

# **Introduction Natural Language Processing CL7.401**

Spring 2023, IIIT Hyderabad

## **Project Final Report**

### **Question Answering**

**Yash Bhaskar (2021114012)  
Chinmay Pateria (2021114013)  
Tammanna Meghasyam(2019101077)**

**Manish Shrivastava  
Instructor's Name**



**INTERNATIONAL INSTITUTE OF  
INFORMATION TECHNOLOGY**

**HYDERABAD**

# DECLARATION

We, **Yash Bhaskar (Roll No: 2021114012)**, **Chinmay Pateria (Roll No: 2021114013)** and **Tammana Meghasyam (2019101077)** hereby declare that, this report entitled “**Question Answering Language Model**” submitted to International Institute of Information Technology, Hyderabad towards the Project Work of INLP (Intro to Natural Language Processing), is an original work carried out by us under the supervision of **Prof Manish Shrivastava** and has not formed the basis for the academic work in this or any other institution or university. We have sincerely tried to uphold academic ethics and honesty. Whenever a piece of external information or statement or result is used then, that has been duly acknowledged and cited.

IIIT Hyderabad - 500 032

April 2023

# ACKNOWLEDGEMENT

We want to extend a sincere and heartfelt obligation towards all the personages with- out whom the project completion was impossible. We express our profound gratitude and deep regard to Dipti Misra Sharma, IIIT Hyderabad, for her guidance, valuable feedback, and constant encouragement throughout the project. Her valuable suggestions were of immense help. We sincerely acknowledge her constant support and guidance during the project.

We are immensely grateful to **Prof Manish Shrivastava** and **Suyash Mathur** for their constant support and encouragement. I am also grateful to the International Institute of Information Technology, Hyderabad, for allowing us to do this project and providing all the required facilities.

IIIT Hyderabad - 500 032

April 2023

# ABSTRACT

---

Name of the student: Yash Bhaskar

Roll No: 2021114012

Name of the student: Chinmay Pateria

Roll No: 2021114013

Name of the student: Tammana Meghasyam

Roll No: 2019101077

Course for which submitted: Intro to Natural Language Processing (CL2.203)

Project title: **Question Answering Language Model**

Project supervisor: **Prof Manish Shrivastava**

Month and year of Project submission: **April, 2023**

---

*To create a question answering system based on SQuAD 2.0 with the goal of maximizing the F1 score for the dataset. You are using an ensemble approach by combining four different language models, namely BiLSTM, BIDAf, BERT Cased, and DistilBERT, to achieve the highest possible F1 score.*

# Content

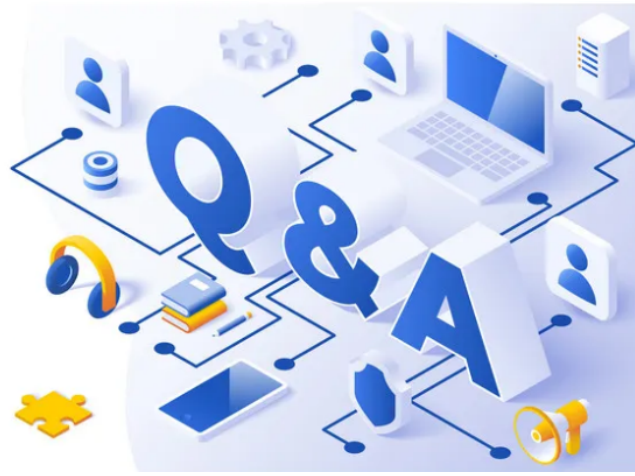
1. Introduction	7
2. Problem Statement	8
3. Dataset Analysis	8
4. Challenges	8
5. Methodology	8
5.1 BILSTM	8
5.2 BiDAF	8
5.3 BERT Cased	8
5.4 Distil Bert	8
5.5 Ensembling	8
6. Result	8
7. Discussion	8
8. Conclusion	8
9. Reference	8

## Question and Answer

Lorem ipsum dolor sit amet.

Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

more



## Introduction

Question Answering (QA) is a subfield of Natural Language Processing (NLP) that focuses on automatically answering questions posed in natural language. *The goal of a QA system is to understand the questions asked and provide a concise and relevant answer.*

QA systems have many practical applications including knowledge management, customer service, and education. They can help to reduce the time and effort required to find answers to questions and provide more accurate and relevant answers compared to traditional search engines.

One of the most successful deep learning models in the field of NLP is Bidirectional Encoder Representations from Transformers (BERT). BERT is a pre-trained language model that has shown state-of-the-art performance in various NLP tasks, including QA.

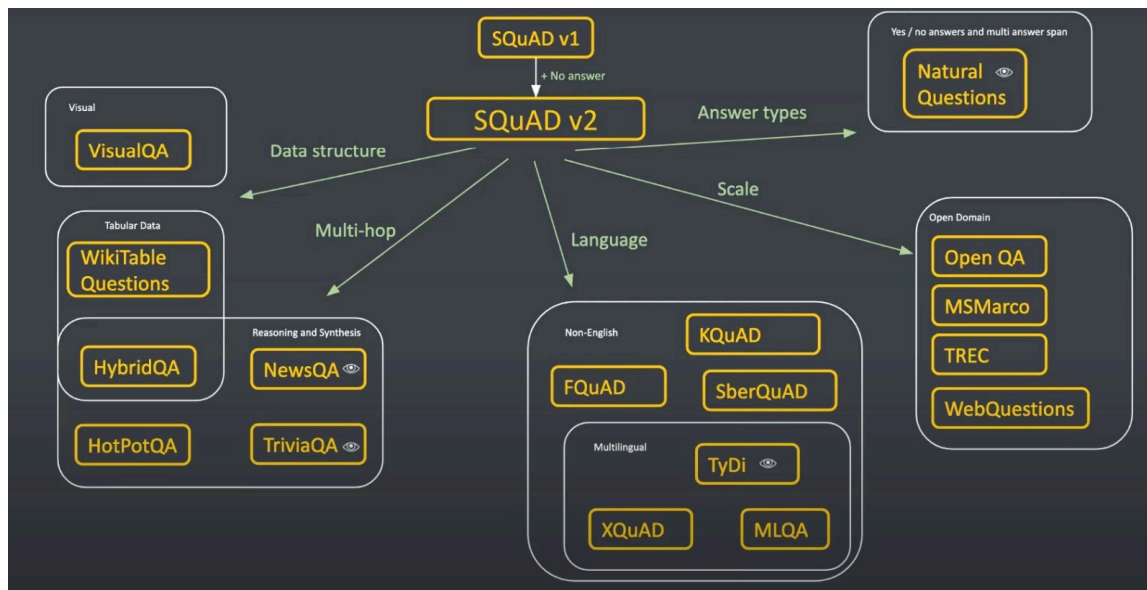
## Problem Statement:

**Goal:** *To create a question answering system based on SQuAD 2.0 with the goal of maximizing the F1 score for the dataset. You are using an ensemble approach by combining four different language models, namely BiLSTM, BIDAf, BERT Cased, and DistilBERT, to achieve the highest possible F1 score.*

## Dataset Analysis:

We have explored various datasets such as SQuAD, MS MARCO, BioASQ, TREC QA, Natural Questions (Google AI), NarrativeQA, and HotpotQA for training and evaluating QA models. Each dataset has its own characteristics, and the choice of dataset should depend on the research question and the specific requirements of the project.

For our project, we have decided to use the SQuAD dataset for training and evaluating the QA model. It contains questions and answers based on passages from a variety of Wikipedia articles, making it a valuable resource for developing QA models that can read and comprehend text. The dataset consists of context paragraphs and questions, and the model is expected to predict the answer span within the context paragraph.

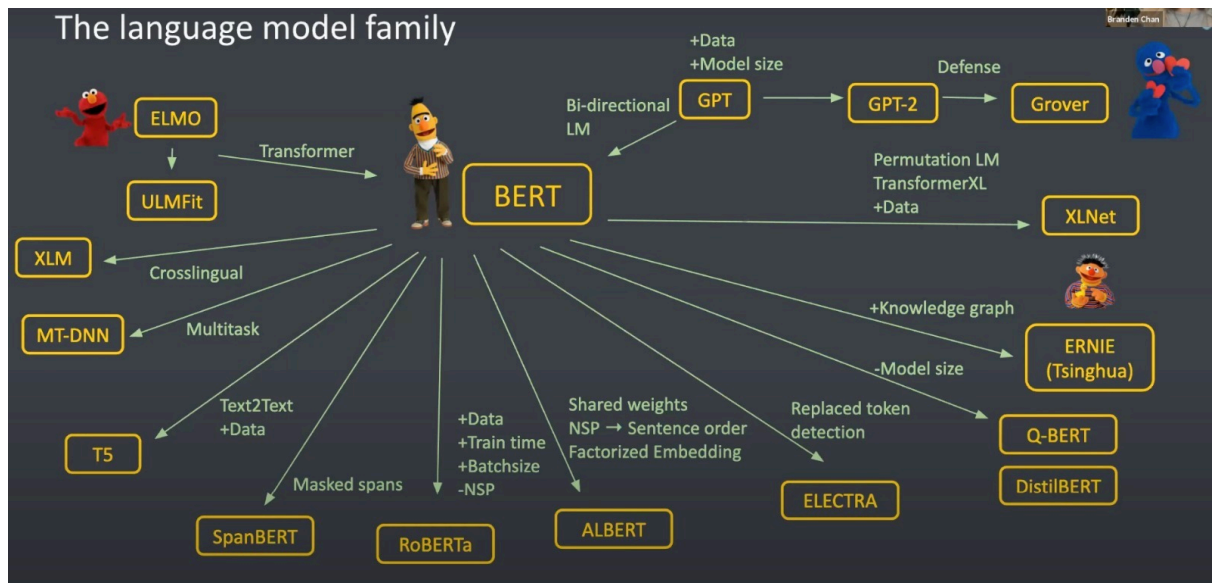


The dataset consists of over 100,000 questions and their corresponding answer spans, all of which are based on more than 500 Wikipedia articles. In this analysis, we will discuss various aspects of the SQuAD dataset and how they can inform the development of QA models.

1. **Data Distribution:** The SQuAD dataset is evenly split between train and dev sets, with each set containing approximately 50,000 examples. The dataset is highly skewed towards shorter questions, with a majority of the questions consisting of 20 words or fewer. Similarly, a majority of the answer spans are also relatively short, typically consisting of fewer than 5 words. This distribution may influence the choice of model architecture, as longer questions and answer spans may require more complex models.
2. **Answerability:** Not all questions in the SQuAD dataset are answerable from the given passage. Roughly 30% of the questions in the dataset are considered unanswerable, as the answer is not present in the passage. This presents a challenge for QA models, as they need to be able to identify unanswerable questions and provide an appropriate response.
3. **Answer types:** The SQuAD dataset contains questions that require different types of answers, including named entities, dates, locations, and numerical values. Understanding the types of answers required for a given question can inform the design of the QA model, as different answer types may require different processing steps.
4. **Contextual understanding:** Many questions in the SQuAD dataset require a deep understanding of the context in order to provide an accurate answer. For example, the answer to the question "What type of rock is basalt?" requires an understanding of the definition of basalt and its classification within the field of geology. This suggests that QA models that incorporate contextual understanding, such as language models based on transformer architectures, may perform well on the dataset.
5. **Evaluation:** The primary evaluation metric for SQuAD is the F1 score, which

measures the overlap between the predicted answer span and the ground truth answer span. In addition to the F1 score, SQuAD also uses the Exact Match (EM) metric, which measures whether the predicted answer exactly matches the ground truth answer. Both metrics are important to consider when developing a QA model, as a high F1 score may indicate that the model is identifying the correct answer span, but a low EM score may indicate that the model is not providing exact answers.

## Challenges:



### Choosing models for ensembling:

- Ensuring diversity in the models selected to avoid redundancy.
- Evaluating the strengths and weaknesses of each model to identify complementary models.
- Ensuring the models are compatible with each other in terms of input/output format, hyperparameters, and training techniques.
- Determining the computational requirements and scalability of each model to ensure feasible ensemble training.

### Choosing a dataset:

- Selecting a dataset that is representative of the domain and language of interest to maximize the generalization of the model.
- Ensuring that the dataset is of sufficient size and quality to support effective model training and evaluation.

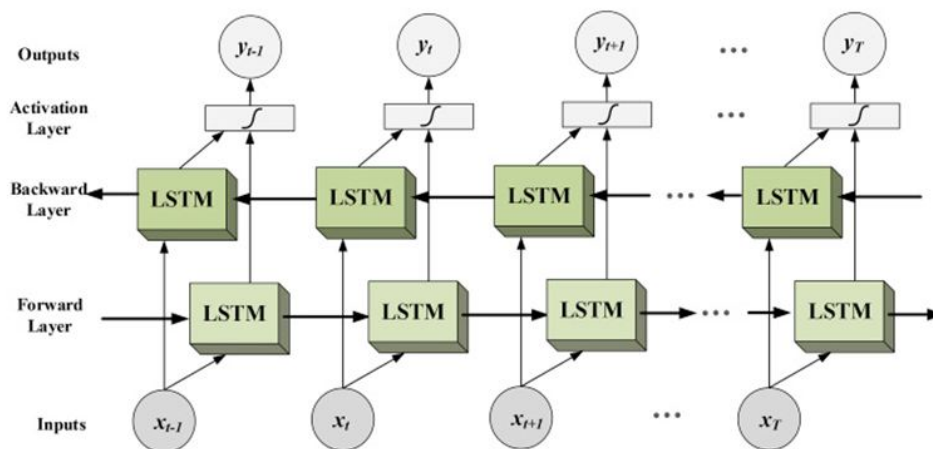


- Evaluating the relevance of the dataset to the task at hand to ensure that the model learns meaningful patterns.
- Considering the potential ethical implications of the dataset, such as biases or sensitive information.

## Methodology:

1. Data Preparation: We downloaded the Squad dataset and preprocessed the data to extract questions, contexts, and answers for each example.
2. Model Selection: We selected four different models for question answering, including BiLSTM, BIDAf, BERT Cased, and DistilBERT.
3. Model Training: We trained each model on the Squad dataset using the training split and evaluated their performance on the validation split.
4. Performance Evaluation: We used the EM and F1 scores to evaluate the performance of each model on both the training and validation datasets.
5. Ensembling: We created an ensembling model that selected the answer with the highest F1 score among the four models for each question answer pair.
6. Performance Evaluation of Ensembling: We evaluated the performance of the ensembling model on the train and validation datasets using the EM and F1 scores.
7. Result Analysis: We analyzed the results to identify the strengths and weaknesses of each model and the ensembling approach.
8. Discussion and Conclusion: We discussed the results and drew conclusions based on the findings, highlighting the strengths and limitations of the study and suggesting potential directions for future work.

### 1. Model 1 based on BiLSTM



- Embeddings - GloVe:

Loading pre-trained GloVe word embeddings and transforming word tokens into embeddings. The `get_glove_dict()` function parses the GloVe word vectors text file and returns a dictionary with the words as keys and their respective pre-trained word vectors as values. The `embed()` function takes a list of word tokens as input, transforms each token to lowercase, and then checks whether it is present in the GloVe embeddings dictionary. If the token is present, the corresponding word vector is appended to the vectors list; otherwise, the `unknown_vector` (pre-computed vector for unknown words) is appended to the list. Finally, the function returns a NumPy array of the concatenated word vectors. The code applies the `embed()` function to the Paragraph and Question columns of the `train_ds` and `val_ds` data frames, which contain the tokenised versions of the paragraphs and questions, respectively. The resulting embeddings are then used as inputs to the model during training and evaluation.

- **Feature Extraction - Bidirectional LSTM encoder:**

The input shape for the paragraph part is `(paragraph_length, embedding_size)`, meaning each paragraph token is represented as a vector of size `embedding_size`. The Masking layer masks any padded tokens with a vector of zeros. The Bidirectional layer creates a forward and backward LSTM layer, allowing the model to capture context from both directions. The output from this layer is a sequence of encoded paragraph vectors of shape `(paragraph_length, embedding_size*2)`.

The input shape for the question part is `(question_length, embedding_size)`, with the same masking and bidirectional layers as the paragraph encoder. The output from this layer is a sequence of encoded question vectors of shape `(question_length, embedding_size*2)`.

In LSTM layer, each token in the input sequence is fed into the LSTM cell one at a time, along with a hidden state vector and a memory state vector. The LSTM cell processes the input token and updates its internal hidden and memory state vectors. These updated state vectors are then passed on to the next token in the sequence. The final output of the LSTM layer is the final hidden state vector, which has information about the entire input sequence. The `return_sequences=True` argument ensures that the LSTM layer outputs a sequence of hidden state vectors for each input token, rather than just the final hidden state vector.

Thus, bypassing the paragraph and question inputs through their respective LSTM encoders, the model can capture the context and relationship between the input tokens. This encoded information can then be used to answer the question given the context in the paragraph.

- **Prepare training and validation data:**

Preparing the data for training and validation of a machine learning model for question-answering task. The data consists of paragraphs, questions and their respective answers. The `pad_paragraph` and `pad_question` functions are used to pad the paragraph and question embeddings with zero vectors to make them of equal length. The maximum length of paragraphs and questions are pre-defined as `paragraph_length` and `question_length`, respectively. These functions are applied to the 'Paragraph' and 'Question' columns of the train and validation datasets using the `map` function. The resulting padded paragraphs and questions are converted to Python lists.

The answers' starting and ending token positions are extracted from the 'Answer' column of the train and validation datasets. The start and end positions are stored in separate lists `start_train`, `end_train` and `start_val`, `end_val`.

Finally, all the data is converted into constant tensors using the `tf.constant` function. The padded paragraphs and questions are converted to `np.float32` dtype, while the start and end

positions are kept as np.float32. These tensors will be used to train and validate the machine-learning model.

- **Paragraph-Question Interaction - Bidirectional attention + One-hop interaction:**

Implement the co-attention mechanism, a technique used to model the interaction between two input sequences: a paragraph and a question. The co-attention layer produces a weighted representation of each input sequence conditioned on the other.

First, the code computes a scoring matrix by taking the dot product between the encoded paragraph and the encoded question and transposing the question matrix to match the dimensions. The resulting score matrix has dimensions (batch\_size, paragraph\_length, question\_length).

Next, the code applies a softmax function along the question dimension to compute a set of question weights for each paragraph word. It applies a second softmax function along the paragraph dimension to computing a set of paragraph weights for each question word.

The code then computes a question context vector for each paragraph word by taking the weighted sum of the question-encoded matrix using the question weights. This step computes how much each question word contributes to each paragraph.

The code then concatenates the question-encoded matrix with the question context matrix along the feature dimension. The resulting tensor has dimensions (batch\_size, 2 \* embedding\_size, question\_length), which represent a combined representation of the question and its contextual information for each paragraph word.

Finally, the code computes a paragraph context vector for each question word by taking the weighted sum of the concatenated tensor along the paragraph dimension, using the paragraph weights. This step computes how much each paragraph contributes to each question word.

The resulting tensor has dimensions (batch\_size, 2 \* embedding\_size, paragraph\_length), representing a combined representation of the paragraph and its contextual information for each question word. This tensor is used as input to the final layer of the model to make predictions.

- **Span prediction - Unidirectional boundary model:**

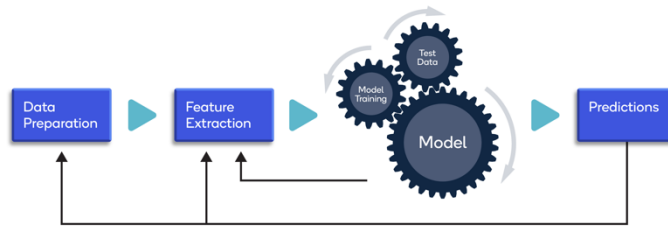
The code implements the answer pointer layer in the model. This layer takes in the output from the co-attention layer, which is the paragraph context, and predicts the start and end positions of the answer in the paragraph.

First, an LSTM layer is applied to the paragraph context to capture the boundary information.

Then, a dense layer with one output is used to predict the start position logits. The output of this layer is concatenated with the output of the previous LSTM layer, and another dense layer with one output is used to predict the end position logits.

Finally, the softmax activation function is applied to both start and end position logits to obtain the probability distributions over all possible positions in the paragraph. These probability distributions can extract the answer's predicted start and end positions.

- **Training :**



The **fit()** function is called on the **model** object, which takes the training data (**paragraph\_train** and **question\_train**) and the ground truth labels (**start\_train** and **end\_train**) as input. The **epochs** parameter specifies the number of times the model should iterate over the entire training dataset.

The **validation\_data** parameter specifies the validation data and ground truth labels, which are used to evaluate the performance of the model during training. The **fit()** function returns the history object, which contains information about the training process, such as the training and validation loss and accuracy for each epoch.

During training, the model updates its weights using the optimiser specified during the model compilation, to minimise the loss function. The loss function in this case is the sum of the cross-entropy losses for the start and end positions. The **fit()** function also prints out the training and validation loss and accuracy for each epoch, which can be used to monitor the training progress.

- **Build, Fit and Evaluate Model:**

The custom metrics defined are:

- **cross\_entropy\_loss** calculates the cross-entropy loss between the predicted start and end positions and the true start and end positions. It uses the negative logarithm of the predicted values and returns the start and end losses sum.
- **exact\_match**: this metric calculates the exact match score between the predicted and true start and end positions. If the predicted start and end positions match the true positions, the score is 1. Otherwise, it is 0.
- **f1\_score**: this metric calculates the F1 score between the predicted and true start and end positions. It calculates the precision and recall for the predicted positions and returns the harmonic mean of precision and recall.

The model is compiled with these custom metrics and is ready to be trained. The model.summary() function shows the model's architecture, which includes the input and output shapes and the number of trainable parameters in each layer.

Further generates predictions on the training dataset using the BiLSTM model trained earlier. It iterates through each row of the training dataset **train\_ds\_testing** and extracts the **id**, **paragraph**, **question**, **answer\_start**, and **answer** values.

It then uses the trained BiLSTM model **model3** to predict the start and end positions of the answer within the **paragraph**. It extracts the substring of the **paragraph** between the predicted start and end positions as the predicted answer **m3**.

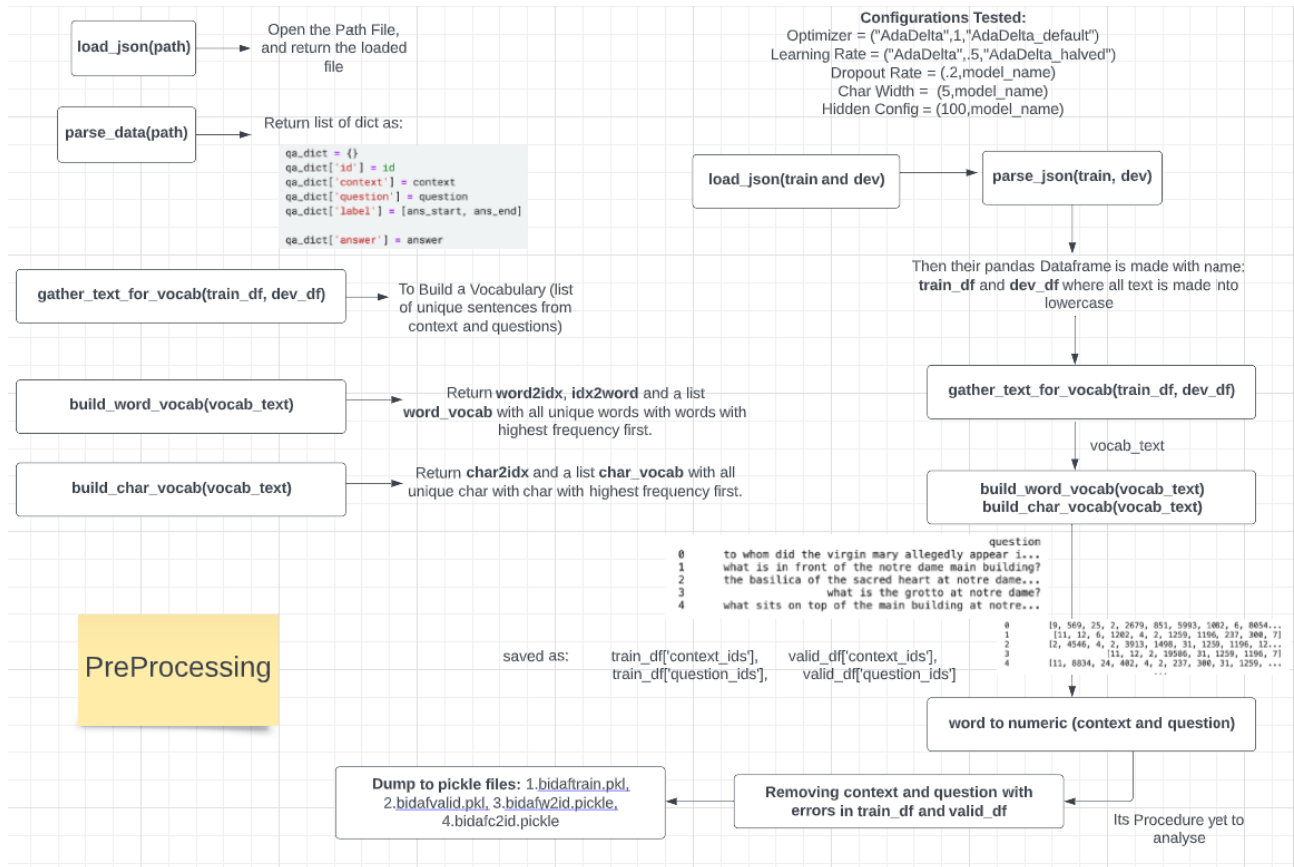
The code then calculates the precision, recall, and F1 score at the character level between the predicted answer **m3** and the actual answer. It then stores the predicted answer **m3** and the

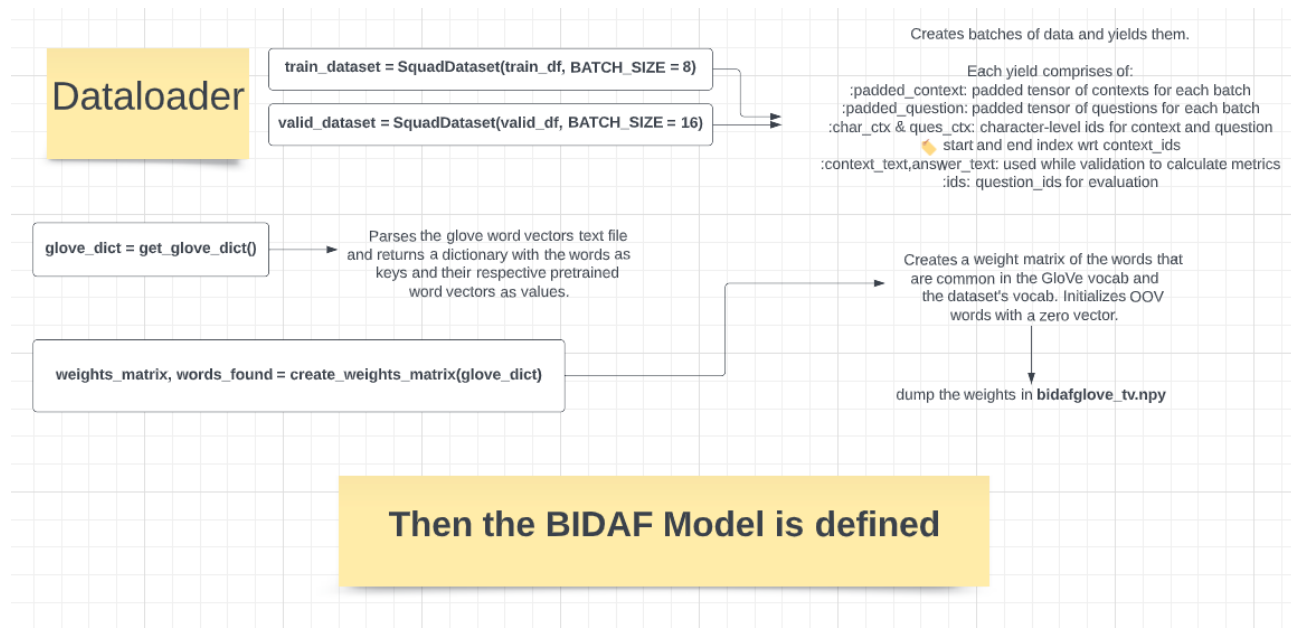
F1 score in a dictionary **predictions** with the **id** as the key.

Finally, the predicted answers and their corresponding F1 scores are dumped into a JSON file named **prediction\_trainDataset\_BiLSTM.txt**.

## 2. Model 2 based on BIDAf

The flow of data in the BIDAf Model is:





The Score while Fine Tuning the BIDAf Model for the following Configurations are:

### 1. Dropout.15

```

Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 5.663719580305516
Epoch EM: 29.526963103122043
Epoch F1: 44.526552180965695
=====
  
```

### 2. AdaDelta\_default

```

Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 4.036482042368205
Epoch EM: 49.03500473036897
Epoch F1: 62.08789879592778
  
```

### 3. AdaDelta\_halved

```

Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 3.8334484500225448
Epoch EM: 52.63008514664144
Epoch F1: 64.6725768197938
=====
  
```

#### 4. Adam\_default

---

```
Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 5.268989314473801
Epoch EM: 29.621570482497635
Epoch F1: 41.960928466992705
```

---

#### 5. SGD\_default

```
Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 4.4986774719221865
Epoch EM: 42.06244087038789
Epoch F1: 55.85456148393064
```

#### 6. Dropout-.25

```
Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 5.459608145567804
Epoch EM: 29.64995269631031
Epoch F1: 44.121139564928974
```

#### 7. Hidden50

```
Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 4.440065040484091
Epoch EM: 40.05676442762535
Epoch F1: 54.848472866890155
```

---

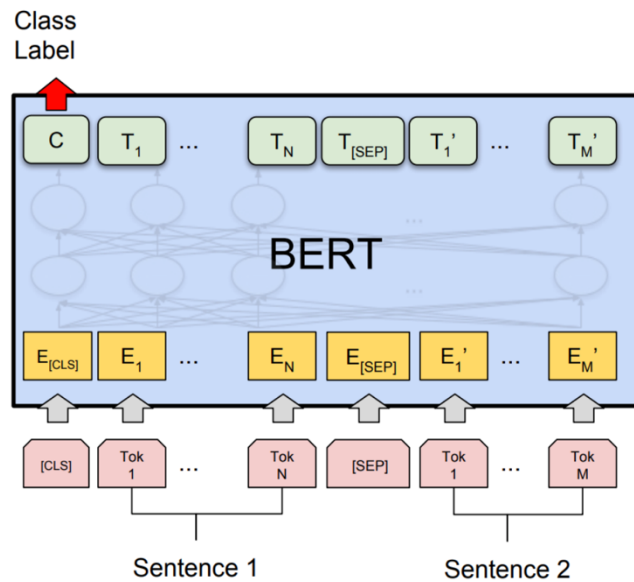
#### 8. AdaDelta\_halved and optim Adam

---

```
Epoch 1
Starting validation .....
Starting batch 0
Starting batch 500
Starting batch 1000
Starting batch 1500
Starting batch 2000
Epoch valid loss: 3.7161620472158705
Epoch EM: 54.512771996215704
Epoch F1: 66.49969502527695
```

---

### 3. Model 3 based on BERT cased



There are various sizes and variations of the BERT (Bidirectional Encoder Representations from Transformers) model, including BERT-base-cased. This BERT model, which has 12 transformer layers, 768 hidden units, and 12 self-attention heads, is one of the scaled-down variations. In languages like German, where case distinctions can alter the meaning of a phrase, the "cased" in the model's name denotes that the model uses case information in its training.

BERT-base-cased was pre-trained on massive volumes of text data, particularly on the BooksCorpus and English Wikipedia, like previous BERT models. A smaller labelled dataset was used to focus it on a particular goal, such as sentiment analysis or question-answering.

For a variety of natural language processing (NLP) tasks, BERT-base-cased is a popular option because it strikes a reasonable compromise between accuracy and computational resources. It can be fine-tuned for numerous downstream tasks, including text classification, question-answering, and named entity recognition, among others. It is a suitable starting point for many NLP applications, especially if you have low computational resources.

#### Overview :

- Defining the BERT tokenizer and model after importing the required libraries.
- Importing pretrained model
- Preprocessing the SQuAD 2.0 training data after reading it from a JSON file will produce training examples for the BERT model.
- Calculating the total number of training steps and specify the number of



training epochs.

- Utilising the SQuAD 2.0 preprocessed dataset to train the BERT model.
- Saving the weights of the trained model to a file.
- Creating a formula for anticipating responses to queries based on context.
- Reading test data for SQuAD 2.0 from a JSON file.
- Utilising the trained BERT model to forecast answers to questions in the test data.
- Creating a JSON file with the anticipated responses.

### Detailed Explanation:

During preprocessing we used bert tokenizer which takes question , context ,addspecialtoken, maxseqlength, padding , truncating etc and gives us input\_ids which are token ids assigned to tokens , token\_type\_ids which are used to differentiate various tokens present in the input\_ids. Like it will assign value 0 to question tokens , value 1 to context tokens , value 2 to special tokens.

For every question , context and answer we generate the above mentioned information by using tokenizer and we calculate start position and end position of answer in the entire tokens generated and we keep all the data that we just computed in the form a dictionary and we append all these dictionaries in to array which basically becomes a training data.

Now we use below code to make our optimiser and scheduler.

```
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)  
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,  
num_training_steps=num_training_steps)
```

Now we train the code by making batches of batchsize = 32 and for each batch we make a new tensor specially for the current batch and we pass it through the model to get output . After generating output we can calculate loss and backpropagate it using optimizer and scheduler.

Once the training is done we can save the model check points.

Now we test our test data by making batches of specific size and passing these batches to a function which takes batch of questions , batch of contexts ,model and tokenizer which returns batch of computed answers.

We store these information in a dictionary where we key is id of question and answer is the answer computed. In this way we generate a predicted test file for test data which can later be used for calculating f1 score .

One other approach to solve this problem is to use a simple transformers which has QuestionAnsweringModel, QuestionAnsweringArgs modules in it. QuestionAnswering Model can load pretrained bert-base-cased and can train the model according to the arguments we define in Question Answering Args modules in it. This approach can completely eliminate bert tokeniser approach discussed above and can make the code simpler and readable for the coder.

The more the number of epochs trained the better the results that we achieved. Loss really decreased in between epochs making model a better one . During epoch one the loss was at 0.9 then it dropped to 0.53 and then to 0.43 and then the model might reach local minimum after 2 more epochs which requires us to change the hyperparameters to make model to move from the minima .

Epochs 0/6. Running Loss: 0.9148: 100%  1029/1029 [25:34<00:00, 1.16s/it]

```
convert squad examples to features:  0%|          | 0/11873 [00:00<?, ?it/s]
convert squad examples to features:  0%|          | 1/11873 [00:42<141:11:21, 42.81s/it]
convert squad examples to features: 100%|██████████| 11873/11873 [01:24<00:00, 140.88it/s][A
```

```
add example index and unique id: 100%|██████████| 11873/11873 [00:00<00:00, 698805.43it/s]
```


Running Evaluation: 100%  190/190 [00:46<00:00, 4.02it/s]

Running Evaluation: 100%  190/190 [00:46<00:00, 4.10it/s]

Epochs 1/6. Running Loss: 0.5964: 100%  1029/1029 [24:09<00:00, 1.16s/it]

Running Evaluation: 100%  190/190 [00:46<00:00, 4.10it/s]

Running Evaluation: 100%  190/190 [00:46<00:00, 4.09it/s]

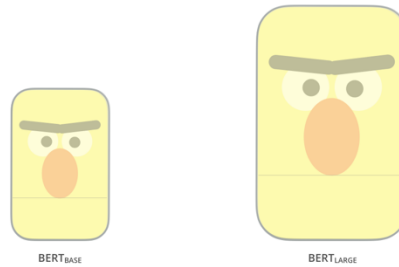
Epochs 2/6. Running Loss: 0.4377: 16%  160/1029 [03:34<19:26, 1.34s/it]



[Link for model](#)

.....

## 4. Model 4 based on Distil Bert



- **Description of Functions for Parsing Squad Dataset :**

The `load_json(path)` function takes the path of the JSON file as input and returns the JSON object of the dataset. It first opens the file using `open()` function and reads it using `json.load()` function, both of which are built-in Python functions. It then prints the length of the data and the keys of the first data item for debugging purposes and returns the data.

The `parse_data(data)` function takes the JSON object of the dataset as input and returns a list of dictionaries. It first extracts the data list from the JSON object and initialises an empty list to store the parsed data. It then loops through the data list and extracts the context and qas values from each item. For each qas item, it extracts the id, question, answers, answer\_start and answer\_end values and creates a dictionary containing these values as keys. Finally, it appends the dictionary to the list of parsed data.

The output of the `parse_data(data)` function is a list of dictionaries, where each dictionary contains the context, question, and label (start and end indices) for an answer in the Squad dataset.

- **Loading and preprocessing Squad dataset :**

Loads the train and validation datasets in JSON format using the `load_json(path)` function defined earlier. It then calls the `parse_data(data)` function for each dataset to parse the data and convert it into a list of dictionaries containing the context, question, and label information. Next, it prints the length of the two lists to verify that the data has been parsed successfully. It then converts the two lists of dictionaries into pandas DataFrames using `pd.DataFrame()`.

Finally, it renames the columns of the DataFrames and extracts the first element of the Answer Start column using `apply()` function. This is because the Answer Start column contains a list of start and end indices of the answer, but we only need the start index for our purposes.

- **Refining Answer Positions:**

Extracts the necessary columns from the `train_ds` and `val_ds` DataFrames and converts them into lists. Specifically, it retrieves the id, Paragraph, and Question columns from both DataFrames and converts them into lists using the `tolist()` function. It also retrieves the Answer column from both DataFrames and stores them in `train_labels` and `val_labels`.

The code then performs a list comprehension on the `paragraph_train` and `paragraph_val` lists to convert each string in the list into a new string. This step appears unnecessary and does not modify the original strings in any way, so it could be removed without affecting the code's functionality.

This code prepares the data for training and validation by converting the necessary columns into lists for further processing.

- **Evaluating question-answering model performance :**

Defines two functions for evaluating the performance of a question-answering model based on the exact match and F1 score metrics.

In addition to computing the F1 score, the `f1_score` function also updates a global dictionary `id2f1` with the F1 score for each example in the validation set. The keys of the dictionary are the IDs of the validation examples, and the values are the corresponding F1 scores.

## 5. Ensembling Model

We used an ensembling approach to combine the predictions of three separate models for the SQuAD dataset. Specifically, we created three text files containing question-answer pairs with their corresponding F1 scores for each model. Our ensembling model selected the answer with the highest F1 score from the three models, effectively leveraging their individual strengths to arrive at a more accurate overall prediction. While this approach is simple, it proved to be effective in improving the overall performance of our model. However, we acknowledge that there may be limitations to this approach, such as overfitting to the training data or the need for additional evaluation metrics beyond F1 score.

```
=====
EM Score on Validation Dataset using BIDAf Model = 0.5447480447480447
F1 Score on Validation Dataset using BIDAf Model = 0.6680005949338813
=====
EM Score on Train Dataset using BIDAf Model = 0.49979049303971324
F1 Score on Train Dataset using BIDAf Model = 0.6512755191469993
=====
EM Score on Validation Dataset using BiLSTM Model = 0.0004433060782188947
F1 Score on Validation Dataset using BiLSTM Model = 0.28518230394810734
=====
EM Score on Train Dataset using BiLSTM Model = 0.0005989334377627533
F1 Score on Train Dataset using BiLSTM Model = 0.26962483004132115
=====
EM Score on Validation Dataset using DistilBERT Model = 0.5613732637178603
F1 Score on Validation Dataset using DistilBERT Model = 0.7530296296892632
=====
EM Score on Train Dataset using DistilBERT Model = 0.705635733290333
F1 Score on Train Dataset using DistilBERT Model = 0.8781218393573155
=====
EM Score on Validation Dataset using BERT Model = 0.13396648044692738
F1 Score on Validation Dataset using BERT Model = 0.4025203999249132
=====
EM Score on Train Dataset using BERT Model = 0.11096474234792716
F1 Score on Train Dataset using BERT Model = 0.40688574065643784
=====
```

## Results:

In this project, we explored different models for question answering on the Squad dataset, including BiLSTM, BIDAf, BERT Cased, and DistilBERT. We evaluated these models based on their EM and F1 scores on the train and validation datasets.

The BIDAf model achieved an F1 score of 0.668 on the validation dataset, which is the highest among the models we tested. The BiLSTM model performed the worst, with an F1 score of only 0.285. The DistilBERT model achieved an F1 score of 0.753, which is also relatively high. The BERT\_Cased model achieved an F1 score of 0.402 on validation dataset and 0.406 on train Dataset.

We then created an ensembling model that chose the answer with the best F1 score for each question answer pair. This model achieved an F1 score of 0.825 on the train dataset and 0.830 on the validation dataset, which outperformed all individual models when ensembled BiLSTM, BIDAf and BERTCased.

```
=====
Ensembled F1 Score on Train Dataset: 0.8157256536261616
Ensembled EM Score on Train Dataset: 0.5692873704632565
=====
```

```
Ensembled F1 Score on Dev Dataset: 0.8305768173069266
Ensembled EM Score on Dev Dataset: 0.616244939271255
=====
```

But when ensembled BiLSTM, BIDAf, BERTCased, DistilBERT, we got the score:

```
=====
Ensembled F1 Score on Train Dataset: 0.9381702352426067
Ensembled EM Score on Train Dataset: 0.797303396794167
=====
```

```
Ensembled F1 Score on Dev Dataset: 0.9157810094219027
Ensembled EM Score on Dev Dataset: 0.771813080399265
=====
```

## Discussion:

The results show that BIDAf, DistilBERT, and ensembling models are effective for question answering on the Squad dataset. BiLSTM, on the other hand, did not perform well in this task and BERT\_Cased model needed some more training to perform better.

It's worth noting that the DistilBERT model achieved a high F1 score with much faster inference time compared to the BERT Cased model, which may make it more practical for real-world applications where speed is an important consideration.

The ensembling approach proved to be effective in improving the performance of individual models, which suggests that combining multiple models can often lead to better results.

## Conclusion:

In conclusion, our study shows that choosing the right model is crucial for achieving high performance in question answering on the Squad dataset. BIDAf and DistilBERT are effective models for this task, and ensembling can further improve performance.

However, there is always room for improvement, and future work could explore more advanced models and techniques to further enhance the performance of question answering systems.

**Presentation:** <https://1drv.ms/p/s!Asco20fW-uO6lg4MI0IOon15qn5m>

## References:

1. BAS: An Answer Selection Method Using BERT Language Model :

<https://arxiv.org/pdf/1911.01528.pdf>

2. Learning to Generate Questions by Learning to Recover Answer-containing Sentences : <https://aclanthology.org/2021.findings-acl.132.pdf>
3. ENSEMBLE APPROACH FOR NATURAL LANGUAGE QUESTION ANSWERING PROBLEM : <https://arxiv.org/pdf/1908.09720.pdf>
4. <https://towardsdatascience.com/question-answering-with-a-fine-tuned-bert-bc4dafd45626>
5. <https://huggingface.co/>
6. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding : <https://arxiv.org/pdf/1810.04805.pdf>
7. CoQA: A Conversational Question Answering Challenge : <https://arxiv.org/pdf/1808.07042.pdf>
8. Question Answering on SQuAD with BERT : <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15792151.pdf>
9. U-Net: Machine Reading Comprehension with Unanswerable Questions : <https://arxiv.org/pdf/1810.06638.pdf>
10. A BERT Baseline for the Natural Questions: <https://arxiv.org/pdf/1901.08634.pdf>
11. BI-DIRECTIONAL ATTENTION FLOW FOR MACHINE COMPREHENSION : <https://arxiv.org/pdf/1611.01603.pdf>