Arabic Question Answering

Nafe Abubaker[1], Abdullah Hamed[2], Ameer Zedany[3]
dept. Computer Systems
Engineering Birzeit University,
Birzeit, Palestine

1200047@student.birzeit.edu[1], 1202063@student.birzeit.edu[2], 1190482@student.birzeit.edu[3]

***Abstract*— The aim of this project is to develop an effective Arabic question-answering system. Due to the scarcity of robust Arabic question-answering systems, unique challenges arise. To address this, we fine-tuned the `ElectraForQuestionAnswering` model using the `AraElectra-Arabic-SQuADv2-QA` pre-trained model. Our implementation includes installing necessary libraries, tokenization, segmentation, model inference, and evaluation using metrics such as precision, recall, F1 score, and accuracy. By leveraging the Electra model and adapting it for Arabic, we aim to enhance user experiences and make information more accessible in Arabic-language applications. Index Terms—Arabic language, Question Answering, Electra, AraElectra, Fine-Tune.**

I. INTRODUCTION

This project aims to develop an Arabic question-answering system designed to handle specific questions and extract concise answers from paragraphs. Despite the importance of question-answering systems in extracting information from lengthy texts, robust solutions for the Arabic language are scarce. The unique linguistic challenges posed by Arabic necessitate the creation of a specialized system. Our goal is to develop a reliable application capable of accurately extracting answers from Arabic text.

The methodology involves several key steps. First, we install the necessary libraries, including `transformers` and `pyarabic`. We use the `ElectraForQuestionAnswering` model, fine-tuned with the `AraElectra-Arabic-SQuADv2-QA` pre-trained model. This fine-tuning process adapts the model to Arabic linguistic patterns, optimizing its performance for question answering.

The implementation stages include tokenization, segmentation, model inference, and evaluation. We use the `AutoTokenizer` to tokenize the input text, treating the question and context as a text pair. The system assigns segment IDs to the tokenized inputs and processes them through the model to obtain start and end scores. The answer is reconstructed by selecting tokens with the highest scores. We visualize these scores using bar plots to provide insights into the model's confidence. An evaluation function calculates precision, recall, F1 score, and accuracy for each question-answer pair to assess the system's performance.

II. ELECTRA ARCHETICTURE

ELECTRA represents a groundbreeaking development in machine learning, conceived through the collaboration of Google AI Language, Stanford, and Google Brain. This model has garnered widespread acclaim in the field for its innovative architecture and token replacement technique, significantly enhancing language comprehension and processing tasks. ELECTRA differs from BERT by employing a generator-discriminator framework inspired by Generative Adversarial Networks (GANs). In this setup, both components share a transformer architecture but function distinctively. The generator is trained through masked language modeling to predict and sample masked token replacements, while the discriminator's role is to differentiate between original and generated tokens. This synergistic training approach optimizes learning efficiency, culminating in an accurate and effective text encoder. In our project, we utilize the `ElectraForQuestionAnswering` model, fine-tuned with the `AraElectra-Arabic-SQuADv2-QA` pre-trained model. This fine-tuning process adapts the model to Arabic linguistic pattterns, optimizing its performance for question answering. The ELECTRA architecture's advancements have captivated researchers and practitioners alike, paving the way for exciting future research and pushing the frontiers of natural language processing.

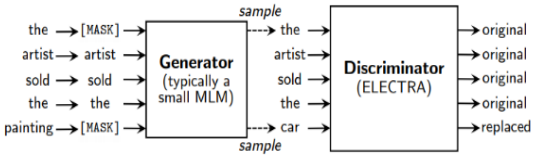


Figure 1 ELECTRA Transformer Architecture

For this project, we utilized the Arabic-SQuADv2.0 dataset, which is a newly developed dataset derived from the widely recognized SQuADv2.0. This dataset is integrated into the fine-tuned `AraElectra-Arabic-SQuADv2-QA` model while preserving the original structure of SQuADv2.0. The Arabic-SQuADv2.0 dataset includes a series of passages or contexts, typically extracted from various articles or documents, organized to maintain the integrity of the source material. Each context is uniquely identified and linked to multiple questions that pertain to its content. These questions are designed to be answerable based on the provided context information. Each question features an answer span, which is a contiguous sequence of words forming the answer, along with an answer start index. Importantly, some questions are intentionally crafted to lack answer spans, highlighting instances where the context does not provide sufficient information to answer the question, thus rendering it unanswerable. This dataset is crucial for training and evaluating our question-answering system, ensuring it can handle the unique challenges posed by the Arabic language. The usee of the Arabic-SQuADv2.0 dataset enables our model to perform effectively in extracting accurate answers from Arabic text. Figure 2

illustrates the data storage format, while Figure 3 shows Feeding a preprocessed example
Arbelectra preprocessor used for preprocessing the dataset in creation and training/testing, since it is recommended to apply the Arabert preprocessing function before training/testing on any dataset.

```
# Example usage: Load dataset from JSON file
dataset_file_path = 'AAQAD-dev.json'
dataset = load_dataset_from_file(dataset_file_path)
```

Figure 2 Code of dataset usage

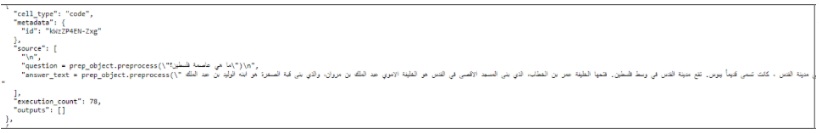


Figure 3 feed preprocessed example.

III. SYSTEM IMPLEMENTATION

A. Installing necessary libraries

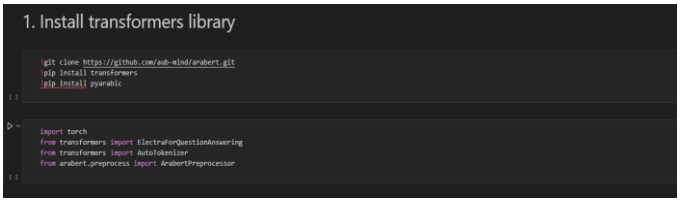


Figure 4 Necessary Libraries

The purpose of this step is to install the necessary libraries and resources for the project. First, the AraBERT repository is cloned from GitHub using the command !git clone https://github.com/aub-mind/arabert.git, which contains code and resources related to the AraBERT model. Next, the Hugging Face transformers library, which provides tools and pre-trained models for natural language processing tasks, including question answering, is installed with !pip install transformers. Additionally, the pyarabic library, offering tools for processing Arabic text, is installed using !pip install pyarabic. The PyTorch framework, used for tensor computation and neural network functionalities, is imported with import torch. The Electra model fine-tuned for question-answering tasks is imported using from transformers import ElectraForQuestionAnswering, and the appropriate tokenizer for the model is loaded with from transformers import AutoTokenizer. Finally, the AraBERT preprocessor, which preprocesses Arabic text, is imported using from arabert.preprocess import ArabertPreprocessor.

B. Loading the fined tuned model

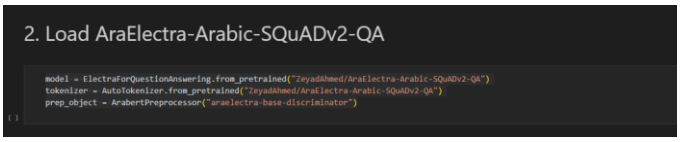


Figure 5 Loading Model

The purpose of this step is to load the pre-trained Electra model and tokenizer, which are specifically fine-tuned for Arabic question answering. The model name, specified as aubmindlab/araelectra-base-discriminator, indicates the particular model to be loaded. The fine-tuned Electra model is loaded using model : ElectraForQuestionAnswering from_pretrained(model_name). And the tokenizer associated with this Electra model is loaded with tokenizer: AutoTokenizer.from_pretrained(model_name).

C. Preprocessing Input Text (Defining Question and text)

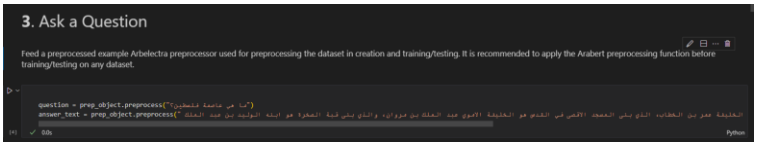


Figure 6 Question and answer

A question to be asked is defined in the code alongside the text from which we seek to extract the answer. This text is ensured to contain the necessary information to address the specified question.

D. Loading the tokenizer

Figure 7 Tokenizer Loading

The tokenizer converts raw input text into a sequence of tokens, the basic units of input for a transformer model. This process involves breaking down the input text into individual words, subwords, or characters, depending on the specific tokenizer used. For this task, the "AutoTokenizer" class from the "transformers" library was imported. A tokenizer instance was created by loading the tokenizer from the "ZeyadAhmed/AraElectra-Arabic-SQuADv2-QA" model, which is specifically designed for Arabic text. This tokenizer effectively decomposes the input text into individual tokens, serving as an essential preprocessing step for model processing. By utilizing the tokenizer, the input text is encoded into token IDs, making it suitable for the ElectraForQuestionAnswering model in question-answering tasks. This ensures the text is properly processed and understood by the model, facilitating accurate question-answering.

E. Apllying the tokenizor to the input text

After defining a question and a text, the tokenizer is appliedto the input text, treating it as a text-pair consisting of a question and an answer. The encode function of the tokenizer takes the question and the text as parameters and converts the text into a sequence of token IDs. The lenngth of the resulting input ids list indicates the total number of tokens. It is printed to define the total number of tokens in the text. Next, to see exactly what the tokenizer is doing, we printed out the tokens with their IDs. The algorithm converts the token IDs backinto their corresponding token strings in order to get the token string and display it. Each token and its ID are printed in two columns. If a token is the special [SEP] token, additional space is added to make it stand out visually. The tokenizer converts the text into

tokens and assigns unique token IDs to each token. The [CLS] token represents the start of the text, and the [SEP] token represents the separation between the question andthe answer. The [UNK] token is a special token that represents an unknown or out-of-vocabulary word. It is typically used when a word or token in the input text is not recognized or in the vocabulary of the tokenizer or language model. To clarify, I the question was: "**ما هي عاصمة فلسطين**", and the answer text was:

" عاصمة فلسطين هي مدينة القدس ، كانت تسمى قديماً يبوس. تقع مدينة القدس في وسط فلسطين. فتحها الخليفة عمر بن الخطاب، الذي بنى المسجد الاقصى في القدس هو الخليفة الاموي عبد الملك بن مروان، والذي بنى قبة الصخرة هو ابنه الوليد بن عبد الملك

Then the tokenizer works as follows:

[CLS]	2
394	ما
634	هي
6,552	عاصمة
2,036	فلسطين
105	؟
[SEP]	3
6,552	عاصمة
2,036	فلسطين
634	هي
1,171	مدينة
3,306	القدس
،	103
678	كانت
11,020	تسمى
17,773	قديما
1,918	يبو
##493	وس
.	20
4,497	تقع
1,171	مدينة
3,306	القدس
305	في
...	
1,089	الملك
[SEP]	3

Figure 8 Tokenizer Output

The algorithm first identifies the index of the first occurrance of the [SEP] token, which separates the question and answer segments. The length of segment A (the question) is determined by the [SEP] index + 1, while the remaining tokens constitute segment B (the answer). Segment IDs are assigned with 0s for segment A tokens and 1s for segment B tokens. An assertion check ensures the correct assignment of segment IDs to each token by verifying that the length of the segment IDs list matches the length of the input IDs list.

The preprocessed input—formatted as input IDs and token type IDs to meet the model's requirements—was then passed to the model. This allows the model to process the input and generate start and end scores, which represent the probabilities of each token being the start and end positions of the answer. The tokens with the highest start and end scores were identified and then joined together with whitespace to form the final answer string. Finally, the answer was printed out, providing the desired response.

After running the input through the model, the code refines the answer string to properly handle subword tokens. It begins with the first identified answer token and processes each subsequent token. If a token starts with "##", it indicates that the token is part of the previous subword and should be merged with it. Otherwise, a space is added before the token. This ensures the answer string is correctly formatted. In models like BERT and ELECTRA, words in the vocabulary can be split into smaller subwords. For example, "playing" might be divided into "play" and "##ing", where the "##" prefix signifies a continuation of the previous subword token.

IV. Scores Visulazation

To create visual representations of scores, the essential libraries matplotlib and seaborn were imported for generating plots. In these plots, the x-axis labels correspond to tokens and their respective indices, which are stored in a token list. The y-axis values are derived from scores, likely representing the start scores for each token.

A. bar plot showing the score for every input word being the "start" word.

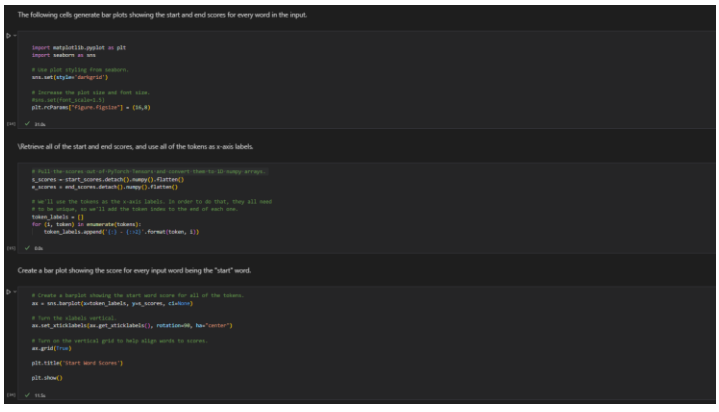
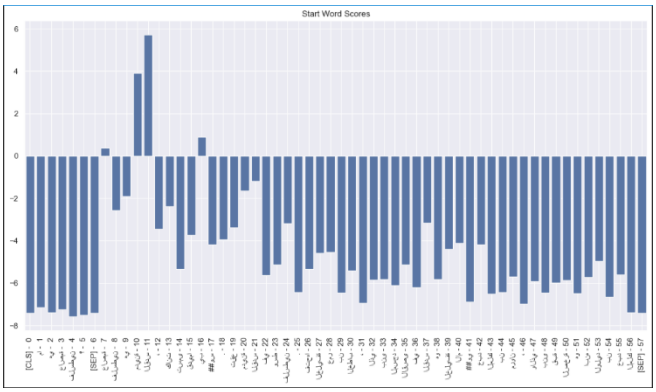


Figure 9 Bar Creation Code



B. *second bar plot showing the score for every input word being the "end" word.*

Figure 11 score for every input word being the "end" word code

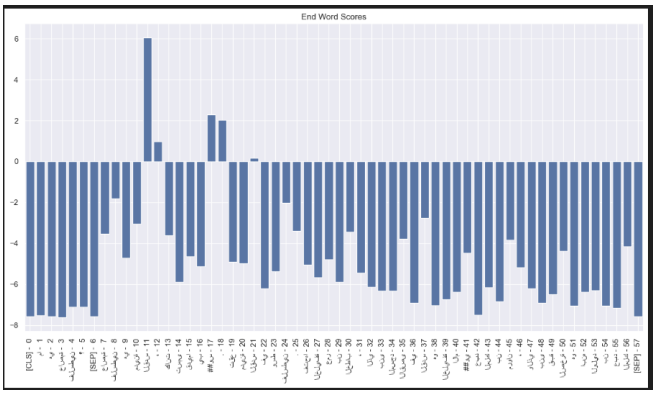


Figure 12 score for every input word being the "end" word

C. *visualizing both the start and end scores on a single bar plot*

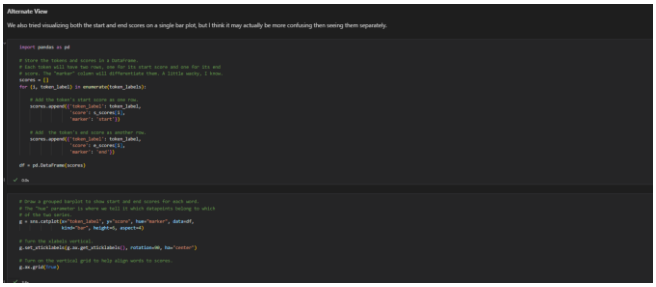


Figure 13 visualizing both the start and end scores on a single bar plot code

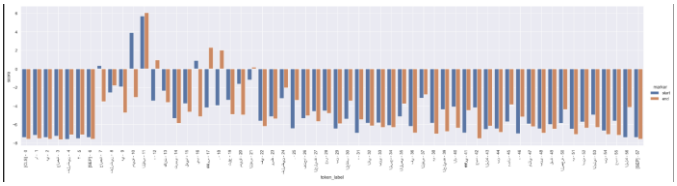


Figure 14 visualizing both the start and end scores on a single bar plot output

V. MODULE TESTING

straightforward Question in the paragraph with a detailed response.

Here, questions directly corresponding to specific answers explicitly mentioned in the paragraph were provided. The objective was to verify whether the code could accurately identify and extract the intended answers. For evaluation, a diverse set of questions commonly found in Arabic language inquiries was selected. These included questions related to place, year, number, reasoning, and general "what" questions. The code's ability to handle different question structures and accurately extract answers in Arabic was thoroughly assessed, resulting in the successful extraction of the intended solutions.

We have Turned the QA process into a function so we can easily try out other examples.

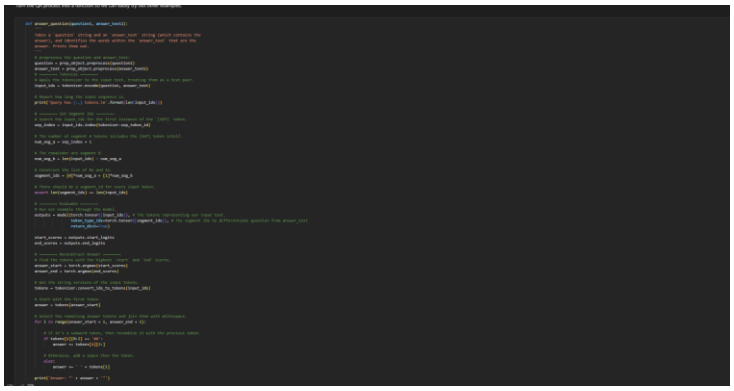


Figure 15 QA Function

A. *Example 1*

Figure 16 Example 1 Data

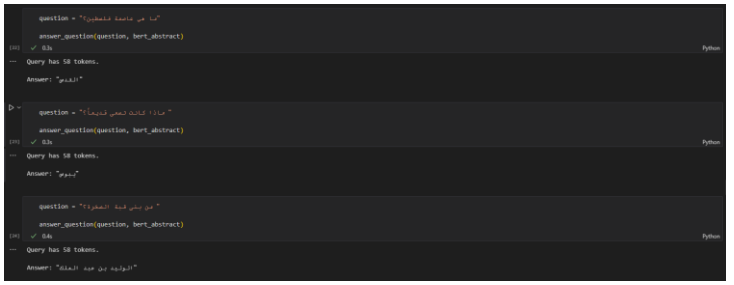


Figure 17 Example 1 Question and Answers

B. *Example 2*

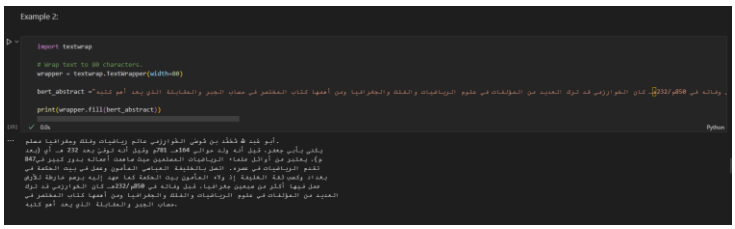


Figure 18 Example 2 Data

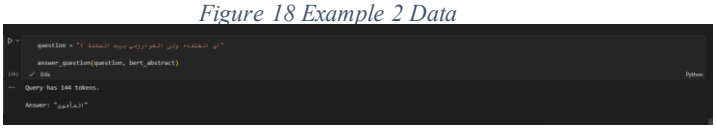


Figure 19 Example 2 Question And Answer

C. Example 3

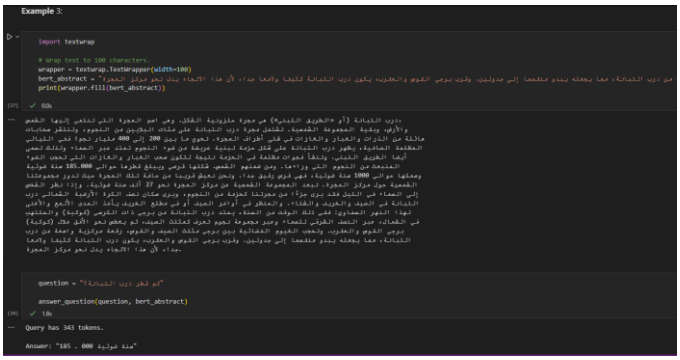


Figure 20 Example 3 Data & Question with answer

VI. Evaluation

As part of our project evaluation, we utilized a subset of the AAQAD (Alexu Arabic Question-Answer Dataset). AAQAD comprises an extensive collection of over 17,000 questions, specifically tailored for our evaluation to address the limitations of small-sized or low-quality datasets. This selected dataset includes questions derived from articles in Modern Standard Arabic (MSA), ensuring a robust and contextually relevant evaluation.

To assess the performance of a question-answering model, an evaluator algorithm was employed. This algorithm takes three inputs: the question to be answered, the text containing the answer, and the correct answer. The evaluation process begins by tokenizing the input text using a tokenizer, followed by encoding the tokens into unique IDs stored in the input_ids variable. The algorithm then identifies the separator token's index, which demarcates the question from the answer, and creates segment IDs to distinguish between question tokens (assigned 0) and answer tokens (assigned 1).

Next, the input_ids and segment IDs are converted into PyTorch tensors to ensure compatibility with the model. The model processes these input tensors, generating outputs that include start and end scores, indicating the likelihood of each token being the start or end of the answer span. Both the predicted and ground truth answers are processed by removing leading and trailing whitespace and converting them to lowercase. Variables for precision, recall, accuracy, and F1 score are initialized to 0.0.

If both the predicted and ground truth answers are non-empty, the algorithm calculates the precision, recall, accuracy, and F1 score. It splits the predicted and ground truth answers into sets of individual tokens and finds the common tokens by determining their intersection. The number of common tokens is calculated. Precision is the ratio of common tokens to predicted tokens, recall is the ratio of common tokens to ground truth tokens, accuracy is the ratio of correctly predicted tokens to the total number of predicted tokens, and the F1 score is the harmonic mean of precision and recall.

$$\text{Recall} = \frac{TP}{TP+FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN} \quad (3)$$

$$\text{F1} = \frac{Recall*Precission}{Recall+Precision} \quad (4)$$

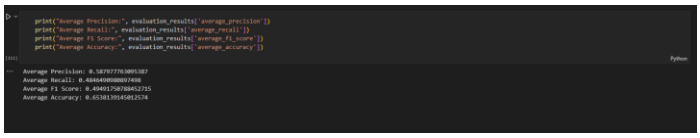


Figure 21 Efficiency Output

As for the results were:

Average Precision: **0.587977763095387**

Average Recall: **0.4846490980897498**

Average F1 Score: **0.49491750788452715**

Average Accuracy: **0.6538139145012574**

VII. Conclusion

In conclusion, this project successfully developed a question-answering system tailored to the Arabic language. The AraElectra model was selected for its suitability and fine-tuned using the Arabic-SQuADv2.0 dataset. The system's implementation included tokenization, segmentation handling, evaluation, and answer reconstruction. Performance metrics confirmed its reliability and effectiveness.

References

[1] Antoun, Wissam & Baly, Fady & Hajj, Hazem. (2020). AraELECTRA: Pre-Training Text Discriminators for Arabic Language Understanding.

[2] Rangasai, K. How do you pick the right set of hyperparameters for a machine learning project? Medium.

[3] zeyadahmed10. Zeyadahmed10/Arabic-MRC. GitHub

[4] adelmeleka. Data Collection/AAQAD Automatic Generator Tool/AAQAD_AutomaticGenerator V3.0.ipynb

[5] 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.

[6] Clark, K., Luong, M.-T., Le, Q. V., & Manning, C. D. (2020). ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. arXiv preprint arXiv:2003.10555.