ISSN: 2473-4901
v10 n6205

Training a large language model to code qualitative research data: Results from discussions of ethical issues

David M. Simmonds
dsimmond@aum.edu
Management Information Systems
Aburn University at Montgomery
Montgomery, AL 36117

Russell Haines
hainesrp@appstate.edu
Computer Information Systems
Appalachian State University
Boone, NC, 28608, USA

Abstract

Comment coding is an important part of qualitative research, but it is a labor intensive process. Furthermore, researchers need to assess whether or not comments were coded accurately and reliability. Here, we present a process for arranging the original comments and using them to train a Google BERT large language model (LLM) that was able to code comments with 73% reliability. This process can be extended by future researchers to potentially code comments made in less-structured research settings, or potentially have the LLM create the comment groupings automatically.

Keywords: transformer model, attention mechanism, text analytics, qualitative research

1. INTRODUCTION

In qualitative research, coding data is an important part of the data analysis process (Sarker et al., 2013). Classifying answers to open ended questions is inherently challenging and humans make mistakes and have disagreements about which label to be applied to a given comment (Faraj et al., 2015). Whenever coding of text is involved, raters are expected to operate reliably, meaning they code similar responses in a similar way, every time. The Kappa statistic measures Inter-rater reliability and is used to determine whether there is an acceptable level of match between coders (Haines et al., 2014). An acceptable level of Kappa is considered substantial at .61 according to McHugh (2012).

In this paper, we examine the research question: Can a large language model be trained to code qualitative data in a reliable way? We use the data from Haines et al. (2014) as a training and evaluation set. In that study, they coded comments in discussions about whether actions were ethical or not. Here, the focus is turned to the use of a Large Language model (LLM) for comment coding to determine whether a LLM could provide a comparable reliability to human coders.

In the following, we report both the methods and the results of our LLM training with the idea that other researchers can use the same or similar techniques. Ultimately, our model was able to achieve 73% agreement with the human coder, which is quite good considering that the model

ISSN: 2473-4901
v10 n6205

could not be expected to perform better than the consensus between human beings, which in Haines et al. was 89% with a Kappa of .71.

TRANSFORMERS & the BERT Model

In the seminal paper "Attention is all you need" (Vaswani et al., 2017), Google introduced the Transformer architecture which has revolutionized Natural Language Processing. In 2018, Open AI took the transformer model and split it in half to focus on text generation. This only required the decoder-the second half of the transformer. These foundation models have been pretrained for weeks of thousands of graphical processing units (GPUs) to understand human language and general knowledge along with some domain knowledge. They have gone on to be embedded in Google Colab for code prediction and Co-pilot in Microsoft Office products. Before transformer models, the AI landscape was dominated by models like the TF-IDF bag of words model for classifying text, or basic neural networks such as RNNs for text generation.

The transformer is made up of an encoder and decoder model. The encoder model makes statistical sense and creates a representation of the patterns and relationships between words and concepts inside the text fed to it. The decoder is then able to generate text from that representation. The big improvement introduced by the transformer model is the ability of the model to remember the relationships between words far away from each other and create more new relationships. Limited only by the size of the input text, transformers can make a map of the strength of the connection between every single word and every other word in the input. This makes them extremely memory hungry, but at the same time, they never forget the relationships. Additionally, transformers are fed word embeddings, which are multi-dimensional vectors of size, 512, 768, 1024 or more. Each position in the vector captures some aspect of the meaning of a word, such that words which mean similar things like dog and puppy have very similar embeddings. Additionally, the difference between words is captured, such that the vectors representing Washington D.C. and USA will have a mathematically similar difference to the vectors representing Berlin and Germany.

For this study, we use Google's BERT transformer model (Devlin et al., 2019). The core of the transformer model is the attention mechanism. There are 3 main types of attention mechanisms: self-attention, multi-head attention and scaled-dot-product attention. Figure 1 shows a diagram which illustrates the attention mechanism and reflects the history and original purpose of transformer models which was language translation. In the diagram, we can see English words with their French equivalent. French and other languages cannot be translated one word at a time, because equivalent words are placed at different relative positions in the sentence. The attention mechanism allows transformers to create a mapping (weights) which indicate the word in the translated language that the translator should pay attention to when translating words from the original language.

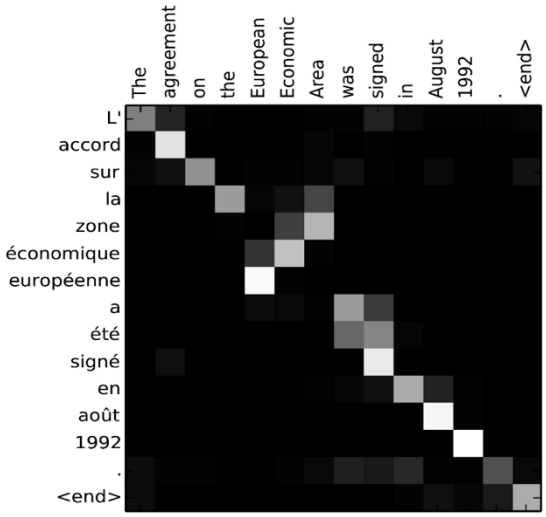

Figure 1: Attention mechanism showing mapping of French and English sentences

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key (Devlin et al., 2019). The attention mechanism used in the BERT model is called "Scaled Dot-Product Attention" (see Figure 2). The input consists of queries (Q) and keys of dimension (K), and values of dimension (V), and the dot products of the three are computed.

ISSN: 2473-4901
v10 n6205

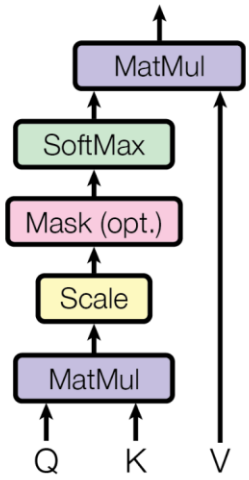

Figure 2: Scaled dot-product attention mechanism reproduced from Vaswani et al. (2017)

In the case of comment classification, attention similarly tells the classifier which words or phrases to pay attention to in order to predict the correct label for the comment. An example comment in our dataset is "just because it's misleading doesn't mean its unethical". Before transformers, using techniques like bag-of-words and TF-IDF models, the classifier would see the word "misleading" and interpret it as "unethical". However, the BERT model has been trained on English language and generally knows that any phrase between "just because" and "doesn't mean" implies that the words in between these two phrases do not determine the overall intent of the sentence. And in fact, the speaker usually means the opposite of the words which come after "doesn't mean" – in this case "it's unethical". So, a transformer-based classifier knows to pay less attention to the fact that the action is misleading because of its model weights; thus, the overall meaning taken from the sentence is the opposite of the next words "it is unethical". These associations (model weights) are then reinforced in the model's fine-tuning on the training dataset since the target label provided in the training dataset indicates that the student was giving support for the action being ethical.

2. METHODOLOGY

DATASET

The Haines et al. (2014) dataset was collected during an experimental study of ethical decision making. A total of 219 participants in 42 groups of either five or six members met virtually in chat rooms for three or four minutes to discuss the ethics of five different marketing scenarios. Their final dataset consists of 5,955 "thoughts," which are the comments made by participating students during the online discussions. All of the comments made during the discussions were human coded via a standardized coding sheet according to their contents. Here, we give the codes and their titles, but readers are encouraged to consult the original article if they wish a more extensive explanation of the codes and their definitions. The codes are: supportive remark (SR) stating that the behavior was ethical, a supportive argument (SA) giving reasons why the behavior was ethical, critical remark (CR) stating that the behavior was unethical, critical argument (CR) giving reasons why the behavior was unethical, compromise or accept part of others opinions (CP), neural remark or neural response to other's opinion (NR), agreement on others' opinions (AO), disagreement on other's opinion (DO), query for clarification or explanation (QC), query for solution (QS), answer to questions (AN), comments on related topic but off track (OT), summary of consensus (SU), uncoded text (UC), humorous comments (HU), off topic comments (OF). Although Haines et al. (2014) reports two coders, only one of which coded all of the comments, the dataset we obtained has three coders, two of which coded all of the comments.

The standard for AI model accuracy is Bayes Optimal Error because AI cannot be expected to perform better than a group of human experts who agree. Therefore, comments with mismatched coding were not considered useful. In the original dataset, a few of the comments span over multiple lines of text, meaning that the student broke up a sentence into parts as they participated in the chat. This meant that some of the comments were unlabeled. Comments without a label are also not useful, since there is no ground truth label for the model to be evaluated on. Overall, from our training set, we excluded labels which were empty, uncoded or mismatched between human coders. We dropped those records, leaving 1820 records which had agreement between all 3 coders. 70% of the data (1274 records) were used for the model training, while 30% of the data was used as hold out validation/test sets (273 or 15% each). During hyper-parameter tuning, the validation set was used to determine accuracy of the model after each epoch. This was used for finetuning instead of the model loss since loss does not have a proportional or direct relationship with accuracy. The test set was used to test the final model's accuracy and also generate the predictions.

The code used to denote human coding consisted of the following. The codes themselves had an imbalanced representation of records in the

dataset as shown in Table 1. The training set has 312 critical arguments but only 2 that were coded QE, QR, or QP. The coding sheet does not contain either of the last three labels, but looking at the comments themselves, they seem to be miscoded queries that should have been coded either QC or QS. The code used for the label dictionary is as follows:

label_dict: {'SR': 0, 'SA': 1, 'CR': 2, 'OT': 3, 'QP': 4, 'CA': 5, 'CP': 6, 'AN': 7, 'QE': 8, 'OF': 9, 'COM': 10, 'QS': 11, 'QR': 12, 'QC': 13, 'NR': 14, 'HU': 15, 'SU': 16, 'AO': 17, 'DO': 18}

Rebalancing the dataset improved the accuracy. This came at the cost of stability of the results, since selecting a different training set and test set tended to change the **validation accuracy** slightly, since prediction of the codes with much less records tended to depend on which of the actual messages were included in the training set. But on average, accuracy increased by about 3% after the rebalancing.

Label	Number of Records
CA	312
EY	259
SA	239
AO	235
CR	209
OF	188
UC	89
NR	80
SR	79
CP	76
QC	39
HU	34
DO	30
QS	30
COM	29
OT	19
SU	15
AN	10
QE	5
QR	2
QP	2

Table 1: Number of records per comment label

The training data was rebalanced by oversampling to become a dataset of 2,000 records with at least 50 records for each label, so the model could get adequate exposure and training to each label. Validation and test sets are not oversampled since they can produce unrealistically high accuracy figures, due to overfitting of the model on the smaller labels which have high consistency due to their repetition.

Columns

The columns used for training included Comment, label, Sequence-# and Scenario-# which was replaced with scenario description. The sequence number was used to reset the sliding window of comments, so that when the sequence # changed to 1, all past comments were erased so that the model would consider only comments related to the particular conversation thread.

Model Training

BERT is a foundation model which can be trained on 2 objectives. The first is next-sentence-prediction. In order to finetune BERT for this objective, it is trained on a dataset which has 2 sentences and a label indicating whether the 2nd sentence follows from the first, or not. The second objective is masked word prediction. BERT can be trained to discover which word is missing (masked) in a sentence in a manner similar to fill in the blank questions given to students on an exam. This makes it suitable for classification of sentences since the comments in this study are responses to the previous comments

Tokenization

Large Language models cannot process text. They ingest numbers which represent each word (token) in the text fed to it. Hence tokenization is a necessary first step to process text. The dataset was tokenized using the BERT base uncased tokenizer to create model readable tokens. Both model and tokenizer are hosted on Hugging face (www.huggingface.com) and the correct tokenizer is automatically loaded when the checkpoint for the model is used.

Each word or sub-word which can be understood by the model, is part of the tokenizer's vocabulary. Each of these words is represented by a number, from 0 to the vocabulary size (-1). First each sentence is split into words separated by spaces. Sentence1 and Sentence 1 are represented by tokens, numbers which represent each word in the model. When the model predicts a next token, it uses a feedforward output layer with an output head the size of the number of possible predictions. In a text generation LLM, that would be the size of the vocabulary, in the order of 10s of thousands of predictions.

Padding is added to make all the input tokens the same length, since the model can only ingest

ISSN: 2473-4901
v10 n6205

rectangular batches--having the same number of tokens. Truncation is allowed in order to ensure that sentences that are too long for the 512 token limit are reduced to fit. Sentences are then batched which speeds up training since the GPUs can parallelize the calculations.

Token types are either 0 or 1. 0 indicates that the tokens belong to the first sentence while 1 indicates that the tokens are taken from the second sentence.

Label: is the numeric representation of the code given to the current comment. In this case, since we are only predicting 22 labels, the feedforward layer only has 22 outputs. The tokenizer can only use numbers as labels, so the program creates a dynamic dictionary of codes present in the training dataset. This dictionary automatically changes when different codes are left out of the training set. In order to test the model and create predictions which are human-readable, the dictionary is used to translate the codes back to the original labels.

Input_ids: are the actual tokens representing each word, as described above.

Token-type-ids: allow the model to distinguish between the two sentences fed to it. Every token belonging to the first sentence is assigned a 1. Tokens in the second sentence are assigned a 0.

Attention Mask: indicates which tokens are actual words and which are zeroes added for padding. The model knows to ignore the words used only for padding.

After Tokenization, renaming the label column as 'labels' and removing the columns which will not be used for training, the dataset looks like this:


```
{'labels': tensor(5),
 'input_ids': tensor([
 101,  1001,  1001,  1001,  2917,  200
3,  1037, 11967,  2445,  2000,
 2493,  1012,  2027,  2020,  2356,  20
00,  7615,  2006,  3251,  1996,
  11967,  2001, 12962,  2030,  2025,
1001,  1001,  1001,  1996, 12698,
  2005,  1037,  2275,  1997, 10899,  2
881,  2000,  5574,  2000,  1021,
  1011,  2000,  2340,  1011,  2095,  1
011, 19457,  2024,  3491,  2076,
   13941,  1998,  2060,  2336,  1005,
1055,  3454,  1012,  2122, 12698,
  3444,  1037,  2440,  1011,  4094,  6
579,  2686,  2103,  2007,  2929,
  1998,  3311,  1997,  5200,  2521,  3
458,  2008,  2029,  6526,  1999,
  1996, 10899,  1012,  1001,  1001,  1
001,  2917,  2024,  1016,  7928,
  2029, 16218,  2094,  1996, 11967,  1
999,  3408,  1997,  3251,  2030,
  2025,  1996, 11967,  2001, 12962,  1
012,  1001,  1001,  1001,  7615,
  1011,  1015,  1024,  1045,  5993,  1
012,  7615,  1011,  1016,  1024,
  1045,  2228,  2023,  6433,  1037,  2
843,  1012, 102,  1001,  1001,
  1001,  2917,  2003,  1996,  6745,  7
615, 14120,  2000,  1996,  3025,
   11967,  1012,  1001,  1001,  1001,
1045,  2123,  2102,  2228,  2017,
  2064,  2655,  2023,  2428, 16655, 23
048,  2389,  1999,  2026,  5448,
  1001,  1001,  1001,  2241,  2006,  1
996,  2126,  8280,  1996, 11967,
  1998,  3025,  7928,  1010,  3443,  1
037,  3830,  2005,  1996,  6745,
  7615,  1012, 102,    0,    0,    0,    0
,    0,    0,    0,
   0,    0,    0,    0,    0,    0,    0,
0,    0,    0,
   0,    0,    0,    0,    0,    0,    0,
0,    0,    0,
   0,    0,    0,    0,    0,    0,    0,
0,    0,    0,
   0,    0,    0,    0,    0,    0,    0,
0]),
 'token_type_ids': tensor([
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
),
 'attention_mask': tensor([
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
  1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
}
```

All LLMs have a maximum input size, which is 512 in the case of BERT. This roughly means the total number of words in the input sentence(s) should not exceed 512 words. But it also does sub-word tokenization where it breaks down a word such as ethically into subwords like ethic + al + ly. This allows the tokenizer to represent a bigger vocabulary with less tokens, but eats into the input size allowed. It also uses start of sentence and end of sentence tokens, along with tokens for punctuation, so that 512 is not as much as it may sound. In order to accommodate this limitation, the scenario and actions were summarized so that the scenario description along with comments was able to fit into the tokenizer's max length of 512 tokens.

The model is then finetuned (trained) on the dataset, over a number of epochs (passes through the dataset). Model training consists of selecting several parameters including Learning Rate, Gamma (rate of reduction of learning rate) and optimizer step size (the number of steps taken down the gradient before learning rate is reduced by multiplying by Gamma.

Model

The model used was a pretrained model from HuggingFace.com. It was downloaded as a model checkpoint: "*google-bert/bert-base-uncased*". The tokenizer was automatically selected from hugging face using the from_pretrained method to ensure compatibility with the model. The size of the model on disk is only 536 MB, but in training, especially depending on the batch size, the memory requirements for the model alone grows 16 to 20 times, easily taking up 10Gb of GPU memory.

Coding Environment

Google Colab was used as the coding environment, which has a variable watch window, code completion, code prediction, and code generation. These features were instrumental in generating code to create a balanced dataset by oversampling the rarer labels (such as QP, QR, QE and AN which only had 2,2,5 & 10 records respectively, compared to CA which had 223 records respectively. The prompt to generate this was:

> *# prompt: split the InstructTuning_df dataframe into a random sample of 70% train and 30% validationtest. Then generate extra records to oversample the Train set to create a dataset that is balanced by label. Then split the validation test into equal parts. Show the label_column distribution for all 3 resulting datasets.*

Model Training

Google Colab also gives access to GPUs which are necessary for parallelizing the layers inside the transformer model used by BERT. An L4 GPU on Google Colab with 22Gb of GPU memory was able to accommodate 64 records per batch of training data, training the model in 25 seconds per epoch. With 20 epochs taking between 9 and 10 minutes.

The concept of LLM "loss" is based on statistics, being the difference between a set of predictions made by the model and the actual values in the dataset. It is founded in in mathematical optimization and statistics. In statistical regression, the software tries several lines of best fit to match the datapoints. It then calculates loss using ordinary least squares (OLS) and accepts the solution with the lowest OLS. Training an LLM is very similar, since the training objective is to minimize the model loss. In LLM, the gradient descent is used to achieve the minimal loss-or best model. At the start of training, model weights (similar to regression coefficients) are randomly initialized. In each of the several "epochs", The model uses weights to generate predictions which are compared to target labels. The loss is then calculated and backward propagation occurs, in which a fraction of the loss is subtracted from the model weights. Trial and error is required to find the optimal hyperparameters (learning rate, gamma, step size). Several runs need to be done in order to determine the best combination of batch_size, epochs, learning rate, gamma and optimizer step size.

The model was eventually trained for 7 epochs with a learning rate of 5.1 e-5, optimizer step size of 1 and gamma of .81. We arrived at these parameters after adjusting the learning rate from 7, 6, 5, 4, 5.5 with several more tweaks and finally 5.1 (e-5). Gamma started at .99 and went to 0.91, 0.8, 0.7, 0.85 with several more adjustments and finally .81. The training graph is shown in Figure 3 below. The ShedulerLR optimizer was used, which allows for a decay in the training rate as it approaches the lowest loss. The learning rate needs to be sufficiently large in the beginning to achieve high accuracy. The loss decay is necessary because as the model approaches its lowest possible loss, it needs to creep to the minimum loss without overshooting. Hence an exponentially decreasing learning rate is required as provided by StepLR. The learning rate is multiplied by $Gamma^{TrainSteps}$ and since Gamma is < 1, the learning rate approaches zero.

ISSN: 2473-4901
v10 n6205

For example, with a Gamma of .82, on the 20th training step the learning rate would become 0.82^{20} = 2% of the initial learning rate.

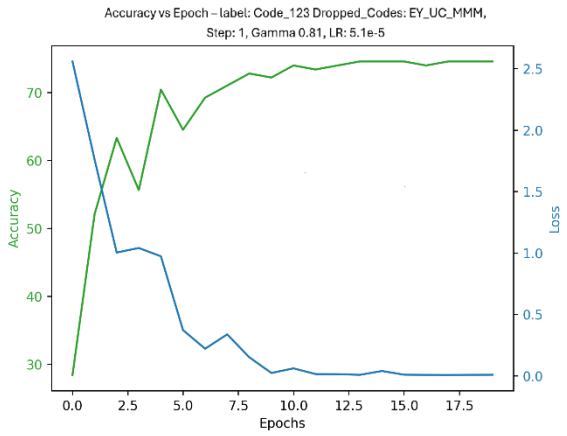

Figure 3: Training accuracy and gamma over training epochs

3. RESULTS

After training, a hold out dataset was used to test the overall accuracy of the model and generate predictions which could be compared by humans against the ground truth labels. A predicted label as well as a prediction confidence was produced. This allowed us to differentiate between comments that were labeled incorrectly with low confidence vs those with high confidence. We achieved a finetuned model with a training loss of .051 and validation accuracy of 73%. The model was used to predict the labels on our test set. González-Carvajal, & Garrido-Merchán (2020) categorized Portugese News items and achieved accuracy of 91%. The first dataset had 9 categories, equivalent to ours, but most articles had between 300 and 700 words, significantly longer than our messages. They also achieved a score of 83% on the categorization of tweets into real or non-real disasters. In the 2nd case, they used 10875 tweets multiples of our dataset and with only 2 categories compared to our 9.

4. LIMITATIONS AND CHALLENGES

The study would have benefited a lot from more data. Studies of this type typically have many thousands of rows of data, for example González-Carvajal et al (2020) which used 167,000 and 10,800 records. By increasing the data available, the classifier would be able to learn about the patterns. We had 5000 records but excluded the ones that the human beings did not agree on. This could have been an avenue of improvement. The length of the messages is also a big problem. Many messages were as short as "OK" which gives practically no indication what they were responding to. Possibly even more challenging is the fact that the messages were captured in a continuous stream, not as responses to any particular previous message. It is very possible for a message to be a response to one 20 messages ago, depending on typing speeds.

5. DISCUSSION AND CONCLUSIONS

The ability of any model to classify a set of comments will be heavily influenced by the initial training data. It means researchers should try to collect training data about many scenarios, from many different sources. A meta study would then be able to remove local and situation biases in attitudes to ethics from affecting the study.

LLMs are generally trained with the objective of next word prediction by minimizing the loss between predicted words and actual words given to it in a training dataset. BERT is special in that it is trained to also do next sentence prediction: whether a second sentence follows from a first. To allow BERT to achieve this, a special classification layer is added to the other layers to allow for classification (Devlin et al., 2019). The model can then be finetuned on pairs of sentences as in our case. Though the foundation model was trained with a classification head that produces a 0 or 1: whether or not sentence-2 follows sentence-1, it can be finetuned to produce many classes, given the right number of training data. In our case, this was 9 classes.

Whereas a regular LLM uses an input sentence to predict the next word, our code and configuration meant BERT was essentially using 2 sentences to produce a next word – which in our case is the comment label. Another analogy might be a conversation between a person and an LLM: the person says "Hi", the regular LLM would be limited to saying "Hello". Our BERT model takes this a step further to act as an observer between the two people in the conversation. Supposing it is finetuned on a conversation between two polite speakers. It is then asked to assess 2 sentences -- Sentence-1: "Hi, how are you?", Sentence-2: Hello I am well, how about you?" BERT would predict a "1" (yes 2nd sentence follows from the 1st). If it were fed a conversation in which one impolite person is involved. Sentence-1: "Hi, how are you?", Sentence-2: "We need to go to lunch right now because I am hungry?" it should predict a 0 (no 2nd sentence does not follow from the 1st because some pleasantries should precede the 2nd response).

In conclusion, we are confident that the answer to our research question is that an LLM can indeed be used for comment coding. Our accuracy of 73% would be somewhat convincing if we were trying to predict 2 labels, but is even more convincing because we are in fact predicting 9 labels. We were able to take this step by building on Adhikari et al. (2019) and finetuning BERT to predict more than 2 classes, by asking the BERT model: *How* does the second sentence follow the first? Is it agreement, disagreement, neutral, critical or supportive? This code and configuration allowed us to label 9 different classes of comments with a substantial level of reliability, before any finetuning measures are even applied. Future researchers can build on these results by examining other comment coding situations. For example, it might be possible to code more categories, longer and less-focused comments, or to even have the transformer model define the classifications itself.

6. FUTURE RESEARCH

There are some things we can attempt in order to improve the accuracy of the classifier. One thing is Prompt finetuning in which we include the thought process which the LLM should use, in classifying each comment. Also including the scenario being discussed could be a fruitful approach. By using chain of thought reasoning, we may be able to increase the accuracy further. There is the possibility of looking at the actual codes used to classify. It became apparent after a quick review, that there were codes which were confused with each other, and the realization that a human would have probably confused them too. We also plan to explore ways in which the classifier could actually challenge the human ratings. That is to say, using generative AI to explain why it chose a particular code. So, it would not just be right or wrong, just more or less reasonable.

We plan to address the empty and mislabeled comments in a future study. We will look where we address the creation of an optimal set of labels which avoid the overlap with other labels and the gaps between labels. But for this study, we decided to limit the discussion to the use of BERT to assist human coders.

7. REFERENCES

Adhikari, A., Ram, A., Tang, R., & Lin, J. (2019). *DocBERT: BERT for Document Classification* (arXiv:1904.08398). arXiv. http://arxiv.org/abs/1904.08398

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. http://arxiv.org/abs/1810.04805

Faraj, S., Kudaravalli, S., HEC Paris, & Wasko, M. (2015). Leading Collaboration in Online Communities. *MIS Quarterly*, *39*(2), 393–412. https://doi.org/10.25300/MISQ/2015/39.2.06

Haines, R., Hough, J., Cao, L., & Haines, D. (2014). Anonymity in Computer-Mediated Communication: More Contrarian Ideas with Less Influence. *Group Decision and Negotiation*, *23*(4), 765.

McHugh, M. L. (2012). Interrater reliability: The kappa statistic. *Biochemia Medica*, *22*(3), 276–282.

Sarker, S., Xiao, X., & Beaulieu, T. (2013). Guest Editorial: Qualitative Studies in Information Systems: A Critical Review and Some Guiding Principles. *MIS Quarterly*, *37*(4), iii–xviii.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. ukasz, & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, *30*. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html