CONVERSATIONAL QUESTION ANSWERING USING BERT

Surya Muthiah Pillai
Computer Science
SUNY Buffalo
New York
suryamut@buffalo.edu

Abstract

Conversational Question Answering (CoQA) Challenge is a challenge aimed at evaluating the ability of the machines to perceive a passage and answer the following interconnected questions. The coqa dataset has about 127k conversation turns collected from 8k passages. CoQA has a rich variety of questions when compared to other question answering datasets like SQuAD 2.0. Text passages are collected from seven diverse domains, five for in-domain evaluation and two for out-of-domain evaluation. The main task of the project is to implement a model that answers the conversational questions based on a given passage by finding the context span of the answer to that question in the passage i.e., to produce a fine-grained output (answer) at the token level.

1 Introduction

The proposal focuses on reading comprehension question answering which aims to better understand a passage and answer a set of interconnected questions that follow. Since the question answering tasks have long range dependencies which requires them to understand context from the passages and previous questions, use of transformers would solve these dependencies and help in understanding the context in a better way. Transformer is an attention-based encoder-decoder model which solves the sequence-to-sequence tasks. Since the question answering task is a sequence-to-sequence task requiring long distant context and the transformer-based models have proven to answer the conversational questions with high precision. We adopted a bidirectional transformer model for our project. Our BERT baseline Implementation achieves an F1 score of 61.4 on the development dataset.

2 Related Work

The Question answering task is always a very challenging task as it needs a complete understanding of the language and using inference and reasoning to answer the questions. Pre-trained language models have proven to be very efficient and efficacious for improving a wide variety of NLP tasks. The idea behind pre-training language models is to create an end-to-end system that effectively understands the language and can then be used to perform specific tasks in that language. Two strategies that previously existed before to apply these pre-trained representations were Feature based approach and Fine-Tuning approach, both these approaches are unidirectional and it makes it difficult to use these architectures especially for conversational question answering as they require context to be incorporated from both directions. BERT (Bidirectional Encoder Representations from Transformers) is a Transformer-based machine learning model for pre-training NLP tasks developed by Google. This model is used to pretrain deep bidirectional representations from unlabeled text in both directions. Pre-trained BERT model can be finetuned with just one additional output layer to create state-of-the-art models for a wide range of tasks like language inference, paraphrasing and question answering tasks, we will be adopting the Bert pretrained models with linear output layer as our baseline model for answering the conversational questions. Several BERT implementations have achieved human-level performance on question answering tasks. The baseline model that is being proposed is the $\text{BERT}_{\text{BASE}}$ (uncased) model, uncased refers to case insensitivity.

3 Approach (BERT Model)

We introduce BERT on COQA and its implementation in this section. Bert Model basically uses 2 steps: pre-training and fine-tuning. For the pre-training process, BERT uses unlabeled words from the English “Wikipedia” (2500M words) and “BooksCorpus” (800M words). During finetuning, unlike training the unlabeled parameters during pre-training, we fine-tune the labelled parameters in any of the downstream tasks. Initialized with the same pre-trained parameters, each downstream task has separate fine-tuned

models. In our case, the downstream task is the Question Answering challenge on COQA.

BERT uses Transformer encoder blocks which uses Multi-Headed Self Attention mechanism that learns contextual relations between words in the given text. Unlike directional models, which read the text input from either left-to-right or right-to-left, the MLM objective is to make the representation to use both the left and the right context, which allows to pre-train a deep bidirectional Transformer. So basically, what happens in a BERT model is,

- pre-training -> unlabeled corpus of 3200M words (MLM/NSP)
- fine-tuning -> labeled corpus of words corresponding to our use-case (COQA)

Building a Question Answering System with BERT: COQA

BERT takes the input question and the passage as one single packed sequence. Calculation of input embeddings can be represented as input-embeddings = token-embeddings + segment-embeddings (+ position-embeddings). The input is defined as explained below before inputting into the model:

• Token embeddings: A [CLS] token is added at the beginning of the question and a [SEP] token is added at the end of both the question and the paragraph and are taken as input to the model.

• Segment embeddings: Sentence A and B markers added to each token. Markers in sentence A correspond to the question whereas in B they correspond to the paragraph. This enables the model to distinguish between sentences.

3.1 Pre-training BERT

Masked LM (MLM) and the Masking Procedure: Let's take an example of an unlabeled sentence, we painted the house green, and suppose the random masking procedure, masks the 4-th token "house", our masking procedure can be further described as,

• 80% probability that the word is replaced by the [MASK] token, e.g., we painted the house green → we painted the [MASK] green.

• 10% probability that the word is replaced with a random word, e.g., we painted the house green → we painted the magistrate green.

• 10% probability that the word is unchanged, e.g., we painted the house green → we painted the house green.

The reason for this is to bias the representation on the actual observed word.

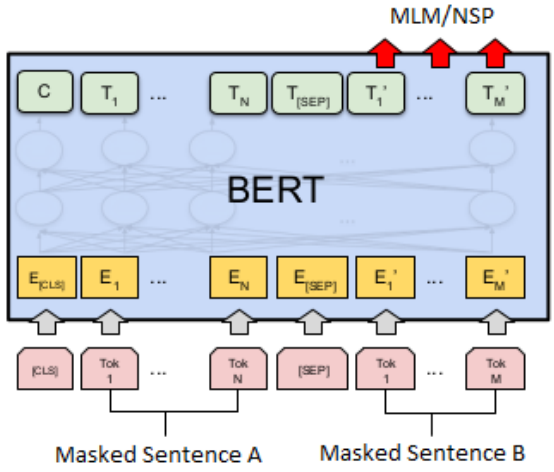

Figure 1: Pre-training on BERT

With the above pre-training done on the corpus of BooksCorpus and Wiki, the same pre-trained model can be used to now predict the labelled words on COQA (Figure 2).

Next Sentence Prediction (NSP): In the BERT training process, the model is inputted with pairs of sentences (sentence A and B) and learns to predict if the second sentence i.e., Sentence B, is the subsequent sentence or Next sentence to the sentence A in the input. The second sentence B is chosen in such a way that the actual subsequent sentence to A has a 50% probability to be chosen while the remaining 50% is a random sentence to be chosen. This is done to so that the random sentence B will be disconnected from the first sentence A to some extent.

Figure 1 is a basic representation of the how the tokenization and other above-explained scenarios happens during pre-training.

3.2 Fine-tuning BERT

We introduce two additional parameters for the fine tuning: a start vector $S \in R^H$ and end vector $E \in R^H$. Let's denote the final hidden vector from BERT for the *i-th* input token as $T_i \in R^H$. We estimate the probability of each word i being the start vector of the answer span from the packed sentence as a dot product between T_i and S followed by a SoftMax over the whole set of dot-product'ed words in the packed-sentence. The end vector is also calculated in a similar way. And the training objective is the log-likelihood

of the predicted start and end positions. If the question has no answer, we keep the start and end positions as 0. Figure 3 can help provide an easy understanding of the above explanation of BERT model on COQA.

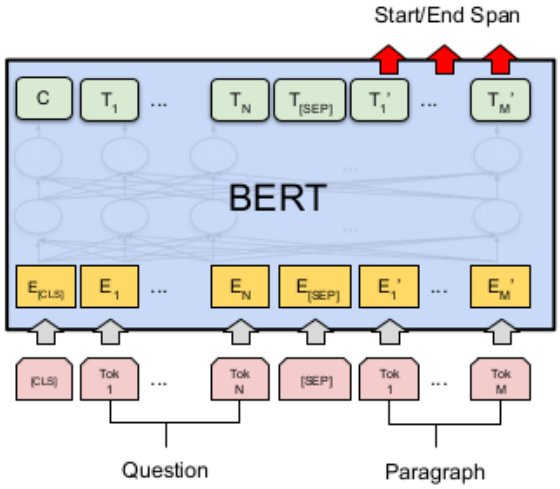

Figure 2: Fine-tuning of BERT on COQA

The Start/End Span will be discussed in the next section of Fine-tuning.

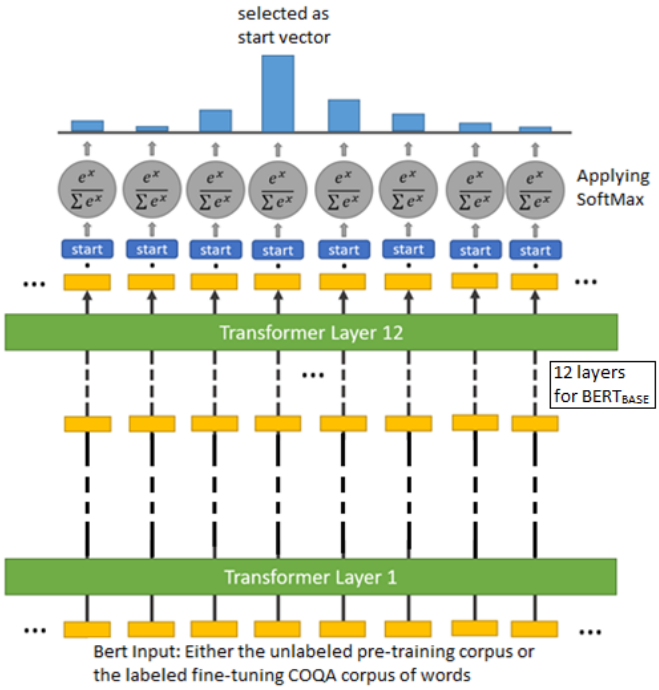

Figure 3: Transformer Architecture of Layers to find start-word and end-word

Apart from the above defined start and end logits (from this point, most values will be referred as logits as any value that takes will be represented within the range of [0,1] by applying an activation function such as SoftMax or Sigmoid), we additionally introduce yes and no logits. As our final goal involves calculating the SoftMax over all the words in order to find the start and end vectors that marks the span, now instead of taking only the start and end logits into account for finding the span, we also include the newly added yes and no logits which will correspond to Yes or No answers in the conversation along with unk logits which will refer to the unknown answers in the paragraph.

The discussion of our baseline implementation and our final model with this additional logits and concepts will be explained in detail in further sections along with the results.

3.3 Implementation of BERT Model as Baseline

Our Main idea is to use $\text{BERT}_{\text{BASE}}$ on top of the fully-connected layer we have defined using Pytorch. So basically, the objective is to use the above idea of fine-tuning BERT which will provide the baseline results along with the fully-connected layers which are going to be user-defined to test our dataset which will be explained in the Section where our final model will be precisely explained. First, we start the fine-tuning by providing the input text into the BERT model. When entering the BERT Model, the inputs are first converted into the digestible format of BERT i.e., the [CLS] and [SEP] tokens are added so the input becomes processable by our BERT model. The baseline which is implemented is $\text{BERT}_{\text{BASE}}$ uncased model which uses 12-layered transformers. So, after adding the required tokens, the process is followed by the masking procedure by adding [MASK] tokens along with two additional tokens [PAD] and [UNK] for padding and unknown words respectively. Now, this input is fed into the 12 transformer layers in the $\text{BERT}_{\text{BASE}}$ model and outputs the processed output of length 768 weights vector.

Baseline vs Improvements

Normally, this length 768 vector is the weights for the start/end token classifier which enforces the marking of the start and end of the answer text span. Since, we only test our baseline model now i.e., $\text{BERT}_{\text{BASE}}$ uncased, after testing, we achieved a maximum f1 score of 61.4% over 4 runs. So, improvements with the above-mentioned fully connected layers that would involve the additional logits and rationale tagging formulas which we are going to use

that can possibly improve the test results and is being currently tested will be explained in detail in the next Section.

4 Our Final Implementation: Adaptation of BERT on COQA

Let's look at how the model is fine-tuned and adapted from the baseline model to improve our performance which is our final architecture to the COQA problem.

Data that is used to provide for BERT consists of tokens which comprises of usually free-form text and other tokens we have discussed earlier in this paper that includes [MASK], [UNK] and [PAD] tokens along with the [CLS] and [SEP] tokens.

4.1 Introducing Yes/No/Unknown Answers

Unlike Squad Dataset which has only 0.1% Questions which has either Yes or No as their answers while COQA being vast has 11.1% of questions with answers as "Yes" and 8.7% with "No" answers contributing to a total of close to one-fifth of the total data. In order to exploit this or rather use it to our advantage over the dataset, we introduce three new answers form yes, no and unk which will be taken into account each time anything involves other tokens such as calculating logits or finding the span which will be discussed later in detail. The above logits yes and no correspond to the yes and no answers to the questions and unk logits refers to the unknown answers of the conversation.

4.2 Paragraph – History of Conversation

As discussed earlier, when the model tries to find an answer, it is supposed to depend on the previous answers as well.

Let's consider the following example from COQA's official paper in Figure 4.

Looking at Q4, the answer is going to depend on Q3A3 pair as well. Because the system needs find who 'his' is in Q4 to which the answer is present in Q3A3. So, what the implementation does is, it takes the whole Question-Answer history that has occurred before the incident Question and appends it to the total span assuming the whole span will contain the answer to the current question. So, we define that by marking the possible text span to our current i-th question in a given particular conversation is given by,

$$\mathbf{Q'_i = \{Q_1A_1, \dots, Q_{i-1}A_{i-1}, Q_i\}} \qquad \textbf{...(i)}$$

where Q_i is the i-th question (or turn) in a conversation.

Q_1: What are the candidates **running** for?
A_1: Governor
R_1: The Virginia governor's race

Q_2: **Where**?
A_2: Virginia
R_2: The Virginia governor's race

Q_3: Who is the democratic candidate?
A_3: **Terry McAuliffe**
R_3: Democrat Terry McAuliffe

Q_4: Who is **his** opponent?
A_4: **Ken Cuccinelli**
R_4 Republican Ken Cuccinelli

Q_5: What party does **he** belong to?
A_5: Republican
R_5: Republican Ken Cuccinelli

Q_6: Which of **them** is winning?
A_6: Terry McAuliffe
R_6: Democrat Terry McAuliffe, the longtime political fixer and moneyman, hasn't trailed in a poll since May

Figure 4: Example conversation from COQA dataset

4.3 Rationale labelling (denoted by R in Figure 4)

After appending the history of QA pairs, next step is rationale tagging. Tokens that belong to the passage i.e., tokens that are present in equation (i) are going to be labelled as 0 or 1. Tokens that are tagged or marked as 1 correspond to the rationale meaning the rationale is going to contain the tokens which are tagged as 1. Tokens that are labelled 0 are not going to be included in the rationale. We tag the token with the probability factor given as follows,

$$\mathbf{p_t = sigmoid\ (\ w_1\ Relu(\ W_2 h_t\ )\ )} \qquad \textbf{...(ii)}$$

where w_1 (lowercase-w) corresponds to the weight vector of single dimension d and W_2 (uppercase-W) corresponds to the 2-dimensional weight vector dxd. h_t is the Bert output vector for token t. p_t is the rationale probability for the token t. The value that the sigmoid function takes in order to calculate the rational logits, is the fully-connected (sometimes will be referred as FC layer) layer defined for this purpose. If the value is above 0.5, we mark it as 1 meaning that it belongs to the rationale or 0 when probability is <0.5.

4.4 Finding the logits

The next step is to calculate the hidden output which is the product of the rationale probability and the Bert output given by,

$$\mathbf{h'_t = p_t \times h_t} \qquad \textbf{…(iii)}$$

A fully-connected layer will be again applied the same as that of equation (ii) on h'_t which will be used to achieve one of the core objectives of our mission, that is finding the start and end logits. So explained in short, we calculate it using the above fully-connected layer which takes the Bert output h_t in equation (ii), consecutively finding h'_t in eq. (iii) and finally applying a FC layer on h'_t in order to find the start and end logits (l^s and l^e respectively).

Note: Unknown answers do not have a rationale. The text span with the highest f1 score is selected as the gold answer to start the training. We train the model in such a way that the probability that this span has the answer increases by defining a training objective which will be discussed further in this document.

Since, we have introduced yes and no tokens, yes, no and unk logits remain to be calculated. For this purpose, we define another pooled output h* to which again the fully-connected layer is applied to obtain the remaining 3 logits – l^y (yes), l^n (no) and l^u (unknown). So, how do we calculate this pooled output for these yes, no and unk logits?

$$\mathbf{a_t = softmax^T_{t=1}\,(w_{a2} Relu(W_{a1} h'_t))} \qquad \textbf{…(iv)}$$

$$\mathbf{h^* = \sum a_t \times h_t} \qquad \textbf{…(v)}$$

Similar to eq. (ii), we apply SoftMax to the output of the fully connected layer to find a_t representing an attention layer which takes h'_t as input instead of h_t. So, point to be noted is that we are applying the FC layer to h't which has already been applied with the FC layer once before (corresponding to eq. (ii)) and then after subject to two times of the layers we apply the SoftMax in eq. (iv). After then only we find the value of h* as in eq. (v) which again undergoes the fully-connected layer once again for the final third time to calculate the yes, no and unk logits.

Now, we have 5 logits – l^s, l^e, l^y, l^n, l^u that corresponds to start/end/yes/no/unk logits respectively (the value that p_t calculates is the rational logits which will not be used anymore).

4.5 Training Objective (finding the perfect span)

With the above 5 values, now we near the end of the main objective, i.e., finding the correct start and end vectors. Without the other logits, naturally these vectors would be calculated by the SoftMax of the tokens alone as stated in the baseline model. But since we have other logits to take into consideration in our improved model, the probability that a word or token becomes the start/end vector of the span which probably is going to contain the answer is given by,

$$\mathbf{p^{start} = softmax([l^s , l^y, l^n, l^u])}$$
$$\mathbf{p^{end} = softmax([l^e , l^y, l^n, l^u])}$$

Since there are two values involved for the training objective p^{start} and p^{end}, we define the cross-entropy loss as follows by adding the two values and dividing by 2.

$$\mathbf{Loss = -1/2N * (\sum_1^N (\log p^{start} + \log p^{end}))}$$

The training objective is to minimize the above loss function which will train the learnable parameters in such a way that each time the loss is reduced, we have a better start and end vector of the text span which will determine the answer to the questions more precisely.

5 Experimental Details and Result Analysis

5.1 Data

Our model was trained and evaluated on the CoQA's training and development dataset. There are about 127k question answer pairs, our model was evaluated against 7983 turns from the development set. All the turns in the development data are in domain, there are no out-domain questions in the development data.

5.2 Evaluation Metrics

Metrics used to evaluate the performance of our model are, Exact Match (EM) score and F1 Score. Exact Match score is a strict metric that measures the percentage of predictions that matches the answers exactly. F1 is a less strict metric that measures the average overlap between the predicted output and the truth answer. It is the harmonious mean of the prediction and recall based on the total coverage of the answer over the actual ground truth. F1 score is the primary evaluation metric in the project.

Runs	Children stories	Literature	Mid high school	News	Wikipédia	In domain	Overall
#Baseline1	56.2	53.4	54..5	58.4	60.0	56.7	56.7
#Baseline2	60.9	58.1	59.1	63.1	65.7	61.4	61.4
#Final1	73.4	69.1	70.1	74.7	76.3	72.5	72.5
#Final2	74.5	71.2	72.2	76.8	78.4	74.6	74.6

Table 1: F1 score on BERT Baseline vs Final model

Runs	Children stories	Literature	Mid high school	News	Wikipédia	In domain	Overall
#Baseline1	46.1	43.1	43.9	47.5	51.6	46.4	46.4
#Baseline2	50.8	47.8	48.6	52.2	56.3	51.1	51.1
#Final1	63.0	60..1	60.0	66.0	66.7	62.7	62.7
#Final2	65.1	62.1	62.1	66.1	68.8	64.8	64.8

Table 2: EM score on BERT Baseline vs Final model

Runs	Learning rate	Weight Decay	Epochs
#1	$1e^{-5}$	0	1
#2	$1e^{-5}$	0.01	2

Table 3 : Hyperparameters used

5.3 Results Comparison (Baseline vs Final Architecture)

The $BERT_{BASE}$ (uncased) baseline model was run multiple times on the development data over different hyper parameters. The out-of-domain results are NA as the development data has no out-of-domain questions. After the baseline testing, our final model was tested for accuracy over a few runs. It was observed there was significant increase in accuracy as we introduced the yes and no logits which influenced the evaluation to a higher extent. Our baseline model produced an accuracy of 61.4% while the final model predicted the correct answers with an accuracy of 74.6%. This was the case as COQA unlike SQUAD was overwhelmed with the Yes or No answers that affected the yes/no logits to a greater extent. The results comparison is tabulated between the final model and baseline for better view of the numbers in evaluation in Tables 1, 2 and 3. So, comparing the results, we can see that there is a noteworthy increase of approximately 14-16% in the accuracy of our finalized model from the baseline model.

5.4 Model Performance Analysis

The BERT model was experimented on various hyperparameters, the model gave best results for the following hyperparameters: Learning rate: 1e-5, training batch size allocated per GPU: 2, Weight decay: 0.01, Epochs: 2, Optimizer: AdamW. Tuning the hyper parameters improved the model performance substantially. For the learning rate parameter, we started with 4e-4, which gave a low F1 score initially. After experimenting the model with several learning rates, the best performance which gave an F1 score of 74.6 was learning-rate=1e-5. A weight decay regularization value of 0.01 was enforced to prevent overfitting which provided noticeable growth in the overall F1 score. We chose AdamW as the model optimizer over the Adam optimizer, because it generalizes the model better than Adam optimizer yielding a better training loss. The Number of training epochs is set to 1, when the number of training epochs was set to 2, model training consumed about more than 9 hours to run and resulted in an increase in performance, but it also resulted in CUDA out-of-memory errors during the course of execution which was resolved by reducing the epochs back to 1. This was run on NVIDIA Geforce RTX 2060. If the end user utilizes a more powerful GPU or uses TPU, the epochs and batch size can be increased with respect to their convenience. This might yield even more better results compared to our model performance that yielded a f1 score of approximately 75 at best in our case.

6 Error Analysis and Discussion

There are a few examples where our model wouldn't perform as expected. But since these types correspond to very low percentage in the bigger picture, trying to tune the model to address these types of issues may affect the model performance in a negative manner. So, its best if it solved in a more definitive way other than the model we have defined. These examples that affect the above scenario mostly include grammatical problems and counting problems. For example, if the model is asked a question on Q: "how many types of vehicle are useable". The rationale will include something like R: "car, truck, bike and bus type of vehicles can be used". So, our answer will be predicted as A: "car, truck, bike, bus" rather than the ground truth which is four. The answer displays the types rather than counting the types. If looked at how the grammatical issues occur, for example, let the question be Q: "What happened to the titanic when band was playing violin", so our rationale would be something like, R: "As the band continues to play the violin, the ship is flooding with the waters of the ocean.". The predicted answer to the question would be A: "the ship is flooding with waters" in which case our answer is supposed to be in past tense meaning that the ground truth i.e., "the ship was being flooded." is different in tense para-phrasing from the predicted answer. Since, the above kind of answers sometimes were correctly present in the rationale and predicted correctly, trying to solve for these types of inconsistencies might adversely affect our model.

7 Conclusion

In this paper, we discussed how we have implemented the $BERT_{BASE}$ uncased model to train our dataset for our baseline model implementation and improved it by introducing additional parameters like yes, no and unk logits that can heavily influence our model especially in the case of COQA. Comparing to other models such as DrQA, PGNet, the BERT model implemented has produced comparable results in competition with those models with even sometimes producing vaguely close to state-of-the-art model results after tuning it in the appropriate manner. The plan execution on improving the results has been successfully tested towards the goal of achieving better accuracies on our model. The above mechanisms and

explanation in this paper should help understand how our BERT baseline works and our final model has been implemented and also the results produced by comparing the baseline model and our model by tuning the hyper-parameters which is believed to assist in how results can be improved on the notion to improve the efficiency as well.

References

[1] Ying Ju, Fubang Zhao,Xuefeng Yang, Yunfeng Liu Technical report on Conversational Question Answering. https://arxiv.org/pdf/1909.10772.pdf

[2] Siva Reddy, Danqi Chen, Christopher D. Manning *CoQA: A Conversational Question Answering Challenge* .

[3] D. A. Chen, J. W. Fisch, and A. Bordes. 2017. *Reading Wikipedia to Answer Open-Domain Questions*.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *Bert: Pre-training of deep bidirectional transformers for language understanding*.

[5] J. Chaumond, P. Cistac, T. Wolf, and V. Sanh (Hugging Face). pytorch-pretrained-bert source code. https://github.com/huggingface/pytorch-pretrained-BERT.

[6] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *Squad: 100,000+ questions for machine comprehension of text.*

[7] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. *Deep contextualized word representations.*

[8] https://huggingface.co/transformers/