Arabic QA DistilBERT Fine-Tuning

Mohamed Abdelgaber, Ali Helmy, Mahmoud Nabil — Team 3

May 26, 2023

1 Introduction

Natural Language Processing (NLP) is a rapidly growing field of artificial intelligence that focuses on enabling computers to understand and analyze human language. One important task in NLP is Question Answering (QA), which involves answering natural language questions posed by users based on a given context. In this project, we aim to build an AI model that can perform QA in Arabic language using the fine-tuned DistilBert model.

Arabic is a complex language with its own unique characteristics, which makes it challenging to build accurate NLP models for it[GSA+21]. Nevertheless, there is a growing demand for Arabic NLP applications in various fields, such as education, healthcare, and finance. Thus, our project can contribute to the development of Arabic NLP by providing a QA model that can accurately and efficiently answer questions in Arabic language.

To achieve our goal, we will use the Arabic Questions and Answers Dataset (AQAD)[AMS+20] after preprocessing it. We make use of the Arabert[ABH] preprocessor and tokenizer in the preprocessing stage to tokenize the text and prepare it for fine-tuning the DistilBert model.

The rest of this report will describe the data analysis we conducted, the data collection and preprocessing steps, and the system architecture of our QA model. Additionally, we will discuss the challenges we faced during the project and our initial plans for addressing them.

2 Data Analysis

The dataset used in this project is the Arabic Questions and Answers Dataset (AQAD) that contains the questions, answers, and the context from which the answer is pulled. The data consitis of 17911 questions and follows the structure of the Stanford Question Answering Dataset (SQuAD)[RZLL16].

2.1 Data Frame

The dataset is put into a dataframe with 4 columns and N rows:

1. title,question,answer and is-impossible columns.
2. Each row represents an entry (QA).

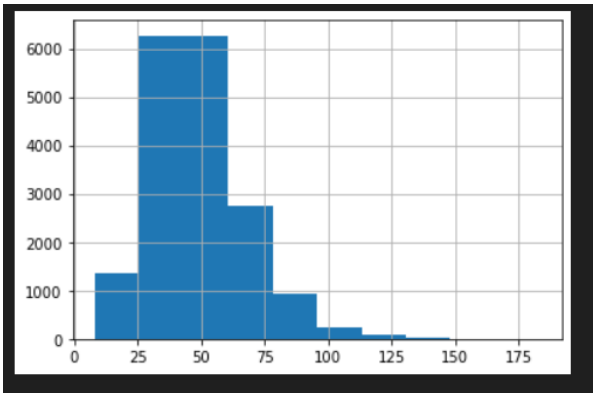
2.2 Analysis

1. We started analysing the dataframe by getting the count of the topics in the table "title" , counting the words that have diacritics in questions and answers, counting the English words, and presenting some of these words. We Found that words that have diacritics represent small numbers and that there are few english words . We used the popular english words to detect the english words present in the dataset so some non-frequently used english words may have not been detected.

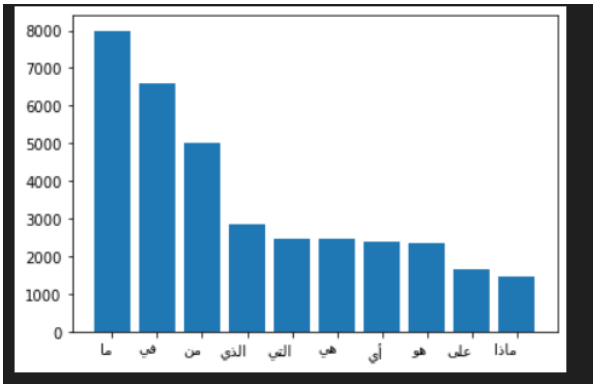
```
The number of Arabic words with تشكيل in the answers is: 389
['قيل']
['بديلاً', 'مؤثرًا']
['وتستخدم']
['وتستخدم']
['أيضًا']
['سنويًا']
['سُلطانا', 'حياةً']
['سُلطانا', 'حياةً']
['تشق', 'ك', 'ل']
['نفس']
['نفس']
['الجهل']
['عمل', 'خي', 'نَت', 'ع', 'ثمار', 'جزائه']
['شهرياً']
['ش', '']
```

2. To proceed with the analysis, the answers that have no values are removed, and we calculated the average question and answer length. We found that answers are more likely to be shorter than questions.

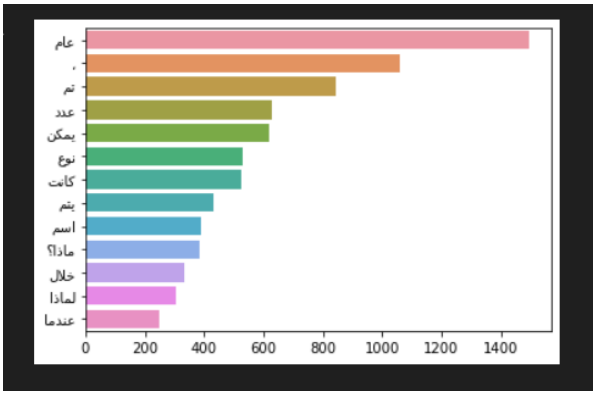
3. Then we showed the number of characters, words, and average word length present in each question and answer using histograms.



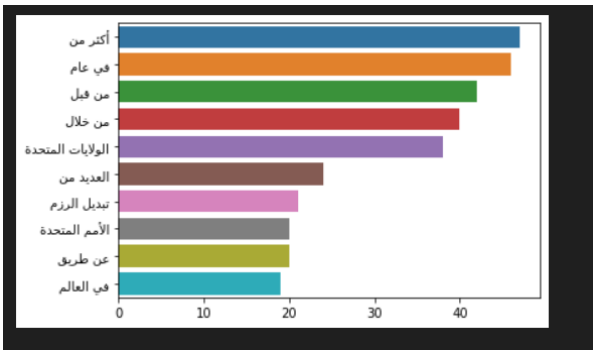
4. Then we used a function to get Arabic stopwords to check their frequencies in the questions and answers, and as expected, the frequencies are very high and words like "في" and "ما" has the highest frequencies.



5. Then we checked what words other than the stop words are the most frequent in questions and answers. "عام" is the most frequent word which indicate that most asked questions in dataset about the year.



6. Finally, we used a function to get the top n-grams in QAs, and we checked for bi-grams and tri-grams in QAs. Most Frequent Bigram in questions are "ما هو" and "ما هي".



3 Data Preprocessing

In this section, we will discuss the data preprocessing steps that we have taken to prepare our dataset for training our Question Answering model.

After reading and parsing the dataset, it is split into train-test splits with 80:20 ratio. The conducted analysis showed that our dataset has a good variety of questions, contexts and answers to ensure that our model is trained to handle a variety of inputs.

To prepare our dataset for training, we have to preprocess it. The following are the steps that we followed in our preprocessing stage.

3.1 Adding Answer End Index

We first processed the answers list of dictionaries to add the key 'answer_end' key with its value to every answer in our dataset. The 'answer_end' indicates the index of the final character of the answer inside the context for a specific question. This step is important because we will use this index to identify the end of the answer during training. The 'answer_end' value is calculated using the 'answer_start' value in addition to the answer length. We'd then compare the answer given in the original dataset with the answer produced from the substring of the context with the 'answer_start' and 'answer_end' indices. In case the answers don't match, the 'answer_end' value is adjusted to meet this criteria.

3.2 Removing Diacritics and Tatweel

Initial experiments with the Arabert tokenizer showed that removing the diacritics and Tatweel reduced the number of '[UNK]' tokens produced from the tokenizer. This is a recommended procedure for Arabert since the same processing is done on the data used to train the tokenizer.

3.3 Using Farasa segmentation

we used Farasa[ADDM16] to segment the Arabic text as recommended for the Arabert tokenizer. Farasa, is a fast and accurate Arabic segmenter that facilitates capturing the details of the morphologically rich Arabic language. Farasa segments the words based on a large number of features and is used in the Arabert tokenizer training.

3.4 Tokenization

We used the pre-trained Arabert Arabic tokenizer to tokenize the Arabic text. This tokenizer is of word-piece type. Word-piece tokenizers are a type of subword tokenization technique commonly used in natural language processing (NLP) tasks. They aim to break down words into smaller subword units called "word-pieces" based on linguistic patterns and frequency statistics observed in a given corpus. The word-piece arabert tokenizer often captured more characters per token proving more efficient compared to character-based tokenizers. This is especially useful when feeding data to the Bert-based models like Distilbert that has a small embedding size of 512 tokens. The 'Padding' and 'Truncation' flags are set to 'True' to make sure the input tokens match the embedding size of the tokenizer.

3.5 Concatenation

After tokenization, we concatenated each context and question to form a single input sequence. This is because our model expects each context and question pair together to produce the answer.

3.6 Adding Answer Start and End Indices to the tokenization

After that, we added the answer start and end indices to the encoding after converting from character to token. This step is crucial because it allows our model to learn where the answer is located within the context without concatenating the answer to the context and question.

4 Methodology

This section will focus on the model architecture, discussing the general structure of the model and the layers used.Furthermore, the Fine-Tuning process of the model will be the focus of the second subsection discussing the hyperparameters used for fine-tuning the DistilBert model and the training process, providing an overview of the key settings that define how the model is trained and updated. What makes DistilBert unique is that during the pre-training phase, knowledge distillation was applied to reduce the size of a BERT model by 40%. This compression was achieved by transferring knowledge from a larger model to a smaller one. Remarkably, despite the size reduction, the distilled model retained 97% of its language understanding abilities. Additionally, the distilled model exhibited a significant improvement in speed, being approximately 60% faster compared to the original larger model. Thus, knowledge distillation successfully achieved a substantial reduction in model size while preserving a high level of language understanding and significantly enhancing computational efficiency.

4.1 System Architecture

The system architecture for the project is based on the fine-tuning of the DistilBert model for the task of Question Answering in the Arabic language. DistilBert is a smaller and faster version of BERT, which makes it easier to train and more efficient to use for QA tasks.

The input to the model is a concatenated string of the context and question, which is then tokenized using the Arabert tokenizer. The tokenization process converts the input text into a series of tokens, which are then mapped to their corresponding indices in the model's embedding layer.

After tokenization, the answer start and end indices are added to the tokenized encoding. This is done to inform the model about the boundaries of the answer in the context.

The model architecture consists of multiple layers, including embedding, attention, and feed-forward layers. The embedding layer converts the tokenized input into a dense vector representation. The

attention layer helps the model to focus on the relevant parts of the input during training and inference. The feed-forward layer maps the output of the attention layer to the final predicted answer.

During training, the model is optimized using the Adam optimizer, which adjusts the learning rate based on the gradient of the loss function. The loss function used in this project is the mean squared error (MSE), which measures the difference between the predicted answer and the true answer, this will be further discussed in the fine-tuning subsection.

Once the model is trained, it is used for inference by feeding it with a new context and question. The model outputs the predicted answer, which can be compared with the true answer to evaluate the performance of the model.

Overall, the system architecture for this project is designed to enable the fine-tuning of the DistilBert model for the task of Question Answering in Arabic language. The architecture includes several key components, including tokenization, encoding, attention, and optimization, which work together to enable the model to learn how to answer questions based on the given context.

4.2 DistilBert Fine-Tuning Parameters

The question answer model used was a pretrained base multilingual cased variant of DistilBERT, which is trained on multiple languages and preserves the case information in the text.DistilBERT model for question answering was provided by the Hugging Face library. After importing and setting up the model, a random seed is set to the value of 42 to insure reproducibility. This ensures that the random number generation throughout the code will produce the same results each time it is run. The random seed is set for both the PyTorch, Python's built-in random module, and NumPy libraries. Next, the code checks if a GPU is available and assigns the device as 'cuda' if it is, or 'cpu' otherwise. This allows the code to run on a GPU if one is available, which can significantly speed up the training process. The *DistilBertForQuestionAnswering* model is moved to the selected device (GPU or CPU) in order to perform computations on that device during training. The model is set to training mode because certain layers, such as dropout and batch normalization, behave differently during training compared to evaluation. The AdamW optimizer is initialized with the model's parameters and a learning rate of $5e-5$ (0.00005). AdamW is an optimization algorithm that combines the benefits of adaptive gradient descent with weight decay, which helps prevent overfitting. The batch size is set to 16 for both the training and validation data loaders. The batch size determines the number of samples that are processed together in parallel during each iteration of training or evaluation. A larger batch size can lead to faster computation, but may require more memory.

4.3 Training

The code runs for a total of 6 epochs, meaning that the entire training dataset will be passed through the model 6 times. Each epoch consists of iterating over the training data in batches and updating the model's parameters. During training, a loss function is utilized to quantify the discrepancy between the predicted start and end positions and the ground truth positions in the training data. The loss is calculated using the outputs of the model, which include the predicted start and end logits (raw scores). The gradients of the loss with respect to the model's parameters are then computed using backpropagation. The AdamW optimizer updates the model's parameters based on these gradients. The optimizer adjusts the weights of the model's parameters to minimize the loss function, using techniques such as gradient descent with adaptive learning rates and weight decay, as mentioned before. The total loss (*loss*) for each batch is accumulated and the number of batches is counted to compute the average loss (*avg_loss*) per batch, which is total loss per batch divided by number of batches. This information is displayed and updated in the progress bar during training. After each epoch, the model is switched to evaluation mode using *model.eval()*. The validation data is then iterated over in batches, and the accuracy for both the start and end positions is calculated and stored. The average accuracy is displayed using a progress bar to assess the model's performance on the validation dataset.

5 Results

We conducted a series of experiments using different tokenizers to analyze their impact on the performance of our model. The three tokenizers we evaluated were:

1. Arabert with Farasa Segmentation: This tokenizer utilized the Farasa segmentation technique for tokenizing the text.

2. Arabert without Farasa Segmentation: This tokenizer did not incorporate the Farasa segmentation technique.

3. DistilBERT Tokenizer: This tokenizer employed the DistilBERT multilingual tokenizer.

5.1 Results Analysis

Upon analyzing the results, we observed that the use of Farasa segmentation had minimal effect on the overall performance. However, the DistilBERT multilingual tokenizer outperformed Arabert in several key metrics. It exhibited a lower average loss on the training data, higher single-index exact match accuracy on the validation dataset, and superior F1 score and double-index exact match metrics as well.

Additionally, we formulated two hypotheses to further investigate the model's performance. Firstly, we hypothesized that shorter contexts would yield better results. To test this hypothesis, we divided the dataset into three groups: [start: 400], [500: 700], and [900: end]. The metrics we developed were then evaluated on each of these groups. As anticipated, our hypothesis was validated, indicating that the model indeed performs better with shorter contexts.

Secondly, we proposed that the representation of a topic in the training dataset would positively correlate with its performance on the validation dataset. In other words, topics that were more extensively represented in the training data should demonstrate better results. To measure this, we introduced the metric *average_pred_over_context*, which denotes the percentage of the context utilized by the model in generating the answer. If the model consistently provided the full context as the answer, the *average_pred_over_context* would be 100%. Our findings supported this hypothesis, suggesting that topics with higher representation in the training dataset performed better on the validation dataset.

5.2 Results Summary

In summary, our experiments comparing different tokenizers revealed that the DistilBERT multilingual tokenizer outperformed Arabert in terms of average loss, single-index exact match accuracy, F1 score, and double-index exact match metrics. Furthermore, our hypotheses regarding the impact of context length and topic representation were substantiated, showcasing the importance of these factors in achieving optimal model performance.

Table 1: The table presents metrics for different models evaluated on datasets with varying context lengths. The "Model Name" column indicates the name of the model being evaluated, while the "Metric" column specifies the specific metric measured. The "Short Length Context" column displays metrics for datasets with shorter context lengths ([start: 400]), the "Medium Length Context" column shows metrics for datasets with medium context lengths ([500: 700]), and the "Long Length Context" column presents metrics for datasets with longer context lengths ([900: end]). The "All Lengths Context" column summarizes the overall metrics across all context lengths. This table allows for a comparison of model performance across different context lengths and provides insights into how models perform under varying contextual conditions. SLC= short length context, MLC=medium length context, LLC=long length context, ALC=all lengths context.

Model Name	Metric	SLC	MLC	LLC	ALC
Arabert	average double indexes exact match	0.113	0.0667	0.0598	0.081
Distilbert	average double indexes exact match	0.18	0.119	0.101	0.137
Arabert	average_F1	0.216	0.147	0.106	0.155
Distilbert	average_F1	0.307	0.22	0.161	0.236
Arabert	average_pred_over_context	0.145	0.0866	0.0605	0.0959
Distilbert	average_pred_over_context	0.114	0.0692	0.0546	0.0809

Table 2: Table 2 provides a comparison of two models, Arabert and Distilbert, based on different metrics at various epochs. The table consists of five columns: "Model Name," "Metric," and epoch-specific columns (epoch 0 to epoch 5). The "Model Name" column specifies the names of the models being evaluated, namely Arabert and Distilbert. The "Metric" column indicates the performance metric under consideration, which includes "Avg. loss" (average loss) and "Avg. single index exact match accuracy" (average single-index exact match accuracy). The subsequent columns represent the values of these metrics at each epoch. For instance, Arabert's average loss ranges from 4.47 at epoch 0 to 1.94 at epoch 5, while Distilbert's average loss ranges from 4 at epoch 0 to 1.42 at epoch 5. Similarly, Arabert's average single index exact match accuracy varies from 0.0863 at epoch 0 to 0.221 at epoch 5, whereas Distilbert's accuracy ranges from 0.155 at epoch 0 to 0.299 at epoch 5. This table allows for a straightforward comparison of the two models across different epochs and metrics.

Model Name	Metric	epoch 0	epoch 1	epoch 2	epoch 3	epoch 4	epoch 5
Arabert	Avg. loss	4.47	3.87	3.31	2.77	2.32	1.94
Distilbert	Avg. loss	4	3.34	2.76	2.22	1.78	1.42
Arabert	Avg. singe index exact match accuracy	0.0863	0.148	0.18	0.193	0.207	0.221
Distilbert	Avg. singe index exact match accuracy	0.155	0.224	0.253	0.271	0.296	0.299

Table 3: Table 3 displays key details regarding different context lengths. The table consists of four columns: "Context Length," "Interval," "Number of Samples," and "Average Length in Characters." The "Context Length" column describes the three categories of context length: "short," "medium," and "long." Each category is defined by an interval, with "[start : 400]" representing the "short" context, "[500 : 700]" representing the "medium" context, and "[900: end]" representing the "long" context. The "Number of Samples" column indicates the total number of instances available for each context length category. Additionally, the "Average Length in Characters" column specifies the average length, measured in characters, for the respective context lengths. The "short" context has an average length of 284 characters, the "medium" context has an average length of 499 characters, and the "long" context has an average length of 1280 characters.

context length	interval	number of samples	average length in characters
short	[start : 400]	974	284
medium	[500 : 700]	1079	499
long	[900: end]	602	1280

6 Conclusion

In conclusion, this project aimed to produce an AI model that can perform the task of Question Answering in the Arabic language. The model is trained on a dataset that contained questions, their answers, and the context from which the answers are derived. The dataset is split into training and testing sets. To preprocess the data, diacritics and tatweel are removed from the Arabic sentences, the text is segmented with farasa, and a pre-trained tokenizer called Arabert tokenizer is used to tokenize the text. The tokenizer used is of the word-piece type. After tokenizing, the contexts and questions are concatenated, and the answer start and end indices are added to the encoding after converting from characters to tokens. The model is fine-tuned on DistilBert, a pre-trained transformer-based model for natural language processing.

References

[ABH] Wissam Antoun, Fady Baly, and Hazem Hajj. Arabert: Transformer-based model for arabic language understanding. In *LREC 2020 Workshop Language Resources and Evaluation Conference 11–16 May 2020*, page 9.

[ADDM16] Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. Farasa: A fast and furious segmenter for Arabic. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 11–16, San Diego, California, June 2016. Association for Computational Linguistics.

[AMS+20] Atef Adel, Bassam Mattar, Sandra Sherif, Eman Elrefai, and Marwan Torki. Aqad: 17,000+ arabic questions for machine comprehension of text. 11 2020.

[GSA+21] Imane Guellil, Houda Saâdane, Faical Azouaou, Billel Gueni, and Damien Nouvel. Arabic natural language processing: An overview. *Journal of King Saud University - Computer and Information Sciences*, 33(5):497–507, jun 2021.

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.