended stopwords. For preprocessing, we used most of the default settings of the TfidfVectorizer[6] from scikit-learn to create a learnable matrix of TF-IDF[7] features. In coding, researchers usually work with more than two labels, making the classification of sections a *multiclass* problem. In the multiclass case, we deal with a low number of positives for each label and lack explicit negative examples (annotations indicating the absence of a label). Cody creates artificial negative examples to increase training data by treating unlabeled sections of text above the last manual

[2] The Levenshtein distance can informally be defined as the minimum number of single-character edits (insertions, deletions or substitutions) that are required to change one word into the other.

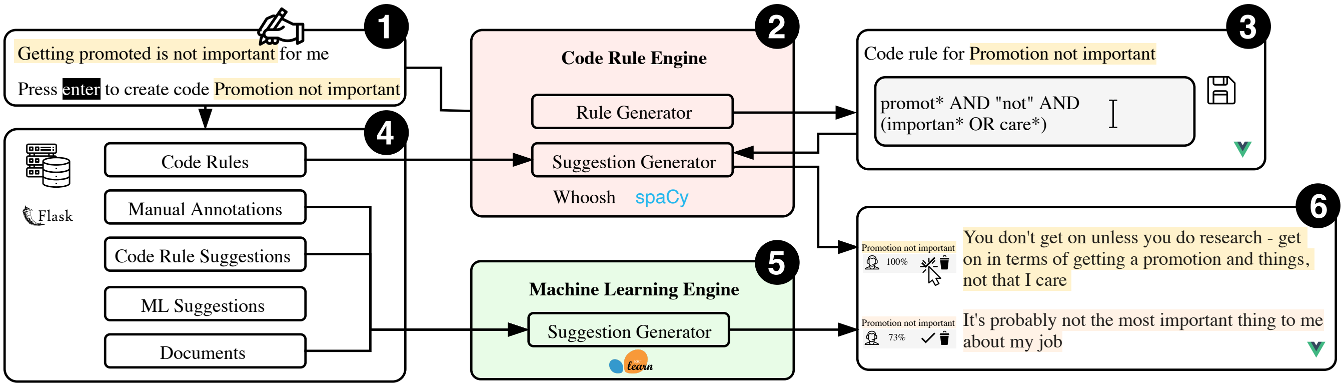
[3] Stopwords are words that occur with a high frequency independent of textual genre, e.g., 'the' in English [26].

[4] We determined cut-off values for similarity scores and Levenshtein distance through iterative testing of labels, annotations, and resulting rules suggestions. As such, the cut-off values are arbitrary, and other values will result in a different balance of words in the suggestions.

[5] We compared various techniques for supervised learning according to precision, recall, f1-Score, and training and prediction time to select the most promising algorithm for our scenario. SGD fitting a logistic regression outperformed other algorithms (SVC, MNB, Random Forest, Logistic Regression, SGD with linear SVMs, Neural Network with LBFGS solver) with an f1-Score of 0.48. With hyperparameter tuning, we could achieve a label accuracy of .677 and an overall accuracy of .734, using a logarithmic loss function, balanced class weights, and the *elasticnet* penalty. While these values might seem unimpressive at first, the scores were achieved with a training set of 90 positive examples from eight different labels for predicting 721 unlabeled sections.

[6] Adjusted settings were sublinear_tf = True, min_df = 2, encoding = latin-1, ngram_range = (1,2)

[7] TF-IDF, short for 'term frequency – inverse document frequency', is a numerical statistic intended to reflect the importance of a word in a document or a collection of documents.



Getting promoted is not important for me

promot* AND "not" AND (importan* OR care*)

You don't get on unless you do research - get on in terms of getting a promotion and things, not that I care

It's probably not the most important thing to me about my job

**Figure 4: System Architecture. (1) User makes an annotation, (2) code rule engine generates rule suggestion, (3) new rule is displayed for user review, (4) save triggers suggestion generator to search indexed document for occurrences, (4) and sends update to suggestions in the database. (5) Machine learning engine retrains model and makes suggestions, (6) displayed for user review in the front end.**

annotation as negatives, assuming that the user makes annotations from top to bottom. Introducing artificial negatives (*greygoo* labels) also enables the algorithm to mark a section as "not relevant" if the predicted label is greygoo. Furthermore, we draw inspiration from the *S-EM algorithm for PU learning*[8] to create a threshold for inaccurate suggestions [39]. We sample spies ($S$) from the labeled training data ($L$) through a test-training split, so that $|S| = 0.1 \times |L|$. After training the model with the available training data for all codes ($C$), we predict labels for every spy ($s$). Cody will only display ML-based suggestions for codes ($c$) for that all spies were predicted correctly, thereby prioritizing precision over recall, i.e.,

$$c = \{\ c \epsilon C : \forall s \epsilon S : q(s|c) = s|c\ \}$$

with *q(s|c)* being the predicted spy-code combination for spy $s$ and $s|c$ being the correct spy-code combination. When the model fails to correctly predict spies for each of the available codes, we deleted all existing ML suggestions.

Our strategy of continuous real-time retraining of the ML model as the labeled data changes impacts the selection of an appropriate ML-model, as low average training times are crucial. In our experiments, model training only took milliseconds, depending heavily on the amount of labeled training data. We expect frequent model retraining to be useful when the prediction model is less stable, which is the case with a low amount of training data – resulting in fast model retraining. As the amount of labeled data grows, the model should become more stable and would not need (re)training after every change.

For ML-based annotations, Cody displays counterfactual explanations in the form of indicative words for a suggestion to both help users understand the words of a sentence that the algorithm learned while potentially providing them with ideas for revising code rules (*R6*). The calculation of counterfactual explanations is comparable to the calculation of *Shapley Values*, which explain a prediction by highlighting the impact of individual features. Cody calculates the impact of a feature (each word of a sentence) by predicting a label while removing one word (or combinations of words) from a sentence (*R6*) (heuristic approach, c.f. [25]). Due to the computational costs of the pairwise comparison, Cody stops after iterating through all one- and two-word combinations.

# 4 EVALUATION

During development, we conducted a formative evaluation to understand how researchers interact with our prototype(s), followed by a summative evaluation to compare the interaction with Cody against MAXQDA.

## 4.1 Formative

Formative evaluations aim at collecting information to improve an artifact [38]. Following the call-for-research for building and evaluating a user-facing interface [26], we firstly focused on evaluating how Cody's design, combining rule-based with ML-based suggestions, was perceived by qualitative researchers and determined necessary changes.

*4.1.1 Method.* We recruited participants following criterion-based sampling via a graduate university mailing list. Participants needed to be PhDs or PhD students with prior training in qualitative research who personally performed qualitative coding for at least one study in the last year. Additionally, participants should have coding experience with a QDAS. Six PhD students agreed to participate, whom we invited for two subsequent iterations over two weeks.

We used contextual inquiry to guide the data collection [4]. Each session for both iterations consisted of three parts: (1) Introduction to Cody (5 mins), (2) in-situ evaluation with the think-aloud-method (25 mins), (3) Semi-structured interview on user experience (30 mins). We provided participants with a task description to follow while sharing their thoughts, ideas, and problems following the think-aloud-method [15]. In the task description, we asked participants to perform three tasks: (1) Load their document into Cody. Participants gave us access to data from own projects, which we converted to a file type that Cody could process. (2) Switch to the coding view, and (3) Perform qualitative coding on the document

[8] S-EM: Spy expectation-maximization, PU: Learning from labeled and unlabeled examples.

by recreating the coding process applied when originally analyzing the data. While participants used Cody to code their dataset, we took notes while observing their progress on a second screen. Each session concluded with a semi-structured interview, during which we asked participants for the features they most liked and disliked; perception of code rules; perception of interface and coding efficiency; trust in suggestions; differences to their usual coding process and perceived usefulness; and willingness to use Cody to partially automate coding.

We transcribed the audio recordings of each session. The first author conducted inductive coding on both transcripts and field notes, followed by discussions with both authors to iteratively refine emergent themes. We summarized findings on a per-participant level by comparing observations and aggregated findings to identify required and future improvements. Our goal was to understand user's work practices with Cody, to improve the user-facing interface. We use pseudonyms for anonymity and present slightly edited quotes for readability.

*4.1.2 Findings: First Iteration.* We started with a prototype running locally on a laptop. While already having the final artifact's functionality, this prototype of Cody aimed to minimize the actions users would have to take to code a document. Code rules were saved automatically and applied with every change. Cody would retrain the ML model whenever users added or edited an annotation, or when a code rule was applied. Due to the relatively small number of labeled data available for model training, the processing time for retraining was in the range of milliseconds. Further, the Cody prototype did not indicate how many suggestions it created so far.

Participants could use Cody with their data and coding scheme, if only for a short period of time. Tom, who commonly works with grounded theory, found Cody useful for initial coding as part of open coding: "*I think it would help me with a certain number of interviews to be faster with initial coding. I always have to identify [security requirements from qualitative interviews with experts], that takes time but has only limited benefit.*" Participants found rules particularly relevant for studies with many similar interviews, where they can learn from an initial sample and use rules to reduce repetitiveness. Lana explains: "*I've roughly 81 interview pieces – it became very boring and repetitive. Because they are only short statements, no in-depth interviews [. . . ], but until then, I learned enough to be able to define rules for the remaining pieces.*" Interestingly, participants felt responsible for incorrect suggestions, having defined the underlying rule themselves: "*it misused customer service, but because I made a mistake*" (Cora). Further, we did not know how participants would think about the quality of suggestions for code rules. The quality did not matter much, as participants required suggestions for rules primarily as examples to learn about the rules' syntax: "*not every researcher is familiar with code rules, that's why it's important that this tool suggests rules and also shows how they should work. Otherwise I think this wouldn't be used*" (Cora).

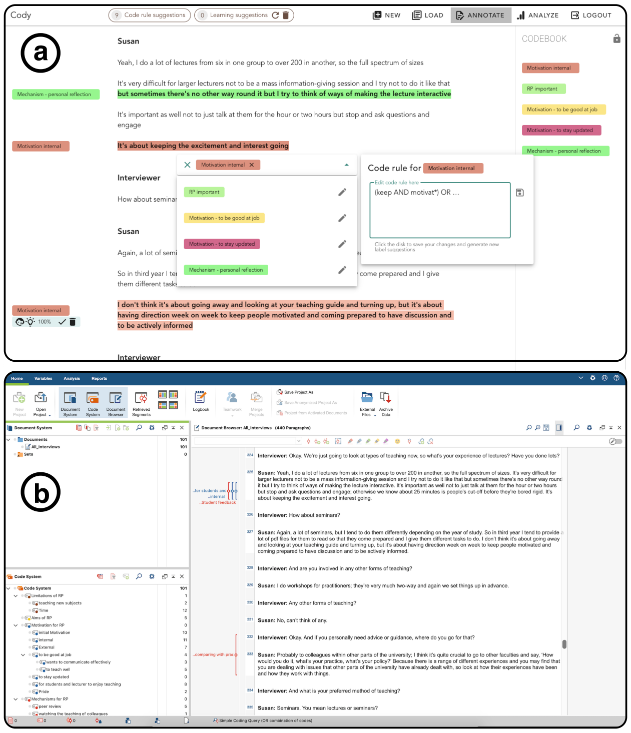
The first prototype iteration convinced us that automated suggestions are perceived as beneficial when applied correctly. However, participants reported that they desired more control over the generation of suggestions, a better way to accept/reject them, and to see the number of generated suggestions. We adjusted the prototype accordingly and deployed it to a server to enable a remote evaluation.

*4.1.3 Findings: Second Iteration.* The second prototype was accessible on the web. Compared to iteration one, we changed the interface to be more intuitive at the cost of requiring more user actions. As such, users now had to save code rules manually, triggering their application. Cody retrains the ML model once every ten changes to annotations rather than after every change. We made this change to reduce the frequency with which we confront users with new suggestions. Further, users can manually request model retraining and the deletion of all ML suggestions. The menu bar now shows the number of existing rule and ML suggestions. Users can accept or reject individual suggestions directly via button-click. We added user profiles to allow for multiple users working with Cody simultaneously.

Overall, participants perceived the second prototype as helpful primarily to structure documents better and faster. Josh explains: "*what you can do much better with this tool than with MAXQDA or other tools is to explicitly deal with a topic. I could go back now and look at everything related to customers, and then I could look at everything related to platforms and so on. I don't have that in the other [tools], I would work through the document linearly, jumping back and forth between topic blocks. And that's why this can improve the coding because I can focus much more.*" Eric thought Cody to help more by reducing workload rather than improving coding quality: "*of course, there would be fewer errors, but it would not directly improve the quality. I would expect myself to work correctly; it would rather make it easier for me.*"

However, participants also had concerns about using Cody: One, Seth was afraid of "*missing certain things*" mainly when using AND operators in rules. Second, Eric had prejudices towards ML and ignored ML suggestions, feeling that they "*cannot work with that little amount data.*" However, he would feel better once he had labeled "*three to four documents*," which would also help him to define code rules: "*to create good code rules, not only do I need coding experience, but I also need to know the text.*" Adding to this, Sven said: "*I think it makes a lot of sense if you let theory guide you and what you want to find in an interview. If I use in-vivo coding, then code rules are of no use to me. But if I want to have some kind of structure, and want to break something down, then it makes sense.*" Participants felt that the usefulness of code rules lies in giving structure and that rules are best defined once they had familiarized with the text. Eventually, automated suggestions would help to "*perceive the text as a whole*" (Josh), as it requires researchers to also re-read individual sections to review suggestions.

To summarize, participants perceived the automated suggestions of the second prototype to be most helpful for "*getting an overview faster*," (Eric) "*having a speed advantage*," (Seth) and building the codebook "*better, more stringent*" (Josh). Despite these benefits, Seth also noted that it would be a "*higher initial effort*," leading to coding "*becoming much easier.*" However, the interaction with the prototype was too short for participants to observe these effects for themselves. Josh explains: "*I can't judge this conclusively, you would have to do it with 20, 30, 40 codes to be able to say that.*"



**Susan**

Yeah, I do a lot of lectures from six in one group to over 200 in another, so the full spectrum of sizes

It's very difficult for larger lecturers not to be a mass information-giving session and I try not to do it like that **but sometimes there's no other way round it but I try to think of ways of making the lecture interactive**

It's important as well not to just talk at them for the hour or two hours but stop and ask questions and engage

**It's about keeping the excitement and interest going**

**Interviewer**

How about seminar

**Susan**

Again, a lot of semi

So in third year I te … come prepared and I give them different tasks

**I don't think it's about going away and looking at your teaching guide and turning up, but it's about having direction week on week to keep people motivated and coming prepared to have discussion and to be actively informed**

**Interviewer**

Code rule for Motivation internal

Edit code rule here

(keep AND motivat*) OR …

Click the disk to save your changes and generate new label suggestions

Document System

| Documents | 101 |
|---|---|
| All_Interviews | 101 |
| Sets | 0 |

Document Browser: All_Interviews (440 Paragraphs)

324 **Interviewer:** Okay. We're just going to look at types of teaching now, so what's your experience of lectures? Have you done lots?

325 **Susan:** Yeah, I do a lot of lectures from six in one group to over 200 in another, so the full spectrum of sizes. It's very difficult for larger lecturers not to be a mass information-giving session and I try not to do it like that but sometimes there's no other way round it but I try to think of ways of making the lecture interactive. It's important as well not to just talk at them for the hour or two hours but stop and ask questions and engage; otherwise we know about 25 minutes is people's cut-off before they're bored rigid. It's about keeping the excitement and interest going.

326 **Interviewer:** How about seminars?

327 **Susan:** Again, a lot of seminars, but I tend to do them differently depending on the year of study. So in third year I tend to provide a lot of pdf files for them to read so that they come prepared and I give them different tasks to do. I don't think it's about going away and looking at your teaching guide and turning up, but it's about having direction week on week to keep people motivated and coming prepared to have discussion and to be actively informed.

328 **Interviewer:** And are you involved in any other forms of teaching?

329 **Susan:** I do workshops for practitioners; they're very much two-way and again we set things up in advance.

330 **Interviewer:** Any other forms of teaching?

331 **Susan:** No, can't think of any.

332 **Interviewer:** Okay. And if you personally need advice or guidance, where do you go for that?

333 **Susan:** Probably to colleagues within other parts of the university; I think it's quite crucial to go to other faculties and say, 'How would you do it, what's your practice, what's your policy?' Because there is a range of different experiences and you may find that you are dealing with issues that other parts of the university have already dealt with, so look at how their experiences have been and how they work with things.

334 **Interviewer:** And what is your preferred method of teaching?

335 **Susan:** Seminars. You mean lectures or seminars?

Code System

| Code System | 101 |
|---|---|
| Limitations of RP | 1 |
| teaching new subjects | 2 |
| Time | 12 |
| Aims of RP | 5 |
| Motivation for RP | 0 |
| initial Motivation | 10 |
| internal | 11 |
| External | 7 |
| to be good at job | 4 |
| wants to communicate effectively | 3 |
| to teach well | 5 |
| to stay updated | 0 |
| for students and lecturer to enjoy teaching | 8 |
| Pride | 2 |
| Mechanisms for RP | 0 |
| peer review | 5 |
| watching the teaching of colleagues | 1 |

Simple Coding Query (OR combination of codes)

**Figure 5: Screenshots of the User Interface of (a) Cody and (b) MAXQDA. Both screens demonstrate what participants in their respective treatment saw during the summative evaluation.**

## 4.2 Summative

A summative evaluation of an intervention or artifact is concerned with its impact on the effectiveness and resulting outcomes [38]. As such, we evaluated Cody's effectiveness compared to MAXQDA, one of the most well-known QDAS [16]. For the summative evaluation, we used the second version of Cody (see Figures 2 and 5a).

*4.2.1 Method.* We invited participants from a pool of 3.500 university students using criterion-based sampling [38]: (I) Bachelor's degree, (II) performed at least one qualitative study, (III) experience with qualitative coding for at least one qualitative study, and (IV) excellent English skills. We selected these criteria to ensure that participants are experienced in qualitative analysis. Eleven people ultimately agreed to participate. Table 1 presents a summary of participant characteristics both for formative and the summative evaluation as well as statistics of participants' interactions with their respective coding tools. We tasked participants with coding a dataset over one week in a between-subject design: one group using MAXQDA, the other Cody. Figure 5 shows screenshots of the interface of (a) Cody and (b) MAXQDA. We used a public dataset of interview transcripts on reflective practice in higher education [19]. By evaluating Cody with a public dataset, we want to enable other researchers to evaluate future tools against the same dataset, as coding depends heavily on the underlying data. Furthermore, the dataset comes with a student guide for participants on how to code, steps to follow, and a complete codebook. Through the student guide, participants can evaluate the transcripts with a concrete goal: to *identify feelings about reflective practice and how it was put into practice* [19]. Thus, we evaluated coding assistance with first-pass coding with a pre-developed codebook, as suggested by Marathe and Toyama [26]. However, participants were free to add new labels should they need to. At the beginning of the week, we invited participants to a 1-hour online workshop to introduce them to the task using the student guide, including a 15 minutes introduction to their respective QDAS. We conducted individual 30 minutes long semi-structured interviews with all participants after they finished the task. During the interview sessions, we asked participants about their coding experience with the QDAS compared to tools they are familiar with; perception and usefulness of automated suggestions; explanations and effect on trust; and if they would use tools that semi-automate coding. None of our participants in any study had prior experience with rule-based coding of qualitative data. We compensated participants with €90 for their time and expertise.

We transcribed the audio recordings of all interviews. The first author conducted inductive coding on the transcripts, followed by iterative discussions with both authors to refine emergent themes. While we could collect usage data from Cody, for MAXQDA, we partly rely on self-reported data from participants, such as the duration of coding. From participants MAXQDA project files, we extracted the number of annotations made and the labels participants used. For Cody, we measured various parts of the interaction, such as the time taken to code, the number of manual or automated annotations, and how often code rules were adjusted. Based on the coded documents, we calculate Krippendorff's Alpha as a measure of intercoder reliability for both treatments [22]. The calculation of Krippendorff's Alpha required some preprocessing: We corrected spelling mistakes in codes and differences in the usage of symbols (- and –), which impact the calculation. For MAXQDA data, we transformed the data to match the export structure from Cody, to use the same calculation. We once again use pseudonyms for anonymity and present slightly edited quotes for readability.

We detail two types of findings: (1) *Impact of Automated Suggestions on Coding* highlights how rule- and ML-based suggestions influenced participants' coding. (2) *Implications for Designing AI-based Coding Support* presents three recommendations for automated QDA assistants.

*4.2.2 Findings: Impact of Automated Suggestions on Coding.*

*Code rules increase coding quality.* An imprecise rule, when applied to an interview, creates multiple wrong suggestions. While participants needed some time to understand how to define rules at an appropriate scope, the process of iterating rules engaged them to think about their coding. Ella explains: "*it helped in the sense that I thought about: 'what does it have to contain to fit?'.*" Further, users tend to work with many overlapping labels. More precise definitions help to reduce overlap: "*as the codebook grows, I'm not even sure which code matches which text correctly. There are overlaps, that's why it's difficult if you haven't defined the codes correctly [...] I think it helps a lot to structure it much, much better from the beginning using exactly these keywords as search criteria.*" (Ena). Overall, participants reported having a better understanding of the coding

**Table 1: Summary of participant characteristics and statistics. Participants are pseudonymized. We use 'Disc' for discipline, 'Meth' for methodology, 'STS' for sociotechnical studies, 'HCI' for human computer interaction, 'IS' for information systems, 'GT' for grounded theory, 'MQ' for MAXQDA. For statistics, we use 'Ann' for annotations, 'Acc' for accepted suggestions, 'R chg' for number of changes to rules, 'ML ref' for number of manual ML refreshs, time in hh:mm, 'Pre' for precision, 'Rec' for recall, and 'GG' for including greygoo examples for training. Precision and recall are taken from the final model retraining.**

| **Formative** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ***I1*** | *Disc* | *Meth* | *QDAS* | ***I2*** | *Disc* | *Meth* | *QDAS* | | | | |
| Cora | IS | Iterativ | Miro | Eric | IS | Deductiv | MQ | | | | |
| Lana | HCI | Inductiv | MQ | Josh | HCI | Iterativ | MQ | | | | |
| Tom | STS | GT | Miro | Seth | HCI | Iterativ | MQ | | | | |
| **Summative** | | | | | | | | | | | |
| *Name* | *Tool* | *Codes* | *Ann* | *Acc* | *R chg* | *ML ref* | *time* | *Pre (GG)* | *Rec (GG)* | *Pre* | *Rec* |
| Ella | Cody | 40 | 207 | 16 | 50 | 16 | 05:06 | 0.76 | 0.78 | 0.20 | 0.13 |
| Ena | Cody | 37 | 383 | 139 | 31 | 23 | 08:26 | 0.61 | 0.56 | 0.58 | 0.36 |
| Kelly | Cody | 52 | 119 | 3 | 51 | 9 | 04:56 | 0.83 | 0.81 | 0.00 | 0.00 |
| May | Cody | 27 | 85 | 2 | 9 | 8 | 03:10 | 0.92 | 0.89 | 0.08 | 0.17 |
| Nas | Cody | 36 | 173 | 48 | 20 | 10 | 06:47 | 0.82 | 0.81 | 0.50 | 0.38 |
| Paul | MQ | 42 | 162 | - | - | - | 08:00 | - | - | - | - |
| Sana | MQ | 40 | 114 | - | - | - | 05:30 | - | - | - | - |
| Stev | Cody | 36 | 126 | 7 | 5 | 11 | 03:55 | 0.79 | 0.77 | 0.31 | 0.15 |
| Tabi | MQ | 62 | 135 | - | - | - | 05:00 | - | - | - | - |
| Vic | MQ | 23 | 101 | - | - | - | 05:15 | - | - | - | - |
| Zoe | MQ | 27 | 152 | - | - | - | 03:30 | - | - | - | - |

scheme. As May puts it: "*we commonly work with definitions, but you don't see, it's mostly concepts, but not what words are relevant. Using [Cody], we have it clear and systematized*." We were interested in seeing if the alleged understanding of the coding scheme translated to increased intercoder reliability (ICR), and calculated Krippendorff's Alpha. We selected Krippendorff's Alpha as a measure for ICR due to its applicability with six individual coders. In their insightful discussion of the value of calculating ICR, McDonald et al. (2019) argue that ICR can be a useful measure when applying a codebook to data [29]. For MAXQDA, five unique coders with an average of 132 annotations/coder had an Alpha of 0.085. For Cody, six unique coders with an average of 182 annotations/coder had an Alpha of 0.332. Also, rules are useful for understanding the work of other coders, mainly when code definitions are not discussed: "*It will be easier for third parties to understand. What was done, which rules were used to code the document* (Sana)."

However, the characteristics of the data and the aim of the analytical process determine the usefulness of code rules. The more structured the data, the easier it is to define rules that result in precise suggestions. Particularly with data from (semi)-structured interviews, rules can be fine-tuned to code specific sections of interviews (e.g., age and demographics) or responses to questions reoccurring across interviews (e.g., why did you decide to enter higher education?). Ella states: "*it depends on the questions and how standardized the whole thing is done. I could imagine if you have a lot of yes-no questions, it can help quite well*." Luckily, interviews with a structure that suffices for rule creation also tend to be repetitive and time-consuming with little analytical reward. With interviews where meaning is hidden in context, code rules fail to provide useful suggestions as they discard dynamic semantics. Ella said: "*I revised [rules], often [. . . ] if you think to general, you suddenly have 120 suggestions, then I changed it and had one. It's hard to balance, the answers can be the same but still so different, that the rule fails to find it*." Further, code rules work best with an established codebook, e.g., when applying deductive coding. Lana states: "*If I don't have a codebook that I want to apply, I just try to see what is in [the document], without defining rules. But I think it makes a lot of sense if theory guides your coding and you want to find something from theory in an interview*."

Despite the drawbacks of rules in dealing with context to make precise suggestions, participants also found rules to help structure data. Thereby, rules enable the scanning of documents for particular topics of interest. As Stev puts it: "*[Cody] definitely is a good support in the sense that, for example, I want to code everything related to motivation, then it takes work off my shoulders. Normally I would do this by hand using Ctrl+F and the mark relevant sections. This helps me not to overlook things*."

To summarize, participants enjoyed working with code rules and used them not only to generate suggestions but also to rethink their coding. While they were not convinced that they could appropriately formulate rules for every type of code or data, they valued the feature for structuring interviews and increasing their understanding, especially for unfamiliar data. Participants using Cody had a higher intercoder reliability, compared to participants using MAXQDA.

*ML suggestions should prioritize precision over recall.* Cody's design purposefully hid the complexity of ML suggestions from the user. While some participants could barely tell whether they worked with ML suggestions, they valued not having to deal with rejecting multiple unhelpful suggestions. As such, systems should prioritize precision over recall when training ML models. Zoe explains: "*if I* *can only accept one of many suggestions, then it's a waste of time, because I have to check every time [. . . ] So I'd rather have [suggestions] less often and more precise*."

Particularly the low number of positive examples for each label is challenging for model training, reinforcing the notion that a system should be careful not to distract the user with premature ML-based suggestions. Despite the low number of positive examples, Kelly had a positive experience with ML-based suggestions: "*those suggestions, that appeared without me changing [a code rule], this was something I didn't have before. And for some sections, where it made sense, it really reduced your workload*." Further, participants were not distracted by having to reject wrong suggestions, given that wrong suggestions are not perceived as prevalent. "*A few times it really helped, but often I had to delete suggestions. Yes, I think it was ok. It's useful that the possibility exists at all*", Nas said.

Thus, ML-based suggestions are a double-edged sword. While they help to not miss exciting phenomena in the data, they lack quality when the number of positive examples is limited, and require strict thresholds. In combination with code rules, ML suggestions are useful to extend suggestions to some of the *false negatives of rules*, supporting users in improving rules by highlighting instances that existing rules are missing. Hence, ML suggestions can support users if they focus on precision over recall, providing limited support while minimizing distractions. The coders' desire to work through their entire dataset additionally reduces the risk of missing relevant sections due to a low recall.

*Checking suggestions is a beneficial extra step.* Earlier user inquiries reported that researchers fear that automation would be adding one more step to coding, having to check not only what code the researcher would use, but also what the computer said [26]. All six participants working with Cody confirmed that while the coding process with Cody did not require them to change their general process, it took time to (re)define rules, and navigate the document, to accept and reject suggestions. Two participants quickly discarded checking seen data for new suggestions for a comprehensive check-up once they finished coding: "*towards the end, I didn't bother because I noticed that new [suggestions] would pop up anytime anyways. But especially in the beginning, I searched for them*" (Nas), "*maybe what was different than if I had done it with another software is that at the end I searched the whole interview for suggestions and either accepted or deleted them*" (Stev). An assistive system should make it easy for users to review suggestions, particularly those added to seen data. Ella and Nas suggest assisting users with reviewing new suggestions, thus reducing the disruption of the coding process. In Ella's words: "*When there are suggestions, I want to be able to go there and return to the position where I left*." Further, reviewing suggestions for seen data had participants reexamine manual annotations, and sometimes revealed sections that had been overlooked. Overall, participants on average took similarly long to code the data between treatments (5:22 h with Cody to 5:20 h with MAXQDA). While we cannot draw conclusions regarding coding time due to the lack of internal validity, participants were convinced that using code rules can accelerate their coding process. However, they said that the number of interviews was too low to make appropriate use of rule-based suggestions.

Thus, reviewing automated suggestions, when provided not only for unseen but also for seen data, introduces an additional step to coding. While participants desired support on the interface-level to quickly review suggestions, they did not perceive Cody's suggestions to impact the coding procedure negatively. On the contrary, Stev and Ena said that they used suggestions to double-check codes in a second-pass and get a better overview of the data as a whole: "*[. . . ] you were brought to look more often and without this help, you would have overlooked one or the other thing especially in the first run, you would have had to go through more often*" (Stev). Ena voiced the following when asked whether the automated suggestions helped: "*Yes, definitely. In the beginning, it was quite time consuming to create all of them and to think about it. But it was cool when I had a page where five or six [annotations] were suggested, and I just had to read through and check 'do they fit, yes, no' [. . . ] I really had the feeling that the work was easier*."

#### 4.2.3 Findings: Implications for Designing AI-based Coding Support.

*Provide suggestions at an appropriate level of detail.* Especially participants using MAXQDA imagined suggestions not at a one-code level of detail visible in the text, but as assistance to reduce the choice of codes for an annotation. Tabi explains: "*It would be nice if I had some suggestions [. . . ] Maybe so that I only have to choose between five codes, so I don't have to look through all 30 codes, when I make a selection in the text. Like three to five options*." Further, Paul suggests to only highlight interesting sections without making a code suggestions, highlighting potential sections of interest: "*the algorithm says, 'something could be here,' but you have to think for yourself if you want to do something with it, it would enhance you own process*." Participants using Cody, on the other hand, showed little interest in simple highlights instead of suggestions. However, they were interested in multi-label suggestions. Kelly explains: "*you might lose the overview and accept [the suggestion] if only one code is suggested. But when you have several, then you can think about it again – which one fits best*?" There were two reasons for this preference. First, having three codes suggested strengthened users' confidence that the algorithm had considered all options. While the algorithm considered all choices for any decision, Sana felt that the algorithm might have missed something: "*With only one code suggested you think 'has it really seen everything?' And with three, I would know that there is a higher probability that it selected the ones that fit*." Second, participants felt at risk of accepting suggestions too quickly, particularly when being tired. Having multi-label suggestions requires users to make an active choice. Eventually, participants felt that this choice would help them trust the algorithm more. Sana explains: "*it remains transparent. Even when you have selected one out of three, you may still be able to see these three later. If you take your time to look at it again and see 'ah there it suggested these three, looking at it again, it still makes sense for me.'*"

To summarize, participants welcomed the idea of having suggestions not only provide one but three to five potential codes, increasing the involvement in decisions at the cost of additional work. It is primarily essential that a human is the last instance for reviewing suggestions, not allowing the system to "*auto-code* (Paul)."

*Explanations are desired but get ignored.* When asked about trust in and transparency of automated suggestions, participants using MAXQDA regarded explanations as elemental to understanding suggestions and working with an assistive system. While participants using Cody partly voice requiring explanations, they pay no attention to the explanations provided by Cody: "*There was something, but I probably didn't look at it very closely*" (Nas), "*generally, if they [suggestions] make sense, they make sense [. . . ] I don't know if it's important that I see or don't see the specific rule*" (May), "*I verify that for myself and think about whether it can make sense*" (Ella). Primarily, participants are convinced by helpful suggestions. Sana explains: "*I would check it myself a few times in the beginning and when I realize that it suggests the right thing, I would not doubt that in the future. I don't know if it needs a direct explanation.*" Hence, explanations should be provided, particularly on user request, but the perceived quality of suggestions decides the user's trust. Tabi explains that reading explanations is a trade-off, requiring time that could otherwise be used for coding. In Tabi's words: "*it would be nice, but takes time. The more explanations you have to read, the longer the process will take*". Eventually, the initial impressions are crucial for users' decision to adopt automated suggestions or ignore them (or turn them off). Further, users saw little value in the confidence scores we showed, saying that "*it would not strengthen my trust [. . . ] having no idea how it was calculated*" (Sana).

*Automation should encourage and support experimentation.* Despite all users of Cody describing using code rules as "*new*" (Ella), "*exciting*" (Kelly), and "*interesting*" (Vic), they rarely started the task by trying to learn how to use them. Only Stev began coding by "*figuring out how to add a code, how to rename it, how do these rules look like, so I wrote an example with an asterisk to see if it automatically highlight the next line, which had such a keyword in it.*" Most participants took some time to figure out how to write code rules in a granularity that worked for their coding. Kelly explains: "*In the beginning, I may have formulated code rules a bit imprecisely, and it came up with suggestions which didn't fit at all. Then I always had to adapt by trial and error. But if you did it a couple of times, then it worked, then you learned how to formulate them in a way that gets you the results you want. And then [the suggestions] helped, because that's when you got suggestions that really fit.*" Participants did not actively look for more information or familiarize with the tool before starting the task. Rather, they wanted to familiarize themselves with the functions and possibilities as they go. Ella explains: "*it's a learning-by-doing kind of process. The general introduction was enough. The rest you have to work out by yourself.*" None of the participants coded the entire dataset in one go, thus valuing on-demand introductions to certain features of a tool: "*I want to be able to say: 'Hey, now I want an introduction to the function.' Instead of being overwhelmed on my first use, why can't the tool remind me like 'Hey, how about trying the automation now?'*" (May).

To summarize, participants follow a learning-by-doing approach in working with code rules. An assistive tool should encourage experimentation and provide some guidance or on-demand assistance while ensuring that users can test without fear. "*I would adjust rules and would work with it because I see the benefit. [. . . ] What is important is that I know that no other labels disappear, that I lose nothing,*" Tom urged.

# 5 DISCUSSION

## 5.1 Working with Automated Suggestions

With our study, we pursue the goal of designing, building, and evaluating a user-facing system that integrates both prevalent strategies for (semi)-automating coding: code rules [9–11, 18, 26] and (supervised) machine learning [2, 21, 28, 43, 47, 49]. Prior work on code rules has focused on evaluating rules defined by experts against gold standard datasets [10, 11, 26], while Cody focuses on enabling and supporting end-users in defining and reworking rules during coding. Through the formative evaluation with qualitative researchers, we identified the importance of rule suggestions to educate and encourage users to work with rules. While we drew some inspiration for automatically creating rule suggestions from literature on text mining [30], information extraction [40], and classification [41], prior work at large did not focus on creating rules that are easy for users to read and edit. From our summative evaluation, we learned that while users had to change the suggested rules, as we intended them to, they valued the support and did not refrain from working with rules. Further, the final rules that users created were quite heterogeneous, some creating short (*Limitations to RP – time: time* AND [limit* OR less OR hard*]*) and some creating complex rules (*Mechanism – watching the teaching of colleagues: teaching* AND [colleagues OR others] AND ["learn* from" OR people* OR technique*]*). We also saw examples of generic rules, which could only be used to navigate through a document, rather than provide accurate suggestions (*Motivation – to be good at job: good* AND job**). While none of our participants were experienced with rule-based coding of qualitative data, it would be interesting to evaluate the impact of such experience on the interaction with code rules. Better initial results might create a positive reinforcement loop, reducing barriers for engaging with rule-based suggestions while fostering a positive perception of the tool. Overall, users were able to define rules that helped them to structure and, to some extend, speed-up certain parts of the coding process. Thus, this paper extends prior work by demonstrating how users interact with code rules as coding support. With our work, we deliver new design implications for systems that integrate code rules and rule suggestions.

Regarding ML suggestions, we had to work around the cold start problem. Previous work required a minimum of 100 positive examples for each code [47], while participants in our evaluation, on average, only created 133 (MAXQDA) or 182 (Cody) positive examples overall. Our participant Kelly reported the most interaction with ML suggestions[9], while others barely noticed them. We believe that the barriers we set for Cody to providing ML suggestions, namely defining cut-off values for prediction confidence and requiring labels to be predicted correctly for all test instances, helped filter out many wrong suggestions. In the summative evaluation, Cody trained the first ML model after participants made ten annotations and triggered model retraining after every ten subsequent changes. Further, artificial negatives allowed the model to determine a section to be neutral and to refrain from making a

[9] For Kelly, the metrics of the last retraining of the model were: (Precision) 0.82, (Recall) 0.81, (F1-Score) 0.81, when including artificial 'greygoo' negative examples. Without them, metrics were: (Precision) 0.50, (Recall) 0.38, (F1-Score) 0.42. For training, 144 positive examples and 751 artificial negative examples were used. This training/prediction cycle resulted in 13 new suggestions for 4 labels that exceeded the cut-off.

suggestion. Participants perceived suggestions based on code rules as more helpful than ML suggestions. The strict quality criteria resulted in users interacting with only a low number of ML suggestions due to the number of positive examples necessary for the algorithm to make appropriate suggestions. Our results and Cody's ability to extend coding more frequently to sections that do not match a code rule could be improved by harnessing strategies for tuning the ML model during usage. For example, Cody could allow the user to adjust cut-off value(s) for rule-based and ML suggestions. Overall, we expect ML suggestions to assist coders with improving code rules by identifying *false negatives* – sections that are not yet covered by a rule despite belonging to the underlying label. Enabling users to define perfect rules would eliminate the need for ML suggestions altogether (but might not be feasible given the costs involved in and practicality of defining ideal rules for certain qualitative research methods and data structure [10]).

We calculated Krippendorff's Alpha to evaluate the coding consistency between our users, both for MAXQDA (0.085) and for Cody (0.332). As for the interpretation of an Alpha of 0.33, Kirppendorff suggests discounting conclusions from coding with an Alpha < 0.67 [22]. Depending on the type of qualitative research, an Alpha of 0.33 can indicate that researchers/coders should discuss and improve the codebook in use. In the context of our study, using Cody resulted in an increased Alpha compared to MAXQDA despite including an additional coder in the calculation. While our experiment setup does not allow us to determine the cause of the difference in Krippendorff's Alpha, the result may provide a quantitative indication that supports our qualitative findings. We believe the difference to have two causes. One, as participants engaged with code rules and ML suggestions, they spend more time reflecting on their coding and going back and forth in the document to review suggestions, potentially also revising previous annotations. Two, Cody makes suggestions at the sentence level, which might have influenced the unit of analysis that participants used for annotations. While with MAXQDA, participants applied codes at various units (individual words – multiple paragraphs), participants using Cody quite frequently applied their codes on the sentence level, too. Thus, the way a system provides suggestions may influence how users code.

## 5.2 Researcher Agency and Reporting

While automated suggestions may serve as proxies for the second coder, they can impact researchers' agency. Especially participants with MAXQDA stated concerns whether automated suggestions could impact coding quality, as coders would be tempted to accept suggestions to reduce their workload. As Cody's users told us that they rarely interacted with explanations, they are at risk of not realizing when a decision by the algorithm bases on incorrect or shallow assumptions (e.g., *higher* being an indicative word for the code *higher education*). However, participants felt responsible for the quality of their coding, and it was vital for them to get results that they can reliably use for subsequent analysis. One path to reduce the risk of carelessly accepting suggestions is to reduce the precision of suggestions by either: one, suggest not one but multiple labels, and have the coder pick the most appropriate one. However, this approach would increase the time it takes to review suggestions. Two, suggest labels only when an annotation is made, rather than preemptively annotating sections in the text (e.g., in the context of semantic annotations, see [43]).

Regarding trust and agency, it also needs to be discussed where calculations are performed, be it for applying rules to documents or training an ML model on data. Qualitative data may contain sensitive information, and researchers might not always anonymize their data prior to coding. Thus, the user of an assistive system must have control over where data is processed and stored, and can ideally run the system on their device or environment. Finally, researchers will only use systems for their projects that are accepted by their respective communities. Participants told us that they would not risk their work being rejected due to reviewers not being familiar with a new QDAS, particularly when authors would have to explain the tool's suggestion algorithm. While researchers would have to take responsibility for the suggestions they accept during coding, we believe that defining code rules can increase transparency in qualitative research projects, both for co-coders, as for reviewers and other researchers. While code rules may not communicate all information that determines the application of a code, they can serve as an indication towards coding and allow, to some extent, the replication of results.

## 5.3 Limitations and Future Work

This work can be improved in several ways. First, participants in the summative evaluation worked with data they had never seen before. Additionally, we told them that coding would take approximately 8 hours. Therefore, the evaluation results regarding coding time can serve only as an indication of the effects of interacting with suggestions. Secondly, participants did not use their coding after the experiment, giving them little incentive to code to the best of their ability. However, we ensured that they did not know whether they would be asked questions about the content or their coding during the final face-to-face interview. The interviews used for coding had roughly 18.600 words, which some participants perceived as too little to make use of automation appropriately. It would be interesting to test Cody *in the field* with the researchers' projects, where researchers deal with more data without an estimation of how long coding will take. A field evaluation can also help us address other limitations of the current study: We explicitly encouraged participants to add their codes to the provided codebook if necessary. While instructions and codebook provided participants with a common coding goal, it may have restricted participants in applying their coding style. Further, the presented results on intercoder reliability are illustrative only for the codebook research method. A field evaluation would allow us to evaluate Cody's impact on other kinds of qualitative research – we expect that the utility of code suggestions might shift towards assisting in uncovering ideas and themes during codebook development. For some coding strategies (e.g., in-vivo coding, as Sven mentioned), the utility of code suggestions may be limited. Thirdly, our strategy to creating artificial negatives assumes that users code linearly from top to bottom, and rarely miss important sections during coding. Further, when using rule suggestions for model training, imprecise or wrong rules can cause errors to propagate, resulting in wrong ML suggestions. In the end, the amount of available training data limits the quality of ML-based suggestions. Participants with Cody made, on average, 182 annotations for 38 labels, resulting in a very spare training set. While we improved our ML model(s) through greygoo labels and one-versus-rest training, the quality of ML-based suggestions during our evaluation was limited. However, our aim was not to improve model training in a cold start case but to understand how participants interacted with ML suggestions. Our results indicate that with artificial negatives, learning from rule suggestions, and careful filtering, ML-based suggestions can be used even in a cold start case with sparse training data. An avenue for future work is to evaluate different data collection strategies for cold start model training. Integrating other technologies to recognize sections that coders intentionally did not annotate, such as eye tracking, could be an exciting research opportunity [44]. Further, participants coded the same documents predominantly using the same codebook, yet we trained the ML model individually for each user. Training a shared model on the examples from multiple coders could increase the quality of ML-based suggestions. Finally, this study focused on each coder working on an individual copy of the data. Integrating and evaluating mechanics for multiple coders to collaborate in coding documents could extend this work. It would be interesting to observe whether formulating rules can help multiple coders discuss their interpretation of labels and how coders work with suggestions based on their co-coder's code rules.

## 6 CONCLUSION

Inspired by previous work concerning AI-based qualitative coding, we set out to understand how real users interact with automated suggestions during coding. We designed and developed Cody, an interactive AI-based system supporting researchers with rule- and ML-based suggestions. We worked with qualitative researchers to iterate our designs, finding that given the right assistance and interface, end-users would (re)define rules, convinced that it would help to improve their understanding, build stringent codebooks, and accelerate their coding. Based on our findings, we conducted a one-week experiment, comparing the coding process of qualitative researchers with MAXQDA and Cody when coding a public dataset of interviews. We found that code rules provide both structure and transparency, particularly when coding new data. Explanations for suggestions are commonly desired but rarely used, and perceived quality rather than confidence scores convince users. Finally, working with Cody (for now) benefits coding quality rather than coding speed, increasing the intercoder reliability, calculated with Krippendorff's Alpha, from 0.085 (MAXQDA) to 0.33 (Cody).

## 7 ACKNOWLEDGMENTS

We are indebted to our participants for their time and expertise. We thank our colleagues for their participation in various pre-tests of Cody and our anonymous reviewers for valuable feedback on drafts.

## REFERENCES

[1] Ahmed Abbasi. 2016. Big Data Research in Information Systems: Toward an Inclusive Research Agenda. *Journal of the Association for Information Systems* 17, 2 (2016), 1–32. https://doi.org/10.1017/CBO9781107415324.004

[2] Giuseppe Abrami, Andy Lücking, Alexander Mehler, Elias Rieb, and Philipp Helfrich. 2019. TEXTANNOTATOR : A flexible framework for semantic annotations. In *Proceedings of the 15th Joint ACL - ISO Workshop on Interoperable Semantic Annotation (ISA-15)*. Association for Computational Linguistics, London, UK, 1–12.

[3] Tehmina N. Basit. 2003. Manual or electronic? The role of coding in qualitative data analysis. *Educational Research* 45, 2 (2003), 143–154. https://doi.org/10.1080/0013188032000133548

[4] Peter M. Bednar and Christine Welch. 2009. Contextual Inquiry and Requirements Shaping. In *Information Systems Development*. Springer International Publishing, New York, NY, USA, 225–236. https://doi.org/10.1007/978-0-387-68772-8{_}18

[5] Matt Chaput. 2020. Whoosh. https://whoosh.readthedocs.io/en/latest/index.html

[6] Nan Chen Chen, Margaret Drouhard, Rafal Kocielnik, Jina Suh, and Cecilia R. Aragon. 2018. Using machine learning to support qualitative coding in social science: Shifting the focus to ambiguity. *ACM Transactions on Interactive Intelligent Systems* 8, 2 (2018), 1–21. https://doi.org/10.1145/3185515

[7] Nan-chen Chen, Rafal Kocielnik, Margaret Drouhard, Jina Suh, Keting Cen, Xiangyi Zheng, Cecilia R. Aragon, and Vanessa Pena-Araya. 2016. Challenges of Applying Machine Learning to Qualitative Coding. In *ACM SIGCHI Workshop on Human-Centered Machine Learning*. Association for Computing Machinery, New York, NY, USA, 6. http://hcml2016.goldsmithsdigital.com/program/

[8] Hao Fei Cheng, Ruotong Wang, Zheng Zhang, Fiona O'Connell, Terrance Gray, F. Maxwell Harper, and Haiyi Zhu. 2019. Explaining decision-making algorithms through UI: Strategies to help non-expert stakeholders. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300789

[9] Edward Collins, Nikolai Rozanov, and Bingbing Zhang. 2019. LIDA: Lightweight Interactive Dialogue Annotator. (2019), 121–126 pages. https://doi.org/10.18653/v1/d19-3021

[10] Kevin Crowston, Eileen E. Allen, and Robert Heckman. 2012. Using natural language processing technology for qualitative data analysis. *International Journal of Social Research Methodology* 15, 6 (2012), 523–543. https://doi.org/10.1080/13645579.2011.625764

[11] Kevin Crowston, Xiaoxhong Liu, and Eileen E. Allen. 2010. Machine Learning and Rule-Based Automated Coding of Qualitative Data. *proceedings of the American Society for Information Science and Technology* 47, 1 (2010), 1–2. https://crowston.syr.edu/content/machine-learning-and-rule-based-automated-coding-qualitative-data

[12] Carla Azevedo De Almeida, Fabio Freitas, Antonio Pedro Costa, and Antonio Moreira. 2019. WEBQDA: The Quest for a Place in the Competitive World of CAQDAS. In *Proceedings of the 2019 International Conference on Engineering Applications (ICEA)*. IEEE, New York, NY, USA, 1–7. https://doi.org/10.1109/CEAP.2019.8883456

[13] Margaret Drouhard, Nan Chen Chen, Jina Suh, Rafal Kocielnik, Vanessa Pena-Araya, Keting Cen, Xiangyi Zheng, and Cecilia R. Aragon. 2017. Aeonium: Visual analytics to support collaborative qualitative coding. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, New York, NY, USA, 220–229. https://doi.org/10.1109/PACIFICVIS.2017.8031598

[14] Jeanine C. Evers. 2018. Current issues in qualitative data analysis software (QDAS): A user and developer perspective. *Qualitative Report* 23, 13 (2018), 61–73.

[15] Mingming Fan, Yue Li, and Khai N. Truong. 2020. Automatic Detection of Usability Problem Encounters in Think-aloud Sessions. *ACM Transactions on Interactive Intelligent Systems* 10, 2 (2020), 1–24. https://doi.org/10.1145/3385732

[16] Fábio Freitas, Jaime Ribeiro, Catarina Brandão, Francislê Neri de Souza, António Pedro Costa, and Luís Paulo Reis. 2018. In case of doubt see the manual: A comparative analysis of (self)learning packages qualitative research software. *Advances in Intelligent Systems and Computing* 621 (2018), 176–192. https://doi.org/10.1007/978-3-319-61121-1{_}16

[17] Abbas Ganji, Mania Orand, and David W. McDonald. 2018. Ease on Down the Code: Complex Collaborative Qualitative Coding Simplified with 'Code Wizard'. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW, Article 132 (Nov. 2018), 24 pages. https://doi.org/10.1145/3274401

[18] Justin Grimmer and Brandon M. Stewart. 2013. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis* 21, 3 (2013), 267–297. https://doi.org/10.1093/pan/mps028

[19] Jamie Harding. 2015. *Identifying Themes and Coding Interview Data: Reflective Practice in Higher Education*. SAGE Publications Ltd., London, UK. https://doi.org/10.4135/9781473942189

[20] Neringa Kalpokaite and Ivana Radivojevic. 2018. Best practice article: Autocoding and Smart Coding in ATLAS.ti Cloud. https://atlasti.com/2018/09/27/auto-coding-and-smart-coding-in-atlas-ti-cloud/

[21] Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych. 2018. The INCEpTION Platform: Machine-Assisted and Knowledge-Oriented Interactive Annotation. In *Proceedings of the International Conference on Computational Linguistics*. Association for Computational Linguistics, Santa Fe, New Mexico, USA, 5–9. http://tubiblio.ulb.tu-darmstadt.de/106270/https://www.youtube.com/watch?v=Xz3Hs8Lyoeghttps://inception-project.github.io/publications/

[22] Klaus Krippendorff. 2004. *Reliability in Content Analysis: Some Common Misconceptions and Recommendations*. Technical Report. University of Pennsylvania. http://repository.upenn.edu/ascpapers/242

[23] C Lejeune. 2011. An Illustration of the Benefits of Cassandre for Qualitative Analysis. *Forum: Qualitative Sozialforschung = Forum: Qualitative Social Research [FQS]* 12, 1 (2011), 19. https://doi.org/10.17169/12.1.1513

[24] Seth C Lewis, Rodrigo Zamith, and Alfred Hermida. 2013. Content Analysis in an Era of Big Data. *Journal of Broadcasting & Electronic Media* 57, 1 (2013), 34–52. https://doi.org/10.1080/08838151.2012.76170

[25] Aron Lindberg. 2020. Developing theory through integrating human and machine pattern recognition. *Journal of the Association for Information Systems* 21, 1 (2020), 90–116. https://doi.org/10.17705/1jais.00593

[26] Megh Marathe and Kentaro Toyama. 2018. Semi-automated coding for Qualitative research: A user-centered inquiry and initial prototypes. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'18)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3173922

[27] MAXQDA. 2020. Keyword-in-Context | MAXQDA. https://www.maxqda.de/hilfe-mx20-dictio/keyword-in-context

[28] Nancy McCracken, Jasy Suet Yan Yan, and Kevin Crowston. 2014. Design of an Active Learning System with Human Correction for Content Analysis. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*. Association for Computational Linguistics, Stroudsburg, PA, USA, 59–62. https://doi.org/10.3115/v1/W14-3109

[29] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. 2019. Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW, Article 72 (2019), 23 pages. https://doi.org/10.1145/3359174

[30] Tetsuya Nakatoh, Satoru Uchida, Emi Ishita, and Toru Oga. 2016. Automated generation of coding rules: Text-mining approach to ISO 26000. In *Proceedings - 2016 5th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2016*. IEEE, New York, NY, USA, 154–158. https://doi.org/10.1109/IIAI-AAI.2016.210

[31] Nvivo. 2020. NVivo 11 - Automatic coding in document sources. http://help-nv11.qsrinternational.com/desktop/procedures/automatic_coding_in_document_sources.htm

[32] Pablo Paredes, Ana Rufino Ferreira, Cory Schillaci, Gene Yoo, Pierre Karashchuk, Dennis Xing, Coye Cheshire, and John Canny. 2017. Inquire: Large-scale early insight discovery for qualitative research. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work, CSCW*. ACM, New York, NY, USA, 1562–1575. https://doi.org/10.1145/2998181.2998363

[33] Michael Quinn Patton. 2002. *Qualitative Research & Evaluation Methods*. Sage Publications Ltd., London, UK. 342 pages. http://books.google.com/books/about/Qualitative_research_and_evaluation_meth.html?id=FjBw2oi8El4C

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. (2011), 2825–2830 pages. https://scikit-learn.org/stable/index.html

[35] Lyn Richards. 2002. Qualitative computing—a methods revolution? *International Journal of Social Research Methodology* 5, 3 (2002), 263–276. https://doi.org/10.1080/13645570210146302

[36] Tim Rietz and Alexander Maedche. 2019. LadderBot: A Requirements Self-Elicitation System. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, New York, NY, USA, 357–362. https://doi.org/10.1109/RE.2019.00045

[37] Tim Rietz and Alexander Maedche. 2020. Towards the Design of an Interactive Machine Learning System for Qualitative Coding. In *Proceedings of the 41st International Conference on Information Systems (ICIS 2020)*. AIS, New York, NY, USA, 9. https://doi.org/10.5445/IR/1000124563

[38] Jane Ritchie and Jane Lewis. 2003. *Qualitative Research Practice: A guide for social science students and researchers*. Vol. 1. SAGE Publications Ltd., London, UK. 336 pages. https://doi.org/10.18352/jsi.39

[39] Stefan Schrunner, Bernhard C. Geiger, Anja Zernig, and Roman Kern. 2020. A generative semi-supervised classifier for datasets with unknown classes. In *Proceedings of the ACM Symposium on Applied Computing*. ACM, New York, NY, USA, 1066–1074. https://doi.org/10.1145/3341105.3373890

[40] Stephen Soderland. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning* 34, 1 (1999), 233–272. https://doi.org/10.1023/A:1007562322031

[41] Kazuko Takahashi, Hiroya Takamura, and Manabu Okumura. 2005. Automatic occupation coding with combination of machine learning and hand-grafted rules. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3518 LNAI, May 2005 (2005), 269–279. https://doi.org/10.1007/11430919{_}34

[42] Ella Tallyn, Hector Fried, Rory Gianni, Amy Isard, and Chris Speed. 2018. The Ethnobot: Gathering ethnographies in the age of IoT. *Conference on Human Factors in Computing Systems - Proceedings* 2018-April (2018), 1–13. https://doi.org/10.1145/3173574.3174178

[43] Tabea Tietz, Joscha Jäger, Jörg Waitelonis, and Harald Sack. 2016. Semantic annotation and information visualization for blogposts with refer. *CEUR Workshop Proceedings* 1704 (2016), 28–40.

[44] Peyman Toreini, Moritz Langner, and Alexander Maedche. 2020. Using eye-tracking for visual attention feedback. In *Lecture Notes in Information Systems and Organisation*, Vol. 32. Springer International Publishing, New York, NY, USA, 261–270. https://doi.org/10.1007/978-3-030-28144-1{_}29

[45] Gregor Wiedemann. 2013. Opening up to big data: Computer-assisted analysis of textual data in social sciences. *Historical Social Research* 38, 4 (2013), 332–358. https://doi.org/10.12759/hsr.38.2013.4.332-358

[46] Ziang Xiao, Michelle X. Zhou, Q. Vera Liao, Gloria Mark, Changyan Chi, Wenxi Chen, and Huahai Yang. 2020. Tell Me About Yourself: Using an AI-Powered Chatbot to Conduct Conversational Surveys with Open-Ended Questions. *ACM Transactions on Computer-Human Interaction* 27, 3, Article 15 (June 2020), 37 pages. https://doi.org/10.1145/3381804

[47] Jasy Liew Suet Yan, Nancy McCracken, Shichun Zhou, and Kevin Crowston. 2014. Optimizing Features in Active Machine Learning for Complex Qualitative Content Analysis. In *Proceedings of the ACL Workshop on Language Technologies and Computational Social Science*. Association for Computational Linguistics, Baltimore, MD, USA, 44–48. https://doi.org/10.3115/v1/w14-2513

[48] Seid Muhie Yimam, Chris Biemann, Richard Eckart de Castilho, and Iryna Gurevych. 2014. Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Association for Computational Linguistics, Baltimore, MA, USA, 91–96. https://doi.org/10.3115/v1/P14-5016

[49] Seid Muhie Yimam, Chris Biemann, Ljiljana Majnaric, Šefket Šabanović, and Andreas Holzinger. 2015. Interactive and Iterative Annotation for Biomedical Entity Recognition. In *Proceeding of Brain Informatics and Health (BIH 2015)*, Vol. 9250. Springer International Publishing, New York, NY, USA, 347–357. https://doi.org/10.1007/978-3-319-23344-4{_}34