±

# QUESTION ANSWERING MODEL

**A Project Report**

*Submitted by*

**ASHUTOSH GANESH JADHAV**
**JAYESH RAJU AKOT**
**SUCHIT PURSHOTTAM AWATE**

***in partial fulfillment for the Project of***
**B. TECH**
**IN**
**COMPUTER SCIENCE AND ENGINEERING**
**At**

**SHRI GURU GOBIND SINGHJI**
**INSTITUTE OF ENGINEERING AND TECHNOLOGY,**
**VISHNUPURI, NANDED**
**(MAHARASHTRA STATE)**
**PIN 431 606 INDIA**

**MAY 23**

# “Question Answering Model”

**A PROJECT REPORT**

*Submitted by*

**ASHUTOSH GANESH JADHAV**
**JAYESH RAJU AKOT**
**SUCHIT PURSHOTTAM AWATE**

***in partial fulfillment for the Project***
***of***

**Final Year B. Tech**

**in**
**Computer Science and Engineering**

**at**

**SHRI GURU GOBIND SINGHJI**
**INSTITUTE OF ENGINEERING AND TECHNOLOGY,**
**VISHNUPURI, NANDED**
**(MAHARASHTRA STATE)**
**PIN 431 606 INDIA**

**MAY 2023**

**Shri Guru Gobind Singhji Institute of Engineering & Technology,**
**Vishnupuri, Nanded (M.S.), India – 431606.**

**Department of Computer Science and Engineering**

# *Certificate*

This is to certify that the project work entitled “Question Answering Model” is submitted by students **Ashutosh Jadhav**, **Jayesh Akot** and **Suchit Awate** to Shri Guru Gobind Singhji Institute of Engineering & Technology, Nanded for the partial fulfillment of the award for the degree of Bachelor of Technology in Computer Science and Engineering. This project is a record of bonafide work carried out by them under my guidance. The content presented in this report has not been submitted to any other University or Institute for the purpose of obtaining any other degree or diploma.

| | |
|---|---|
| Guide | Head of Department |
| **U.V. Kulkarni** | **J.M.Waghmare** |

# ABSTRACT

Extractive Question Answering is a part of Natural Language Processing and Information Retrieval. A context is provided in Extractive Question Answering so that the model can refer to it and make predictions about where the answer is inside the passage. The SQuAD dataset, which is entirely task-based, is an example of a question-and-answer dataset. Data labelling for question answering tasks (QA) is a costly procedure that requires oracles to read lengthy excerpts of texts and reason to extract an answer for a given question from within the text. QA is a task in natural language processing (NLP), where a majority of recent advancements have come from leveraging the vast corpora of unlabeled and unstructured text available online. This work aims to extend this trend in the efficient use of unlabeled text data to the problem of selecting which subset of samples to label in order to maximize performance. This practice of selective labelling is called active learning (AL).

# ACKNOWLEDGEMENT

At the very beginning of preparing this report, I would like to convey my gratitude to my parents for their blessing in completing this report. I am obliged to all those people who helped me to organize this report and for their suggestions, instructions and support with proper guidelines. Firstly, I would like to thank my honorable project supervisor and Project Coordinator **U. V. Kulkarni** , Professor, SGGSIE&T for their continuous guidance in completing of this report. His valuable advice has helped me a lot in preparing this report properly. I am really thankful to him for the support that he has provided.

## TABLE OF CONTENTS

## Table Of Figures

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **EQA** | Extractive question answering |
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **MLM** | Masked Language modelling |
| **NSP** | Next Sentence Prediction |
| **AL** | Active Learning |
| **SQuAD** | Stanford Question Answering Dataset |
| **NLP** | Natural Language Processing |
| **RNN** | Recurrent Neural Networks |
| **GPT** | Generative Pre-trained Transformers |
| **EM** | Exact Match |
| **JSON** | JavaScript Object Notation |
| **RoBERTa** | Robustly Optimized BERT pretraining approach |
| **TPU** | Tensor Processing Units |

# 1. Introduction:

## 1.1 Introduction to project:

Extractive question answering (EQA) is the task of finding an answer span to a question from a context paragraph. Most of the deep learning models for this task perform well when annotated data is present. Scaling such models to new domains often requires creation of new datasets. However, collecting labels for these corpora is expensive and time consuming which may involve multiple steps such as article curation, question and answer sourcing. Alleviating the annotation efforts for any of these steps is not only of research but also of practical interest. In this work, we address the problem of extracting answer spans to a question from unannotated context paragraph. Some works have already been proposed to solve EQA in both semi-supervised and unsupervised setting. Unsupervised methods focus on creating a synthetic corpus and further train a supervised model on the synthetic corpus. In semi-supervised methods the focus is on different pre-training tasks that improve the initialization of the EQA models. Our work can be categorized as the latter with one key difference: to further perform question answering without annotations on answer spans. To validate our approach, we use the pre-trained BERT model using SQuAD.

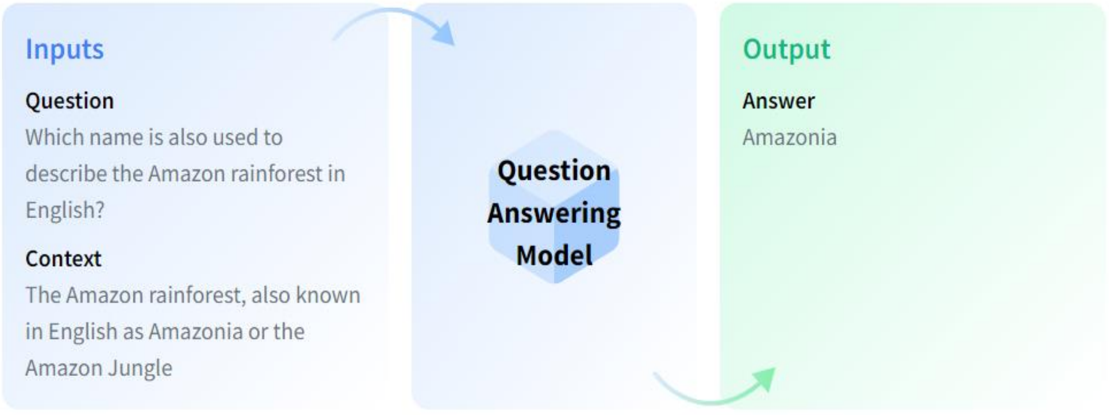
Inputs
Question
Which name is also used to describe the Amazon rainforest in English?
Context
The Amazon rainforest, also known in English as Amazonia or the Amazon Jungle
Question Answering Model
Output
Answer
Amazonia

Fig1.1 QA Model overview

## 1.2 Category :

Natural language processing (NLP) is the field that deals with computer understanding of text and other forms of human language. Over the last years, the field has been experiencing a significant surge of interest due to advances brought by the widespread use of transformer based models . Said models take advantage of attention mechanisms and vast quantities of readily available text corpora to learn general characteristics of language in a self-supervised manner. In NLP, labelling is often a complex and costly endeavor performed by humans. In the case of question-answering, an oracle, typically a human annotator, must first read a text excerpt and then reason about where in the context the answer to the question appears. As a result, effective techniques minimizing sampling costs can make cost-prohibitive use cases feasible, therefore, widening the situations where these models can be effective. More efficient labelling in question answering (QA) problems could substantially increase the viability of use cases such as automated support processes or automatic text summarization due to the potential decrease of the required amount of labelled data. Several techniques can be leveraged to reduce the number of labels needed. Self-supervised methods, data augmentation and active learning (AL) strategies can be used on their own or in conjunction with each other to this end. As a company that solves problems using machine learning, Peltarion's services are often used in various automation projects where no previous solution or infrastructure is available. New use cases often require extensive data collection or labelling to implement effectively. Therefore, a more time and cost-effective labelling process would reduce project costs for customers, lower the data readiness requirements for projects and thus allow for more people and companies to make use of their products. Recent NLP research has already observed a trend of increased performance with a fixed set of labelled task-specific samples thanks to refined self-supervision training stages and more complex models. Performing self-supervised learning allows models to learn without explicit sample labels. Language models such as "Bidirectional Encoder Representations from Transformers" (BERT) learn to model the probability distribution of a sub-word unit given its surrounding context. Since its first introduction, BERT has become the field's defector benchmarking model for all NLP tasks due to its simplicity and close to state of the art performance. Many of these generally pre-trained models are capable of performing well in a variety of tasks without further task-specific modelling (zero-shot learning). However, additional problem-specific training has been shown to outperform both models only trained on specific tasks and the zero-shot capabilities of generally pre-trained models. This two stage training improves generalization without increasing the number of labelled examples while increasing the observed performance.

## 1.3 Objective:

Specifically, our method employs a conditional auto-encoding scheme that reconstructs question given a passage while assuming a latent distribution over the answer phrases. The encoder of our model is a Question Answering model that jointly encodes the context and the question to estimate the probability distribution over possible answer spans. This is further given as input along with passage to the decoder which is a Question Generation model. We use a shared architecture for both the encoder and the decoder. Therefore, our model can be viewed as a self-supervised machine comprehension model that learns from itself. We list our contributions as follows:

- We propose a novel method to perform unsupervised answer span extraction given a corpus of questions and associated paragraphs.
- We obtain an accuracy of 90% on unsupervised answer sentence selection.
- We obtain strong results (83.0 EM, 88.25 F1 on SQuAD dev set) for EQA when there is no annotation on the answer spans.

## 2.Problem Statement

The objective of this project is to build an extractive QA model capable of accurately answering questions based on a given passage of text. The model should be able to understand the context, identify relevant information, and extract the correct answer from the passage.

# 3. Data:

## 3.1 Extractive Question Answering:

Extractive Question Answering (QA) requires models to demonstrate machine reading comprehension, as the answers to some questions are not always answered verbatim in the text. Models learning this task often use position-based pointers to identify two unrestricted positions within a question prompt. The text between said two pointers is called a span. This span of text is extracted and compared with a set of golden labels for evaluation. The following example illustrates this process using an example from a QA benchmarking dataset called SQuAD. Context: The Normans were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. Question: From which countries did the Norse originate? Starting position: 41 Ending position: 45 Answer: Denmark, Iceland and Norway It is important to note why QA models evaluate text excerpts and don't do so on pointer labels. Different models consider the smallest unit of text differently. Since there is no standard definition of a text unit, models would interpret index-based labels differently. For that same reason, the transformation of text sequences into sets of text units must remain consistent. The same pre-processing is applied to both the training data and the labels. This shared definition of a text unit allows the measurements of overlap and intersections between sets of text units in model predictions and ground truths.

## 3.2 SQuAD 1.0:

SQuAD 1.0 is a question answering dataset comprised of over 100 thousand text-question pairs obtained from Wikipedia and labelled by humans. Each question pair has four possible answers, each given by a different oracle. Since all four answers are considered equally valid, model predictions are compared against all of them and only the resulting highest score is considered. Unlike more modern QA datasets, this iteration of SQuAD does not contain any unanswerable questions. The dataset comes fully labelled and pre-split into training, validation and test sets, the distribution of which is as follows:

- Train: 87,599
- Dev: 10,570
- Test: 9,533

Of which this work only uses train and development sets.

### 3.3 The SQuAD format

Each data point in a dataset following the standard SQuAD format has the following features:

1. id: a string feature.
2. title: a string feature.
3. context: a string feature.
4. question: a string feature.
5. answers: a dictionary feature containing one or several entries:
    a) text: a string feature.
    b) answer start: an int32 feature.

For this work, the only relevant features will be context, question and the answer text field. Note that data in natural language processing is not tabular, which elicits the need for nontrivial pre-processing steps before being used in a model. Since this pre-processing is highly dependent on the model being used.

# 4. Theory:

## 4.1 Transformer-based Neural Networks

Multi-layer Neural Networks
Multi-layer perceptions are a fairly compact and fast-to-evaluate models. At its core, this architecture is essentially comprised of multiple linear regression models, or layers, with continuous non-linearity functions between them. A given neuron j in the first layer can be defined as the linear combination of hidden parameters θ and an input vector z of length $d_z$ as follows:

$$n_{1,j} = h_1(\sum_{i=1}^{d_z} \theta_{1,j,i} z_i)$$

These are then transformed using a differentiable non-linear activation function $h_1$ to be able to explore non-linear relationships between input data and targets. Building on this concept, multilayer perceptrons form a linear combination of the previous layer's non-linear outputs. The model contains several layers l, each composed of $d_l$ neurons. The following is the output for a neuron m in layer number l of an MLP where the previous layer is of size $d_{l-1}$ and the layer's activation function is $h_l$ .

$$n_{l,m} = h_l(\sum_{j=1}^{d_{l-1}} \theta_{l,m,j} * n_{l-1,j})$$

This process may be nested repeatedly to create a network of any arbitrary depth and capture more complex relationships between inputs and outputs. Much like other parametric models, the optimal parameters ˆθ can be found by minimizing the average value of a loss function dependent on them. The non-linearities present between layers causes the loss function to be non-convex, causing the appearance of local minima and making parameter optimization challenging, MLP architectures that are several layers deep are commonly referred to as deep neural networks (DNNs). This umbrella term encompasses a varied set of architectures, some of which include recurrent connections between layers which are not covered in this work.

## 4.2 BERT model

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art natural language processing (NLP) model introduced by Jacob Devlin et al. in 2019. It revolutionized the field by significantly advancing the performance of various NLP tasks, such as question answering, text classification, and language translation. BERT is a transformers model pretrained on a large corpus of English data in a self-supervised fashion. This means it was pretrained on the raw texts only, with no humans labelling them in any way (which is why it can use lots of publicly available data) with an automatic process to generate inputs and labels from those texts. More precisely, it was pretrained with two objectives:

- Masked language modeling (MLM): taking a sentence, the model randomly masks 15% of the words in the input then run the entire masked sentence through the model and has to predict the masked words. This is different from traditional recurrent neural networks (RNNs) that usually see the words one after the other, or from autoregressive models like GPT which internally mask the future tokens. It allows the model to learn a bidirectional representation of the sentence.
- Next sentence prediction (NSP): the models concatenates two masked sentences as inputs during pretraining. Sometimes they correspond to sentences that were next to each other in the original text, sometimes not. The model then has to predict if the two sentences were following each other or not.

This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks: if you have a dataset of labeled sentences for instance, you can train a standard classifier using the features produced by the BERT model as inputs.

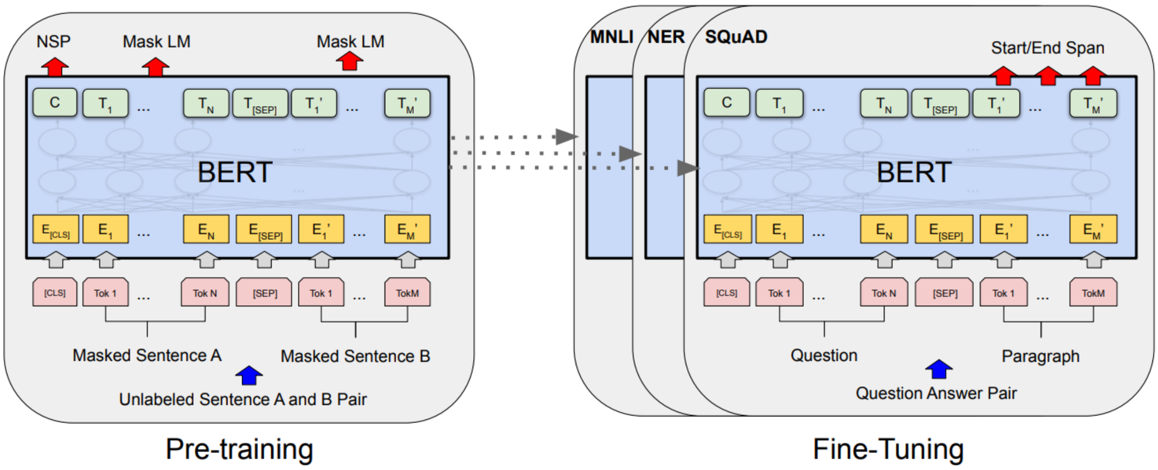

Figure 4.2: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

BERT is a pre-trained language model that utilizes the Transformer architecture, which was originally introduced by Vaswani et al. in 2017. The Transformer architecture employs self-attention mechanisms to capture relationships between words in a sentence or sequence of tokens. This allows the model to effectively handle long-range dependencies and capture contextual information.

Here are the key components and features of the BERT model:

- Tokenization: BERT uses a WordPiece tokenizer that splits words into subwords or tokens. Each token is assigned an index, and the input text is converted into a sequence of tokens.
- Pre-training: BERT is pre-trained on a large corpus of text data using unsupervised learning. It is trained on two main tasks: masked language modeling (MLM) and next sentence prediction (NSP). MLM involves randomly masking some tokens in the input and training the model to predict the masked tokens. NSP involves training the model to determine whether two sentences are consecutive in the original text or not.

- Bi-directional Context: BERT is designed to capture bi-directional context by employing a bidirectional Transformer. This means that during pre-training and fine-tuning, BERT considers both the left and right contexts of a token when making predictions.

- Transformer Layers: BERT consists of multiple layers of Transformers. Each layer has a multi-head self-attention mechanism and a feed-forward neural network. The self-attention mechanism enables the model to weigh the importance of different tokens in the sequence based on their relevance to each other.

- Fine-tuning: After pre-training, BERT can be fine-tuned on specific downstream tasks by adding task-specific layers on top of the pre-trained model. During fine-tuning, the model is trained on labeled data for the target task, such as question answering or sentiment analysis.

- Contextual Word Embeddings: BERT generates contextualized word embeddings, where the representation of each word is influenced by its context in the sentence. This allows the model to capture the meaning and relationships between words more accurately.

BERT has achieved remarkable performance on a wide range of NLP benchmarks and has become a popular choice for various NLP applications. It has been widely adopted and serves as the basis for many subsequent models, such as RoBERTa, ALBERT, and ELECTRA, which aim to improve upon BERT's limitations and enhance its capabilities.

By leveraging the power of pre-training and fine-tuning, BERT enables deep understanding of natural language and has significantly advanced the state-of-the-art in many NLP tasks, making it one of the most influential models in the field.

### 4.2.1 Training Data

The BERT model was pretrained on BookCorpus, a dataset consisting of 11,038 unpublished books and English Wikipedia (excluding lists, tables and headers).

### 4.2.2 Training Procedure

**Preprocessing**

The texts are tokenized using WordPiece and a vocabulary size of 30,000. The inputs of the model are then of the form:

```
[CLS] Sentence A [SEP] Sentence B [SEP]
```

With probability 0.5, sentence A and sentence B correspond to two consecutive sentences in the original corpus and in the other cases, it's another random sentence in the corpus. Note that what is considered a sentence here is a consecutive span of text usually longer than a single sentence. The only constrain is that the result with the two "sentences" has a combined length of less than 512 tokens.

The details of the masking procedure for each sentence are the following:

- 15% of the tokens are masked.
- In 80% of the cases, the masked tokens are replaced by [MASK].
- In 10% of the cases, the masked tokens are replaced by a random token (different) from the one they replace.
- In the 10% remaining cases, the masked tokens are left as is.

### Pretraining

The model was trained on 4 cloud TPUs in Pod configuration (16 TPU chips total) for one million steps with a batch size of 256. The sequence length was limited to 128 tokens for 90% of the steps and 512 for the remaining 10%. The optimizer used is Adam with a learning rate of 1e-4, β1=0.9 and β2=0.999, a weight decay of 0.01, learning rate warmup for 10,000 steps and linear decay of the learning rate after.

## 4.3 Transformer-based encoders blocks

Transformers use several parallel attention calculations simultaneously to form an attention layer. Each of these parallel calculations is commonly referred to as a head. For each head n, each input matrix has an associated set of parameters $\theta^Q{}_n$ , $\theta^K{}_n$, $\theta^M{}_n$ learnt during the training stage.

$$head_n = Attention(Q\theta_n^Q, K\theta_n^K, M\theta_n^M)$$

Each head's output is then concatenated together, and a final linear transformation is per- formed by multiplying the combined output by an additional set of trainable parameters $\theta^O$.For any given layer *l*, multi-head attention is calculated as follows:

$$MultiHead(Q, K, M)_l = Concat(head_1, ..., head_l)\theta^O$$

All transformations, as mentioned above, are positionally independent, meaning that no consideration is given by the self-attention mechanism to the order of the inputs. Since writ- ten language is susceptible to order and relative positioning, an additional positional embed- ding is concatenated to every token in the input. In Vaswani et al. , the position embedding is given by sine and cosine functions dependent on the token position in the input. These functions are simple to learn and, when combined, provide unique and continuous representations for all positions up to a fixed input length.

Each encoder block in a transformer-based neural network includes a layer formed with multi-head attention followed a fully connected layer. An illustration of a possible encoder block configuration can be seen in figure 4.3.1.

### 4.3.1 Tokenization and representation of text for Machine Understanding

Pre-processing of text data for use in machine learning models is a multi-step process highly dependent on the model being used. This section will first describe the overarching steps common to all techniques and then explain how BERT implements them.

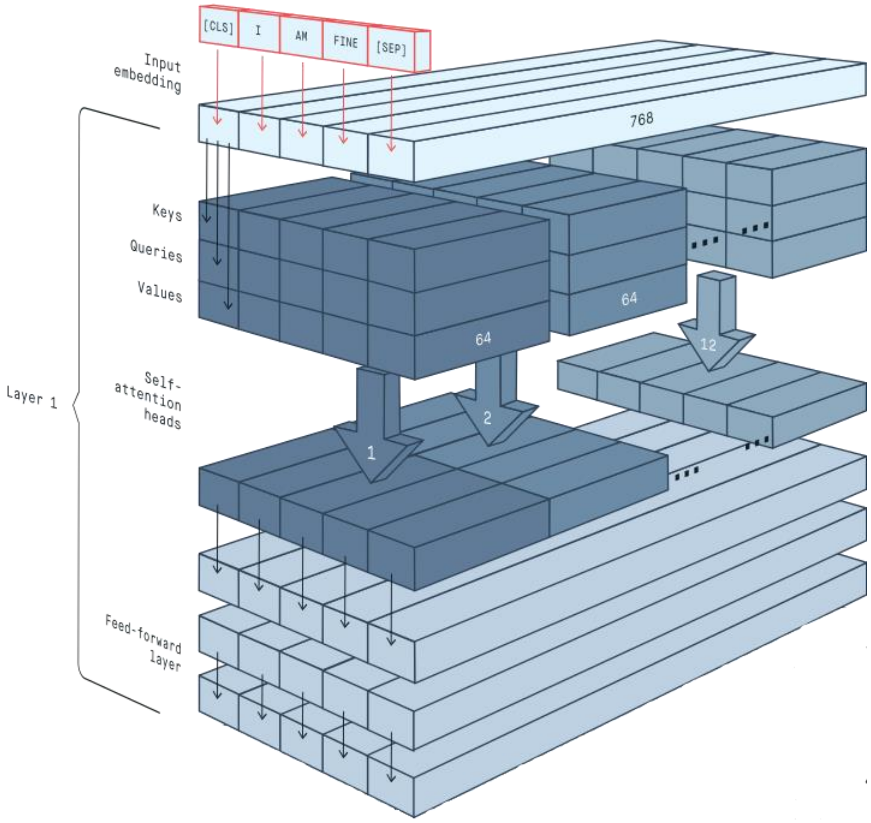

Figure 4.3.1: Illustration of transformer-based encoder block with $d_W$ = 768, $d_k$ = 64 and12 self-attention heads.

The first step in processing text data is determining what a unit of text $w$ (token) is. There are many different levels of token granularity. Smaller units of text allow for greater flexibility in the kind of text data that can be processed at the cost of poorer representation density. Figure 4.3.1 illustrates how different tokenisation levels affect the number of units needed to represent the same sequence of text $W$. All tokens have finite discrete values they can take as defined by a vocabulary $V$. This vocabulary is the list of word units the model can interpret. The finite nature of $V$ makes the selection of values critical as these methods cannot represent word units outside its vocabulary. Figure 4.3.1 shows how the vocabulary changes depend-ing on the tokenisation granularity. When used in a model, each possible state $v$ P $V$ has aset of learnt parameters $\theta_v$ associated with it. The associated parameters learnt during train- ing are of an arbitrary size $d_v$ which can be set by a hyper-parameter and become reusable dense vectorial representations for each token in the vocabulary. Each token in $W$ is replaced by its corresponding $\theta_v$. The resulting representation of the input is a matrix whose size is dependent on the sequence length $d_W$ and $d_v$.

BERT tokenises sequences of text using a technique called WordPiece.This algorithm analyses a target text corpus to determine candidates should be included in the vocabulary.It starts by adding every character type in the corpus to $V$. With a large enough corpus, this first stage produces space inefficient tokenisations that are versatile enough to represent every word in the target languages. It then iteratively creates more space efficient tokens by merging two smaller ones together. A language model is trained to estimate the likelihood of the training data given the vocabulary. The combination of tokens that increase the training data likelihood the most is added to $V$. Additions to the vocabulary are made iteratively until$V$ is the desired size $d_V$.

Sample: “The fox jumped me”

Character level:

Vocabulary: [“T”,”h”,”e”,” ”,”f”,”o”,”x”,”j”,”u”,”m”,”p”,”d”]
tokenisation: [“T”,”h”,”e”,” “,”f”,”o”,”x”,”
“,”j”,”u”,”m”,”p”,”e”,”d”, “ “, “m”,“e”]
Indexing: [1,2,3,4,5,6,7,4,8,9,10,11,3,12,4,10,3]

New sample: “The pot”
Tokenisation: [“T”,”h”,”e”,”
“,”p”,”o”,”t”] Indexing:
[1,2,3,4,11,6,1]

Word level:

Vocabulary: [“The”, “fox”, “jumped”, “me”]
Tokenisation: [“The”, “fox”, “jumped”,
“me”] Indexing: [1,2,3,4]

New sample: “The pot”
Tokenisation: [“The”, “<Unknown>” ]
Indexing: [1, -1]

Above is example of tokenisation at different levels.

BERT stores and learns dense representations of tokens by storing them in an embedding matrix. Text sequences are broken down into a series of integer indices corresponding to token positions from the built vocabulary. Each of these indices is then used to locate the corresponding dense representation within the embedding matrix.

## 4.4 Active Learning

Active learning (AL) aims to learn the true mapping between our data and our labels using the least amount of training data points. This section will cover uncertainty-based sampling techniques and how they can be used in neural networks.

Active learning is a machine learning approach that involves iteratively selecting and labeling a subset of unlabeled data to train a model. Unlike traditional supervised learning, where all training data is labeled in advance, active learning aims to minimize the labeling effort by actively choosing the most informative instances for annotation.
The key idea behind active learning is to identify instances that are challenging or uncertain to the model, as these instances are likely to provide the most valuable information for further training. By actively selecting and annotating these instances, the model can learn more effectively with fewer labeled examples compared to traditional supervised learning approaches.

Active learning can be particularly beneficial in scenarios where labeling large amounts of data is expensive, time-consuming, or resource-intensive. It allows machine learning models to achieve high performance even with limited labeled data, making it a valuable approach in various domains such as natural language processing, computer vision, and healthcare.

Overall, active learning offers a strategy to intelligently select and label the most informative instances, facilitating efficient model training and improving performance while minimizing the annotation effort.

# 5. Methodology

## 5.1 Preparing the data

The dataset that is used the most as an academic benchmark for extractive question answering is SQuAD, so that's the one we'll use here. There is also a harder SQuAD v2 benchmark, which includes questions that don't have an answer.

### 5.1.1 The SQuAD dataset

The SQuAD (Stanford Question Answering Dataset) is a popular dataset used for training and evaluating extractive question answering models. It was created by researchers at Stanford University and consists of real questions posed by human annotators on a set of Wikipedia articles. Here's an explanation of the SQuAD dataset:

- Dataset Format: The SQuAD dataset is provided in JSON format. Each entry in the dataset represents a specific question-answer pair associated with a particular passage from a Wikipedia article.
- Passage: Each entry contains a passage of text taken from a Wikipedia article. The passage provides the context in which the question is asked. It can range from a few sentences to a few paragraphs.
- Question: Each entry includes a question that was created by a human annotator. The question is formulated based on the information present in the corresponding passage.
- Answer: The SQuAD dataset is unique because it provides answer annotations at the level of individual tokens within the passage. For each question, there is a corresponding answer span in the passage that contains the correct answer. The answer span is defined by its start and end token positions within the passage. These token positions allow the model to locate and extract the correct answer.

### 5.1.2 Preprocessing the Data

When using the SQuAD (Stanford Question Answering Dataset) for training an extractive question answering (QA) model, the training data needs to undergo preprocessing to ensure its quality and compatibility with the model.

Preprocessing of the data include:

1. Tokenization: Tokenization is the process of splitting the text into individual tokens or words. In the case of the SQuAD dataset, both the passages and questions need to be tokenized. This step is crucial for further processing and analysis.

2. We can pass to our tokenizer the question and the context together, and it will properly insert the special tokens to form a sentence like this:

[CLS] question [SEP] context [SEP]

3. The labels will then be the index of the tokens starting and ending the answer, and the model will be tasked to predicted one start and end logit per token in the input, with the theoretical labels being as follow:

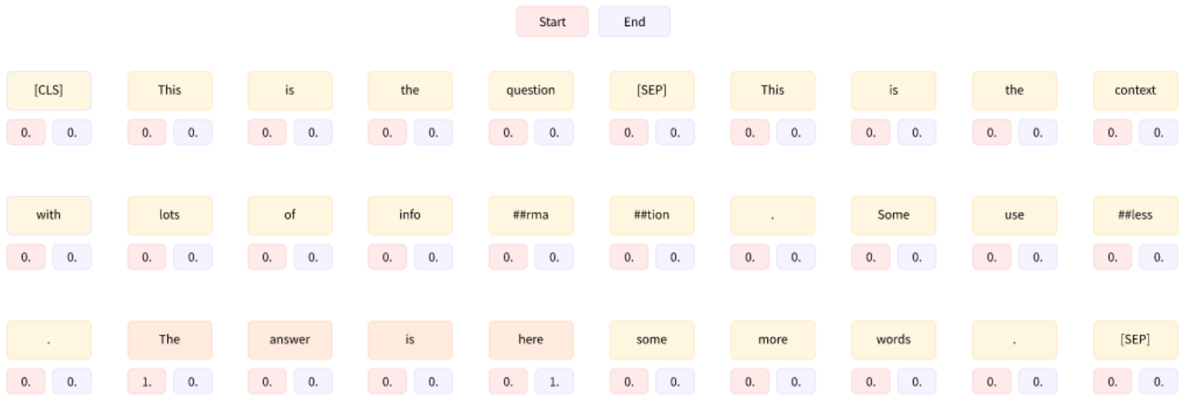

Figure 5.1.2 Indexing of the tokens

4. Lowercasing: Converting all the tokens to lowercase helps in reducing vocabulary size and ensuring consistency. It prevents the model from treating the same word in different cases as different tokens.

5. Removal of Special Characters and Punctuation: Removing special characters and punctuation marks is often necessary to improve the model's ability to understand the text. These characters can introduce noise and unnecessary complexity to the data.

6. Removal: Stopwords are commonly used words in a language (e.g., "a," "an," "the") that do not carry significant meaning and can be safely removed without affecting the context. Removing stopwords can reduce noise in the data and improve processing efficiency.

7. Normalization: Normalizing the text involves converting words to their base or canonical form. For example, reducing words to their root form (e.g., "running" to "run") or converting verbs to their base infinitive form can help in capturing the common meaning of related words.

8. Processing the validation data: The purpose of processing the validation data is to ensure its compatibility with the model and to evaluate the model's performance. Preprocessing the validation data will be slightly easier as we don’t need to generate labels (unless we want to compute a validation loss, but that number won’t really help us understand how good the model is). The real joy will be to interpret the predictions of the model into spans of the original context. For this, we will just need to store both the offset mappings and some way to match each created feature to the original example it comes from. Since there is an ID column in the original dataset, we’ll use that ID.

Preprocessing the training data in the SQuAD dataset helps in standardizing and preparing the text for the extractive QA model. It ensures that the model receives clean, consistent, and tokenized input, which is essential for effective training and accurate question answering.

## 5.2 Postprocessing the data

Postprocessing the data in an extractive question answering (QA) model, involves refining the model's output to provide more user-friendly and readable answers. The model will output logits for the start and end positions of the answer in the input IDs. We don’t need to compute actual scores (just the predicted answer). This means we can skip the softmax step. To go faster, we also won’t score all the possible (start_token, end_token) pairs, but only the ones corresponding to the highest n_best logits (with n_best=20). Since we will skip the softmax, those scores will be logit scores, and will be obtained by taking the sum of the start and end logits (instead of the product, because of the rule log(ab)=log(a)+log(b)).

To demonstrate all of this, we will need some kind of predictions. Since we have not trained our model yet, we are going to use the default model for the QA pipeline to generate some predictions on a small part of the validation set. We can use the same processing function as before; because it relies on the global constant tokenizer, we just have to change that object to the tokenizer of the model we want to use temporarily.

Now that the preprocessing is done, we change the tokenizer back to the one we originally picked. We then remove the columns of our eval_set that are not expected by the model, build a batch with all of that small validation set, and pass it through the model. If a GPU is available, we use it to go faster. Since the Trainer will give us predictions as NumPy arrays, we grab the start and end logits and convert them to that format:

start_logits = outputs.start_logits.cpu().numpy()

end_logits = outputs.end_logits.cpu().numpy()

Now, we need to find the predicted answer for each example in our small_eval_set. One example may have been split into several features in eval_set, so the first step is to map each example in small_eval_set to the corresponding features in eval_set.

With this in hand, we can really get to work by looping through all the examples and, for each example, through all the associated features. As we said before, we’ll look at the logit scores for the n_best start logits and end logits, excluding positions that give:

- An answer that wouldn’t be inside the context
- An answer with negative length
- An answer that is too long (we limit the possibilities at max_answer_length=30)

Once we have all the scored possible answers for one example, we just pick the one with the best logit score.

The final format of the predicted answers is the one that will be expected by the metric we will use. As usual, we can load it with the help of the Evaluate library. This metric expects the predicted answers in the format we saw above (a list of dictionaries with one key for the ID of the example and one key for the predicted text) and the theoretical answers in the format below (a list of dictionaries with one key for the ID of the example and one key for the possible answers).

Now let’s put everything we just did in a compute_metrics() function that we will use in the Trainer. Normally, that compute_metrics() function only receives a tuple eval_preds with logits and labels. Here we will need a bit more, as we have to look in the dataset of features for the offset and in the dataset of examples for the original contexts, so we won’t be able to use this function to get regular evaluation results during training. We will only use it at the end of training to check the results.

The compute_metrics() function groups the same steps as before; we just add a small check in case we don’t come up with any valid answers (in which case we predict an empty string).

We obtained strong results (83.0 EM, 88.25 F1 on SQuAD dev set).

### 5.3 Fine-tuning the model

Fine-tuning BERT on the SQuAD dataset allows the model to learn the patterns and representations necessary for accurate question answering. The model learns to predict the start and end positions of the answer span within the passage, leveraging BERT's contextual understanding of the language. This process tailors BERT to the specific task of extractive QA and enhances its performance on answering questions based on the given passages.

#### 5.3.1 Preparing everything for training

- First we need to build the DataLoaders from our datasets. We set the format of those datasets to "torch", and remove the columns in the validation set that are not used by the model. Then, we can use the default_data_collator provided by Transformers as a collate_fn and shuffle the training set, but not the validation set.
- Next we reinstantiate our model, to make sure we’re not continuing the fine-tuning from before but starting from the BERT pretrained model again.

- Then we will need an optimizer. As usual we use the classic AdamW, which is like Adam, but with a fix in the way weight decay is applied.

Once we have all those objects, we can send them to the accelerator.prepare() method. Remember that if you want to train on TPUs in a Colab notebook, you will need to move all of this code into a training function, and that shouldn’t execute any cell that instantiates an Accelerator. We can force mixed-precision training by passing fp16=True to the Accelerator (or, if you are executing the code as a script, just make sure to fill in the Accelerate config appropriately).

We can only use the train_dataloader length to compute the number of training steps after it has gone through the accelerator.prepare() method. We use the same linear schedule as in the previous sections.

To push our model to the Hub, we will need to create a Repository object in a working folder. First log in to the Hugging Face Hub, if you’re not logged in already. We’ll determine the repository name from the model ID we want to give our model.
Then we can clone that repository in a local folder. If it already exists, this local folder should be a clone of the repository we are working with. We can now upload anything we save in output_dir by calling the repo.push_to_hub() method. This will help us upload the intermediate models at the end of each epoch.

### 5.3.2 Training loop

We are now ready to write the full training loop. After defining a progress bar to follow how training goes, the loop has three parts:

- The training in itself, which is the classic iteration over the train_dataloader, forward pass through the model, then backward pass and optimizer step.

- The evaluation, in which we gather all the values for start_logits and end_logits before converting them to NumPy arrays. Once the evaluation loop is finished, we concatenate all the results. Note that we need to truncate because the Accelerator may have added a few samples at the end to ensure we have the same number of examples in each process.

- Saving and uploading, where we first save the model and the tokenizer, then call repo.push_to_hub(). As we did before, we use the argument blocking=False to tell the Hub library to push in an asynchronous process. This way, training continues normally and this (long) instruction is executed in the background.

### 5.3.3 Using the fine-tuned model

We can use the model we fine-tuned on the Model Hub with the inference widget. To use it locally in a pipeline, you just have to specify the model identifier.

For example:

```
from transformers import pipeline

# Replace this with your own checkpoint

model_checkpoint = "ashutosh2109 /bert-finetuned-squad"
question_answerer = pipeline("question-answering", model=model_checkpoint)

context = """
 Transformers is backed by the three most popular deep learning libraries — Jax, PyTorch and
TensorFlow — with a seamless integration
between them. It's straightforward to train your models with one before loading them for inference
with the other.
"""
question = "Which deep learning libraries back Transformers?"
question_answerer(question=question, context=context)
```

Output:

```
'score': 0.9979003071784973,
 'start': 78,
 'end': 105,
 'answer': 'Jax, PyTorch and TensorFlow'
```

This shows our model is working as well as the default one for this pipeline.

## 6. Applications:

- **Information Retrieval:** Extractive QA models can be used to retrieve specific information from large collections of documents or knowledge bases. Users can ask questions, and the model can provide precise answers by extracting relevant information from the available sources.

- **Customer Support and FAQs:** Companies often employ extractive QA models to automate customer support and frequently asked questions (FAQs) systems. Users can ask questions about products, services, or common issues, and the model can provide instant and accurate responses by extracting answers from the company's knowledge base or documentation.

- **News and Article Summarization:** Extractive QA models can summarize news articles or lengthy documents by extracting the most relevant information and answering questions about the key points. This allows users to quickly grasp the main ideas without reading the entire text.

- **Educational Assistance:** Extractive QA models can assist in education by answering questions from students or providing explanations for specific concepts. Students can ask questions related to their coursework, and the model can extract relevant information from textbooks, lecture notes, or educational resources to provide answers and explanations.

- **Legal and Compliance:** Extractive QA models can be used in the legal domain to assist with legal research, contract analysis, or compliance-related tasks. Lawyers, legal professionals, or compliance officers can ask questions about specific legal documents, regulations, or case law, and the model can extract relevant information to support their work.

- **Healthcare:** Extractive QA models have applications in healthcare for tasks such as medical question answering, patient education, and clinical decision support. Users, including patients or healthcare professionals, can ask questions about symptoms, diseases, treatment options, or drug interactions, and the model can provide answers by extracting information from medical literature or trusted sources.

- **Virtual Assistants and Chatbots:** Virtual assistants and chatbot applications can leverage extractive QA models to provide human-like interactions and answer user queries. Users can ask general knowledge questions, get recommendations, or seek assistance with various tasks, and the model can extract relevant answers from the available knowledge sources.

- **Technical Support:** Extractive QA models can be employed in technical support scenarios, such as IT help desks or troubleshooting systems. Users can ask questions about technical issues or error messages, and the model can extract relevant answers or solutions from technical documentation or knowledge bases to assist with problem resolution.

- **Interviewing and Recruitment:** Extractive QA models can be utilized in the interviewing and recruitment process. Interviewers can ask questions to assess a candidate's knowledge, skills, or problem-solving abilities, and the model can extract answers from the candidate's resume or relevant documents for evaluation.

- **Language Learning:** Extractive QA models can aid language learning by providing answers and explanations to language-related questions. Users can ask questions about grammar rules, vocabulary, idioms, or language usage, and the model can extract information from language resources to provide accurate responses.

These are just a few examples of the use cases where extractive QA models can be applied. The versatility and accuracy of these models make them valuable in a wide range of industries and applications, facilitating quick access to relevant information and improving user experiences.

## 7. Future Scope

- **Model Performance:** Enhancing the performance of extractive QA models is an ongoing focus. Future research can explore novel architectures, model ensembles, and techniques to further improve accuracy, especially in handling complex questions and challenging passages. Continued advances in pre-training techniques, such as self-supervised learning, can also contribute to better model performance.

- **Multi-Lingual QA:** Expanding extractive QA models to support multiple languages is an important future direction. Currently, most models are trained and evaluated on English-language datasets like SQuAD. Extending QA models to other languages would enable wider accessibility and applicability in diverse linguistic contexts.

- **Cross-Domain Generalization:** Extractive QA models often struggle to generalize well across different domains. Future research can focus on techniques that enable models to adapt and perform well in various domains, even with limited or no domain-specific training data. This would enhance the versatility and practicality of extractive QA models.

- **Multi-Hop Reasoning:** Extractive QA models primarily focus on answering questions that require information from a single passage or document. Enabling models to perform multi-hop reasoning, where answers may require information aggregation or inference from multiple sources, is an important future research direction. This would allow QA models to handle more complex queries that involve deeper understanding and reasoning.

- **Contextual Understanding:** Extractive QA models primarily rely on the local context within the passage to identify the answer span. Future work can explore techniques to incorporate a broader context, such as a collection of related documents or a knowledge graph, to improve the model's understanding and ability to answer questions that require global context.

- **Beyond Extractive Answers:** While extractive QA models focus on extracting the answer span from the passage, future research can explore methods to generate more abstractive answers. This would involve generating answers in a more natural language form, summarizing relevant information from the passage, or providing additional context to enhance the answer's quality and readability.

- **Adversarial Evaluation:** Developing robust evaluation methodologies for extractive QA models is an ongoing challenge. Future work can explore adversarial evaluation techniques to identify and address model weaknesses, such as sensitivity to paraphrases, adversarial examples, or subtle linguistic variations. This would help ensure the reliability and generalizability of extractive QA models.

- **Real-Time QA:** Enabling extractive QA models to operate in real-time scenarios with low latency requirements is an important future scope. Efficient model architectures, optimization techniques, and hardware acceleration can contribute to real-time question answering, enabling applications in chatbots, virtual assistants, and other interactive systems.

- **Domain-Specific QA:** Extractive QA models can be further specialized for specific domains, such as healthcare, legal, or technical domains. Customized training on domain-specific datasets and incorporating domain-specific knowledge sources can improve the model's performance in specialized domains.

- **Explainability and Trust:** Addressing the black-box nature of extractive QA models is crucial for building trust and understanding their decision-making. Future research can focus on developing techniques to provide explanations for model predictions, highlighting the relevant evidence used to generate the answer, and enabling users to understand and verify the model's reasoning process.

These future scopes hold immense potential for advancing extractive QA models, making them more accurate, versatile, and applicable across various domains and languages. Continued research and innovation in these areas will contribute to the development of more advanced and capable extractive QA systems.

## 8. Conclusions

In this work, we propose a novel method for Supervised span selection. We showed that by finetuning a pretrained model, one can get considerable gains in terms of time, money and power. You can use Question Answering (QA) models to automate the response to frequently asked questions by using a knowledge base (documents) as context. Answers to customer questions can be drawn from those documents.

This project focused on training an extractive question answering (QA) model by fine-tuning the BERT (Bidirectional Encoder Representations from Transformers) model on the SQuAD (Stanford Question Answering Dataset) dataset. The objective was to enable the model to accurately predict answer spans given a passage and a corresponding question.

Through the project, we followed a comprehensive methodology, including preprocessing the training data, processing the validation data, and postprocessing the model predictions. We fine-tuned the BERT model by updating its parameters using backpropagation and optimizing it with a suitable loss function.

The results of the fine-tuning process were evaluated on the validation set using metrics such as the F1 score and exact match (EM) score to measure the model's performance. Hyperparameter tuning was performed to optimize the model's configuration.

By fine-tuning BERT on the SQuAD dataset, we successfully adapted the powerful language model for the extractive QA task. The model demonstrated the ability to accurately extract answer spans from the given passages, showing promising performance in providing precise answers to user questions.

However, there are still opportunities for further improvement and exploration. Future research could focus on enhancing the model's performance by investigating novel architectures, incorporating multi-hop reasoning, enabling better cross-domain generalization, and exploring abstractive answer generation.

Overall, this project highlights the effectiveness of fine-tuning BERT on the SQuAD dataset for training an extractive QA model. The learned model can be leveraged in various applications, such as information retrieval, customer support, education, and more, where accurate and efficient question answering is required.

# 9. References

1. Stalin Varanasi, Saadullah Amin, Günter Neumann. AutoEQA: Auto-Encoding Questions for Extractive Question Answering, Saarland Informatics Campus, D3.2, Saarland University, Germany Geman Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany.

2. Salvador Marti Roman. Active Learning for Extractive Question Answering. Linköping University | Department of Computer and Information Science Master's thesis, 30 ECTS | Statistics and Machine Learning 2022

3. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

4. Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). SQuAD: 100,000+ Questions for Machine Comprehension of Text. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 2383-2392).

5. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv preprint arXiv:1907.11692.

6. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2020). HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv preprint arXiv:2006.03654.

7. Sun, Z., Yu, Z., Wang, W., Liu, X., & Pang, L. (2019). ERNIE 2.0: A Continual Pre-training Framework for Language Understanding. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP) (pp. 4370-4379).

8. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova Google AI Language