International Journal of Modern Developments in Engineering and Science
Volume 2, Issue 7, July 2023
https://www.ijmdes.com | ISSN (Online): 2583-3138

# Interactive Question Answering Model using Natural Language Processing

Saara Anand[1*], Ram Kripalu Neelmani[2], Naman Manocha[3]
[1,2,3]*Bachelor of Technology, Department of Computer Science and Engineering, Vellore Institute of Technology, Amaravati, India*

***Abstract*: This paper aims to provide an empirical study and comparative analysis of the well-known Deep Learning Models on the Stanford Question Answering Dataset (SQuAD). Keeping in mind the enormous data in the current times, SQuAD Dataset serves as a benchmark for question answering tasks. It aims to solve the issues pertaining machine comprehension and huge context-based question answering tasks. This is a challenging problem in NLP as it requires the model to understand the context and use its reasoning abilities to accurately respond to questions. In this study, Natural Language Processing, Exploratory Data Analysis and Deep Learning Models like Bidirectional LSTMs (BiLSTM), BERT, DistilBERT, BiDAF, Ensemble Learning, Backpropagation neural networks and Optimization techniques have been incorporated for achieving the highest efficiency. Finally, a comparison of each model's performance based on evaluation metrics like accuracy, precision and F1-score has been done.**

***Keywords*: Natural Language Processing, BiLSTM, BERT, BiDAF, Ensemble Learning, SQuAD.**

## 1. Introduction

With the ever increasing data in the present times, it becomes highly important to use proper Deep Learning Models and Natural Language Processing (NLP) Systems. Interactive question answering is a challenging task that requires not only retrieving relevant information but also maintaining a coherent conversation with the user. This research paper presents a novel interactive QA model that utilizes NLP technologies to enable real-time, interactive, and contextually-aware responses to user queries. Natural Language Processing, commonly abbreviated as NLP, is a fascinating field of artificial intelligence that focuses on enabling computers to understand, interpret, and interact with human language in a way that is both meaningful and contextually relevant. Natural Language Processing's (NLP) Question Answering (QA) subfield focuses on automatically responding to queries presented in natural language. A QA system's objective is to comprehend the queries and offer a clear, pertinent response. Numerous real-world uses for QA systems exist, such as knowledge management and customer education and service. They may assist in minimising the time and effort needed to locate solutions. This model would utilize pretrained language models like BiLSTM, BERT, DistilBERT and BIDAF. LSTM stands for Long Short-Term Memory, and it is a type of recurrent neural network (RNN) architecture. LSTM networks are designed to overcome the shortcomings of traditional RNNs in capturing long-term dependencies in sequential data. The key idea behind LSTM is the introduction of memory cells, which are responsible for storing and accessing information over long periods of time. These memory cells have an internal structure that allows them to learn and forget information based on the input data and the network's objectives. BiLSTM stands for Bidirectional Long Short-Term Memory. It is a type of recurrent neural network (RNN) architecture that combines the forward and backward information flow to capture dependencies in sequential data more efficiently. In a traditional LSTM, information flows only in one direction, from past to future. However, in certain applications, understanding the context from both past and future perspectives is important. BiLSTM addresses this limitation by processing the input sequence in both forward and backward directions simultaneously. BERT stands for Bidirectional Encoder Representations from Transformers. It is a powerful natural language processing (NLP) model introduced by Google in 2018. BERT is based on the Transformer architecture, which is a type of deep learning model designed to handle sequential data efficiently. Unlike previous NLP models that predominantly relied on a unidirectional approach (e.g., LSTM), BERT employs a bidirectional approach, enabling it to consider both the left and right context of a word simultaneously. This bidirectional capability allows BERT to capture more nuanced relationships between words and improve the understanding of the overall context. DistilBERT is a small and quick model based on the BERT architecture. Knowledge distillation is performed during the pre-training phase to reduce the size of a BERT model by 40%. To leverage the inductive biases learned by larger models during pre-training, the authors introduce a triple loss combining language modeling, distillation and cosine-distance losses. BIDAF stands for Bi-Directional Attention Flow, and it is a deep learning model architecture designed for machine comprehension tasks, specifically for question-answering. The main objective of BIDAF is to accurately locate the relevant answer span within a given context paragraph when posed with a question. By incorporating the attention mechanism and bi-directional modeling, BIDAF enables the model to effectively understand the interplay between the

*Corresponding author: saaraanand1@gmail.com

question and context, and to accurately identify the answer span. BIDAF has demonstrated impressive performance on various question-answering benchmarks, such as the Stanford Question Answering Dataset (SQuAD), by achieving high accuracy in locating the correct answer spans within the provided context paragraphs. Our aim would be to use an ensemble approach by combining four different language models, namely BiLSTM, BIDAF, BERT and DistilBERT, to achieve the highest possible F1 score. Ensemble methods in machine learning refer to techniques that combine multiple models to improve predictive performance and generalization. The core idea behind ensembling is that by aggregating predictions from diverse models, the ensemble can often outperform any individual model.

## 2. Dataset

We have investigated a number of datasets for training and assessing QA models, including SQuAD, MS MARCO, BioASQ, TREC QA, Natural Questions (Google AI), NarrativeQA, and HotpotQA. Each dataset has unique properties, so selecting a dataset should rely on the project's specific requirements and the research question. The SQuAD dataset will be used in our study for training and evaluating as per our requirements of our QA model. It includes questions and responses based on experts from several Wikipedia articles, making it a useful tool for creating reading-capable quality-assurance models. The dataset consists of questions and context texts, and within the context paragraph, the model is anticipated to forecast the answer span. The dataset consists of over 100,000 questions and their corresponding answer spans, all of which are based on more than 500 Wikipedia articles. 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.

## 3. Proposed Methodology

We downloaded the Squad dataset and preprocessed the data to extract questions, contexts, and answers for each example. We then selected four different models for question answering, including BiLSTM, BIDAF, BERT and DistilBERT. We trained each model on the Squad dataset using the training split and evaluated their performance on the validation split. We used the EM and F1 scores to evaluate the performance of each model on both the training and validation datasets. To increase our efficiency and accuracy. We created an ensembling model that selected the answer with the highest F1 score among the four models for each question answer pair. Then we moved to the performance evaluation and evaluated the performance of the ensembling model on the train and validation datasets using the EM and F1 scores. We analyzed the results to identify the strengths and weaknesses of each model and the ensembling approach, followed by a discussion of the results and drew conclusions based on the findings, highlighting the strengths and limitations of the study and suggesting potential directions for future work.

The first model we have used is of BiLSTM. BiLSTM stands for Bidirectional Long Short-Term Memory. It is a type of recurrent neural network (RNN) architecture that combines the forward and backward information flow to capture dependencies in sequential data more efficiently. In a traditional LSTM, information flows only in one direction, from past to future. However, in certain applications, understanding the context from both past and future perspectives is important. GloVe word embeddings are used for BiLSTM. 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.

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.

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.

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.

We have then moved on to the BiDAF Model. The Bi-directional Attention Flow (BiDAF) network is a multi-stage hierarchical process that uses a bi-directional attention flow mechanism to accomplish query-aware context representation without early summarization. It expresses context at various levels of granularity. The advantage of using a bi-directional attention flow to create query-aware context representations is that it allows attention at every time step and representations from lower layers to pass through to the modeling layer above. The Bi-directional Attention Flow (BiDAF) network is a multi-stage hierarchical process that uses a bi-directional attention flow mechanism to accomplish query-aware context representation without early summarization. It expresses context at various levels of granularity.

The advantage of using a bi-directional attention flow to create query-aware context representations is that it allows attention at every time step and representations from lower layers to pass through to the modeling layer above.

The most important model we have implemented is that of BERT. Bidirectional Encoder Representations from Transformers, or BERT, is a deep learning model that is based on Transformers. In Transformers, each output element is connected to each input element, and the weightings between them are dynamically determined based upon their connection. This procedure is known as attention in NLP. In the past, language models could only interpret text input sequentially -- either from right to left or from left to right -- but not simultaneously. BERT is unique since it can simultaneously read in both directions. Bidirectionality is the name for this capacity, which the invention of Transformers made possible. BERT is pre-trained on two distinct but related NLP tasks—Masked Language Modeling and Next Sentence Prediction—using this bidirectional capacity. There are various sizes and variations of the BERT (Bidirectional Encoder Representations from Transformers) model, including BERT. 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 resources.

The BERT model is distilled into the DistilBERT model. During the pre-training phase, knowledge distillation was used to shrink a BERT model by 40% while maintaining 97% of its language understanding capabilities and increasing its speed by 60%. Combining language modeling, distillation, and cosine-distance losses creates a triple loss that makes use of the inductive biases that larger models picked up during pre-training. DistilBERT is a lightweight, quick, and small model that is simple to utilize for on-device applications and costs less to pre-train.

The path to the JSON file is passed as an argument to the load_json(path) function, which returns the dataset's JSON object. First, it uses the built-in Python functions open() and json.load() to open and read the file, respectively. For debugging purposes, it then prints the first data item's keys and length before returning the data. The dataset's JSON object is passed into the parse_data(data) function, which outputs a collection of dictionaries. The parsed data is first initialized into an empty list after the data list has been extracted from the JSON object. The context and QA values are then extracted from each item in turn by looping through the data list. The id, question, answers, answer_start, and answer_end values for each qas item are extracted, and these values are then used as keys in a dictionary that is created. The dictionary is then added to the list of the parsed data. A collection of dictionaries containing the context, question, and label (start and finish indices) for each answer in the Squad dataset are the result of the parse_data(data) function.

We used the load_json(path) function previously defined to load the train and validation datasets in JSON format. The data is then parsed into a list of dictionaries including the context, question, and label information by calling the parse_data(data) function for each dataset. In order to confirm that the data has been successfully parsed, it then prints the length of the two lists. It then uses pd.DataFrame() to turn the two lists of dictionaries into pandas DataFrames. Finally, it renames the columns in the DataFrames and uses the apply() function to retrieve the first element from the Answer Start column. This is due to the fact that, although the response Start column lists both the start and end indices of the response, we are only interested in the start index.

The Lookahead optimizer improves the training process by "looking ahead" to prevent overshooting in weight updates. The concept is described in the paper "Lookahead Optimizer: k steps forward, 1 step back"

Training Loop: The code sets up a training loop with a specified number of epochs (num_epochs). It iterates through the data in the train_dataloader, which is created using the SquadDataset class. In each iteration, it performs the forward pass, computes the loss, and applies the backward pass to update the model's parameters.

Lookahead Optimizer: The code initializes an AdamW optimizer with a learning rate of 5e-5 and then wraps it with the Lookahead optimizer. The la_steps parameter represents the number of lookahead steps, which is set to 5. The la_alpha parameter controls the linear interpolation factor, which is set to 0.8, meaning the Lookahead optimization interpolates 80% towards the inner optimizer. The pullback_momentum parameter is set to "none," which means the momentum is not pulled back during the lookahead update.

GPU/CPU Handling: The model is moved to the appropriate device using the **device** variable, which is determined based on the availability of a CUDA-compatible GPU.

Fine-Tuning: The training loop runs for the specified number of epochs, and at the end of each epoch, it prints the progress. After training, you can save the fine-tuned model using the save_pretrained method.

Additionally, hyperparameters such as batch size and lookahead parameters may need to be tuned based on the specifics of your dataset and problem.

Fine-Tuning Loop: The code continues with the fine-tuning process using the Lookahead-optimized AdamW optimizer. The model is set to train mode (model.train()) and iterates through the training data in batches. For each batch, the forward pass is performed to compute the loss, and the backward pass updates the gradients. The Lookahead optimizer is used to update the model's parameters with lookahead steps.

Saving Fine-Tuned Model and Tokenizer: After the fine-tuning loop is completed, the code creates a directory named MODEL_DIR (if it doesn't exist) and saves both the tokenizer and the fine-tuned model to that directory using the save_pretrained method.

Loading Fine-Tuned Model and Tokenizer: Finally, the code loads the tokenizer and fine-tuned model from the saved directory using the AutoTokenizer and AutoModelForQuestionAnswering classes, respectively. This allows you to use the fine-tuned model for inference on new data or for further evaluation.

Thus, we have defined the BERT tokenizer and model after importing the required libraries, followed by 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, then we calculated the total number of training steps and specify the number of training epochs. We then utilised the SQuAD 2.0 preprocessed dataset to train the BERT model and then save the weights of the trained model to a file. We have created a formula for anticipating responses to queries based on context followed by reading test data for SQuAD 2.0 from a JSON file and creating a JSON file with the anticipated responses.

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 tokensizer 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 this 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 simpletransformers which has QuestionAnsweringModel, QuestionAnsweringArgs modules in it. QuestionAnswerting Model can load pretrained BERT 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.

The main idea we have followed here is to use the Ensemble approach to achieve a higher F1 Score.

Ensemble learning is a general meta-approach to machine learning that seeks better predictive performance by combining the predictions from multiple models.

In the first step of creating ensemble architecture three base models have been trained on the full training set and evaluated against the evaluation data. The architectures have been implemented in PyTorch.

Presented ensemble model is based on a class-specific weighted voting. Algorithm 1 presents, how the class-specific weights are obtained. It is an average F1 metric obtained after evaluation the model on all questions from specific class. Algorithm 2 describes, how does the voting mechanism work. It gathers all candidate answers from all models and if at least two models give duplicate answer their weights are added. Finally, the candidate answer is this one with highest weight or returned by globally best model (in specific class) when no duplicated answer exists.

Our main goal is to check the working of ensemble models can improve best machine comprehension architectures. At the first stage one of the best four models were chosen for studying their accuracy using different techniques. These are DistilBERT, Bidirectional Attention Flow(BiDAF) , Bidirectional LSTM (BiLSTM) and Bidirectional Encoder Representations from Transformers(BERT) . All the models use deep learning combined with different types of attention mechanisms. Error analysis shows that each model obtains better results on different type of questions. Therefore, the models could be combined together in order to produce a better outcome for all questions of any kind. One of the most obvious approach is to build ensemble model. The main goal is to avoid weaknesses and use all strengths of analysed architectures. The idea of building ensemble models is to combine predictions from different, well performing and separately trained models and calculate the actual prediction as the average or weighted predictions. In presented work, an answer comparison mechanism has been defined and implemented, to obtain a final answer based on separated answers given by chosen models. Before building the ensemble model comparative studies were performed between models, with particular reference to their attention layers and analysis of the results gained by models, including error analysis. The SQuAD dataset was used to train and evaluate the models. The proposed ensemble mechanism brings an improvement in predictions accuracy.

## 4. Results

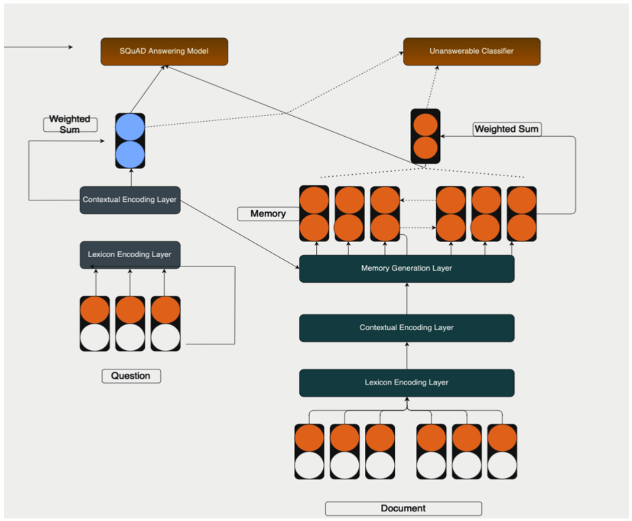


Fig. 1. Architectural diagram of BiLSTM

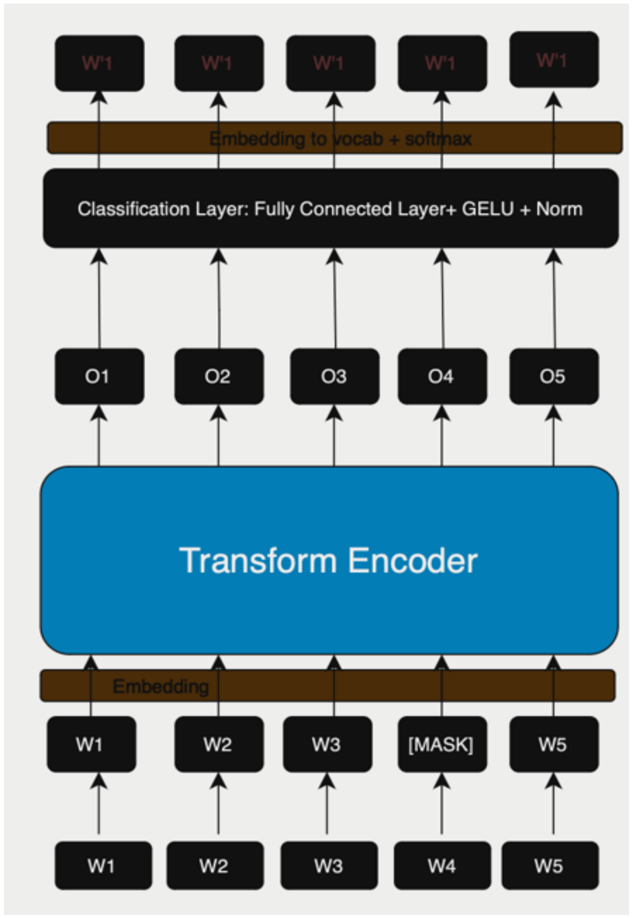

W'1
W'1
W'1
W'1
W'1
Embedding to vocab + softmax
Classification Layer: Fully Connected Layer+ GELU + Norm
O1
O2
O3
O4
O5
Transform Encoder
Embedding
W1
W2
W3
[MASK]
W5
W1
W2
W3
W4
W5

Fig. 2. Architectural diagram of BERT

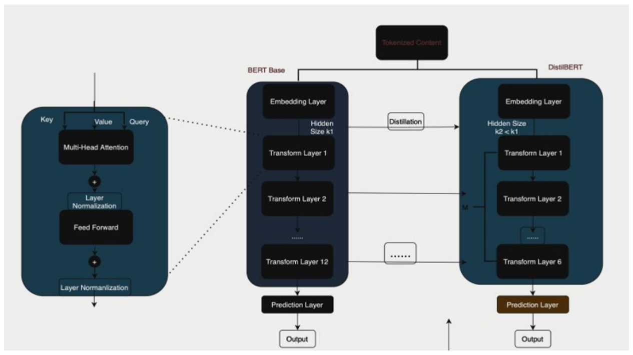


Fig. 3. Architectural diagram of DistilBERT

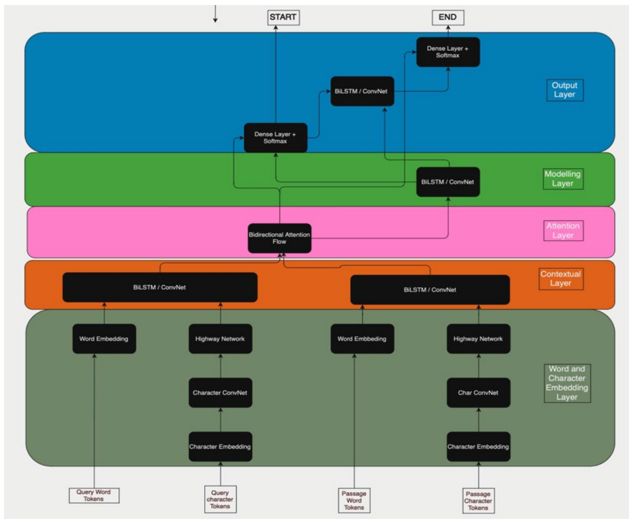


Fig. 4. Architectural diagram of BIDAF

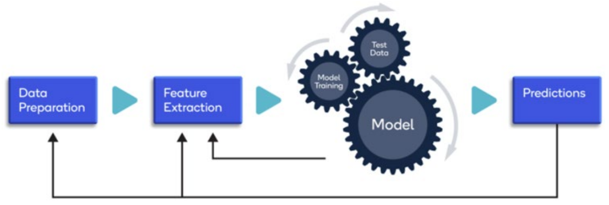


Fig. 5. Basic working of the models

Table 1
EM and F1 scores of each model

| MODEL | Dataset | EM Score | F1 score |
|---|---|---|---|
| Bi LSTM | Dev | 0.0003725182137 | 0.2587250052116 |
| Bi LSTM | Train | 0.0004989127943 | 0.2394558902758 |
| Bi DAF | Dev | 0.5398709732878 | 0.6649952873842 |
| Bi DAF | Train | 0.5049570942344 | 0.6688789641243 |
| BERT | Dev | 0.1347834324908 | 0.4334487891223 |
| BERT | Train | 0.1148590349743 | 0.4149308943890 |
| Distil BERT | Dev | 0.5644257924784 | 0.7549088089954 |
| Distil BERT | Train | 0.7198289083405 | 0.8709847094735 |

Table 2
Ensemble using BiLSTM, BERT and BIDAF

| MODEL | Dataset | EM Score | F1 score |
|---|---|---|---|
| Ensemble | Train | 0.5589492384289 | 0.803908844944 |
| | Dev | 0.6128290383983 | 0.833848949894 |

Table 3
Ensemble using BiLSTM, BERT, BIDAF and DistilBERT

| MODEL | Dataset | EM Score | F1 score |
|---|---|---|---|
| Ensemble | Train | 0.7834273423834 | 0.912238984939 |
| | Dev | 0.7734647272782 | 0.898234239483 |

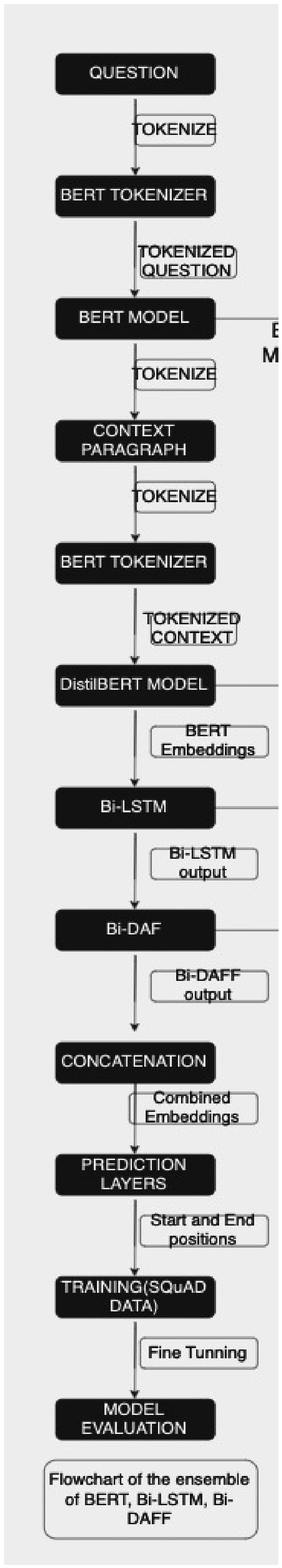
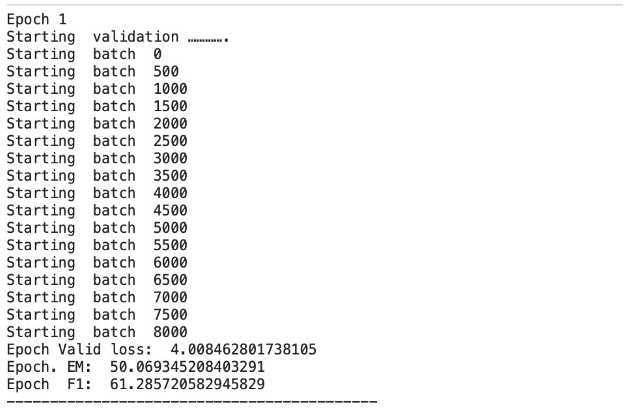
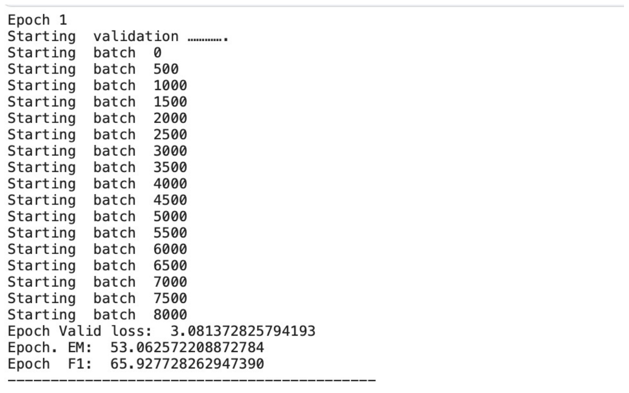


Fig. 6. Architecture diagram of ensemble model



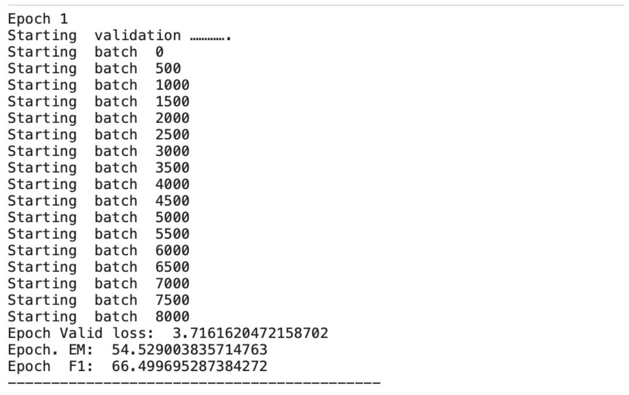
```
Epoch 1
Starting  validation ………….
Starting  batch  0
Starting  batch  500
Starting  batch  1000
Starting  batch  1500
Starting  batch  2000
Starting  batch  2500
Starting  batch  3000
Starting  batch  3500
Starting  batch  4000
Starting  batch  4500
Starting  batch  5000
Starting  batch  5500
Starting  batch  6000
Starting  batch  6500
Starting  batch  7000
Starting  batch  7500
Starting  batch  8000
Epoch Valid loss:  4.008462801738105
Epoch. EM:  50.069345208403291
Epoch  F1:  61.285720582945829
-------------------------------------------
```

Fig. 6. Finetuning BERT using Adadelta



```
Epoch 1
Starting  validation ………….
Starting  batch  0
Starting  batch  500
Starting  batch  1000
Starting  batch  1500
Starting  batch  2000
Starting  batch  2500
Starting  batch  3000
Starting  batch  3500
Starting  batch  4000
Starting  batch  4500
Starting  batch  5000
Starting  batch  5500
Starting  batch  6000
Starting  batch  6500
Starting  batch  7000
Starting  batch  7500
Starting  batch  8000
Epoch Valid loss:  3.081372825794193
Epoch. EM:  53.062572208872784
Epoch  F1:  65.927728262947390
-------------------------------------------
```

Fig. 7. Finetuning BERT using Ada halved



```
Epoch 1
Starting  validation ………….
Starting  batch  0
Starting  batch  500
Starting  batch  1000
Starting  batch  1500
Starting  batch  2000
Starting  batch  2500
Starting  batch  3000
Starting  batch  3500
Starting  batch  4000
Starting  batch  4500
Starting  batch  5000
Starting  batch  5500
Starting  batch  6000
Starting  batch  6500
Starting  batch  7000
Starting  batch  7500
Starting  batch  8000
Epoch Valid loss:  3.7161620472158702
Epoch. EM:  54.5290038357l4763
Epoch  F1:  66.499695287384272
-------------------------------------------
```

Fig. 8. Finetuning BERT using Ada halved + Optim Adam

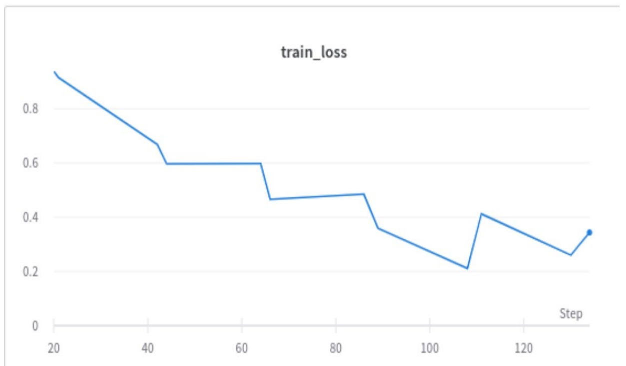


Fig. 9. Loss graph for BERT

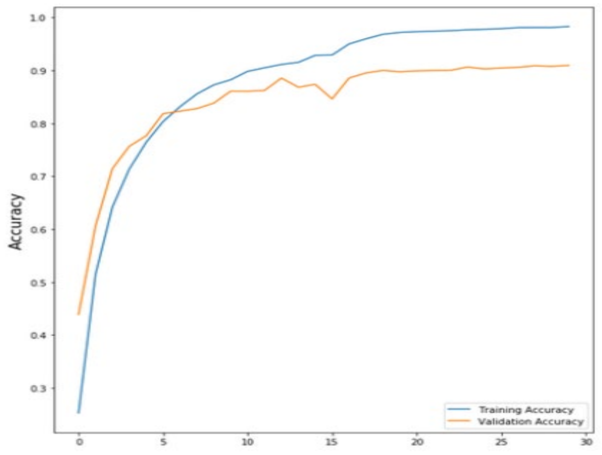
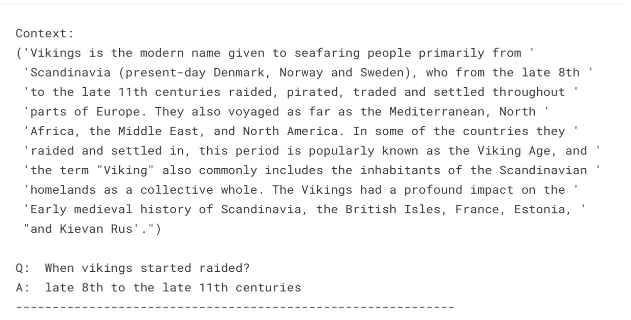


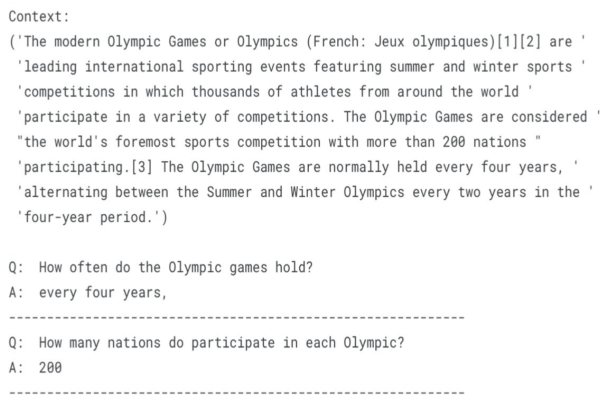
Fig. 10. Accuracy graph of our ensembled model



```
Context:
('Vikings is the modern name given to seafaring people primarily from '
 'Scandinavia (present-day Denmark, Norway and Sweden), who from the late 8th '
 'to the late 11th centuries raided, pirated, traded and settled throughout '
 'parts of Europe. They also voyaged as far as the Mediterranean, North '
 'Africa, the Middle East, and North America. In some of the countries they '
 'raided and settled in, this period is popularly known as the Viking Age, and '
 'the term "Viking" also commonly includes the inhabitants of the Scandinavian '
 'homelands as a collective whole. The Vikings had a profound impact on the '
 'Early medieval history of Scandinavia, the British Isles, France, Estonia, '
 "and Kievan Rus'.")

Q:  When vikings started raided?
A:  late 8th to the late 11th centuries
------------------------------------------------------------
```

Fig. 11. Application I of our final model



```
Context:
('The modern Olympic Games or Olympics (French: Jeux olympiques)[1][2] are '
 'leading international sporting events featuring summer and winter sports '
 'competitions in which thousands of athletes from around the world '
 'participate in a variety of competitions. The Olympic Games are considered '
 "the world's foremost sports competition with more than 200 nations "
 'participating.[3] The Olympic Games are normally held every four years, '
 'alternating between the Summer and Winter Olympics every two years in the '
 'four-year period.')

Q:  How often do the Olympic games hold?
A:  every four years,
------------------------------------------------------------
Q:  How many nations do participate in each Olympic?
A:  200
------------------------------------------------------------
```

Fig. 12. Application II of our final model

## 5. Conclusion and Future Work

In this project, we explored different models for question answering on the Squad dataset, including BiLSTM, BIDAF, BERT, 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 0.664 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 0.239. The DistilBERT model achieved an F1 score of 0.754, which is also relatively high. The BERT model achieved an F1 score of 0.433 on validation dataset and 0.414 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 0.912 on the train dataset and 0.898 on the validation dataset, which outperformed all individual models when ensembled BiLSTM, BIDAF and BERT.

Our Question Answering Model was successful and gave accurate results. The results demonstrate that the Squad dataset may be successfully used to answer questions using the BIDAF, DistilBERT, and ensembling models. On the other hand, BERT model needed more training to perform better and BiLSTM struggled in this task. It's crucial to note that the DistilBERT model outperformed the BERT model in terms of F1 score and inference time, making it potentially more useful for real-world applications where speed is a key factor. The ensembling method was successful in enhancing the performance of individual models, indicating that merging different models can frequently produce superior outcomes. Thus, choosing the appropriate model is essential for getting excellent performance in question-answering on the Squad dataset, according to the findings of our study. For this work, BIDAF and DistilBERT are useful models, and ensembling can boost performance even more.

Though there's always room for improvement, future research can examine more sophisticated models and methods to increase the performance of question-answering systems.

Given the rapidly evolving nature of technology, there are several potential future directions and areas of work that researchers and developers may explore to further improve these technologies, like to develop more efficient and faster models without compromising on performance. This could involve exploring techniques like model compression, quantization, and knowledge distillation. Focusing on deploying QA models in real-world applications such as customer support, virtual assistants, or search engines, where they can provide immediate value to users. We could also plan to combine text-based QA with other modalities like images, audio, or video, enabling models to answer questions that involve multi-modal inputs, for increased efficiency.

## References

[1] Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD.

[2] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.

[3] Yatskar, M. (2018). A qualitative comparison of CoQA, SQuAD 2.0 and QuAC.

[4] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.

[5] Alzubi, J. A., Jain, R., Singh, A., Parwekar, P., & Gupta, M. (2021). COBERT: COVID-19 question answering system using BERT. Arabian journal for science and engineering, 1-11.

[6] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[7] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.
Min, S., Seo, M., & Hajishirzi, H. (2017). Question answering through transfer learning from large fine-grained supervision data.

[8] Jia, R., & Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems.

[9] Nsaka, P., Dong, J., & Lee, A. Optimizing Match-LSTM for SQuAD v2. 0.

[10] Huang, Z., Xu, P., Liang, D., Mishra, A., & Xiang, B. (2020). TRANS-BLSTM: Transformer with bidirectional LSTM for language understanding.

[11] Wang, W., Yang, N., Wei, F., Chang, B., & Zhou, M. (2017, July). Gated self-matching networks for reading comprehension and question answering. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, Volume 1: Long Papers, pp. 189-198.