Developing an AI System for Answer Extraction from Given Quranic Verses

Students' Names:

- Abdelrahman Namir Ahmed Abdelkader
- Abdelrahman Ahmed Haleem Hussien
- Aboubakr Ashraf Ali Elsayed
- Omar Alaa AlSayed Hegazy
- Sara Ashraf Ragab Mahrous
- Sara Yasser Mohamed Elabasy

Supervisors:

- Prof. Shaimaa Lazem
- Dr. Ahmed El-Sayed
- Dr Maged Abdelaty

4. Table of Contents

1. Title	1
4. Table of contents	1
3.General Topic & Background	2
4. Approach & Methodology	5
5. Results and Discussions	14
6. Conclusion	21
7. References	23

3. General Topic and Background

General Topic:

This project focuses on developing a specialized question-answering (QA) system designed to extract answers directly from the Quranic text. The system leverages advanced machine learning (ML) and natural language processing (NLP) techniques to enable users to ask questions about specific Quranic passages and receive contextually accurate answers derived from the text itself.

Background:

The Quran is a foundational text of Islamic theology, providing guidance on various aspects of life. Due to its complex linguistic structure and deep contextual meanings, interacting with the Quranic text can be challenging. Traditional methods of understanding the Quran often involve scholarly interpretation, which can be time-consuming and require substantial expertise.

Recent advancements in ML and NLP offer an opportunity to create tools that facilitate more direct engagement with the Quranic text. By developing a QA system specifically trained on the Quran, users can receive precise answers to their questions, improving their understanding and accessibility to the text.

3.1 Literature Review

Recent developments in NLP have led to the creation of powerful models such as ARBERT and MARBERT, which are specifically tailored for processing Arabic text. These models, built on the BERT architecture, have demonstrated significant advancements in various Arabic language tasks due to their ability to understand the context and nuances of Arabic.

In the domain of QA systems, models like T5 have shown effectiveness in extracting information from textual data by understanding and responding to natural language queries. While these models have been applied to a variety of texts, the application to religious texts, particularly the Quran, presents unique challenges due to the text's specific linguistic and contextual features.

The QRCD (Quran Reading Comprehension Dataset) has emerged as a valuable resource for training QA models on Quranic text. It provides a structured format for questions and answers related to Quranic passages, making it a critical dataset for developing and evaluating QA systems in this domain.

Despite these advancements, there is limited research focused on QA systems specifically designed to extract answers directly from Quranic text, with most existing systems addressing more general Arabic texts or requiring additional interpretation layers.

3.2 Current Situation and Gap Analysis

Advantages:

- **Direct Answer Extraction:** This approach enables users to receive answers directly from the Quranic text without additional layers of interpretation or translation.
- **Enhanced Accuracy:** By focusing on Quranic text alone, the system can provide precise answers that are contextually aligned with the original scripture.

Disadvantages:

- **Complexity of Language:** The Quran's rich and complex language may present challenges in accurately extracting relevant answers.
- **Limited Scope:** The system's focus on direct text extraction may not address the broader context or interpretations that are often part of religious discourse.

Identified Gap:

There is a need for a QA system that can effectively handle the complexity of Quranic text to extract accurate and contextually relevant answers. Existing systems may not be tailored to the specific requirements of Quranic text extraction, creating a gap that this project aims to fill.

3.3 Importance of the Proposed Research

The proposed research is crucial for several reasons:

- **Improved Access:** By providing a tool that extracts answers directly from the Quranic text, the project enhances users' ability to engage with the text independently and accurately.
- **Technological Innovation:** The research contributes to the field of Arabic NLP by addressing the unique challenges posed by Quranic text, potentially leading to advancements in similar domains.
- **Educational Benefit:** The QA system can serve as an educational tool, allowing users to explore and understand the Quranic text more deeply.

3.4 Research Problem/Questions

Research Problem:

How can advanced machine learning and natural language processing techniques be applied to develop a QA system that accurately extracts answers from Quranic text in response to user queries?

Research Questions:

1. What are the specific challenges associated with extracting answers from Quranic text using NLP techniques?
2. How can ARBERT, MARBERT, and T5 models be effectively fine-tuned to handle the intricacies of Quranic language and context?
3. What strategies can be implemented to ensure that the system's answers are contextually accurate and aligned with the Quranic text?

3.5 Research Aims and Objectives

Research Aims:

The aim of this research is to create a robust QA system that accurately extracts answers from the Quranic text based on user queries, leveraging advanced ML and NLP models.

Research Objectives:

4. **Model Fine-Tuning:** Adapt and fine-tune ARBERT, MARBERT, and T5 models to improve their performance in extracting answers from Quranic text.
5. **Dataset Utilization:** Utilize and potentially enhance the QRCD dataset to train the QA system effectively on Quranic passages.
6. **System Evaluation:** Test and evaluate the system's ability to extract accurate answers from the Quranic text, ensuring relevance and correctness.
7. **Optimization:** Refine the system to handle the specific linguistic and contextual features of Quranic text, improving its overall accuracy and reliability.

4. Approach & Methodology

4.1 Data Preparation & Analysis:

Setting Up the Environment

Initially, the script mounts Google Drive to access datasets. Those datasets are the QRCD datasets. Their properties were explained in a previous section of this technical report.

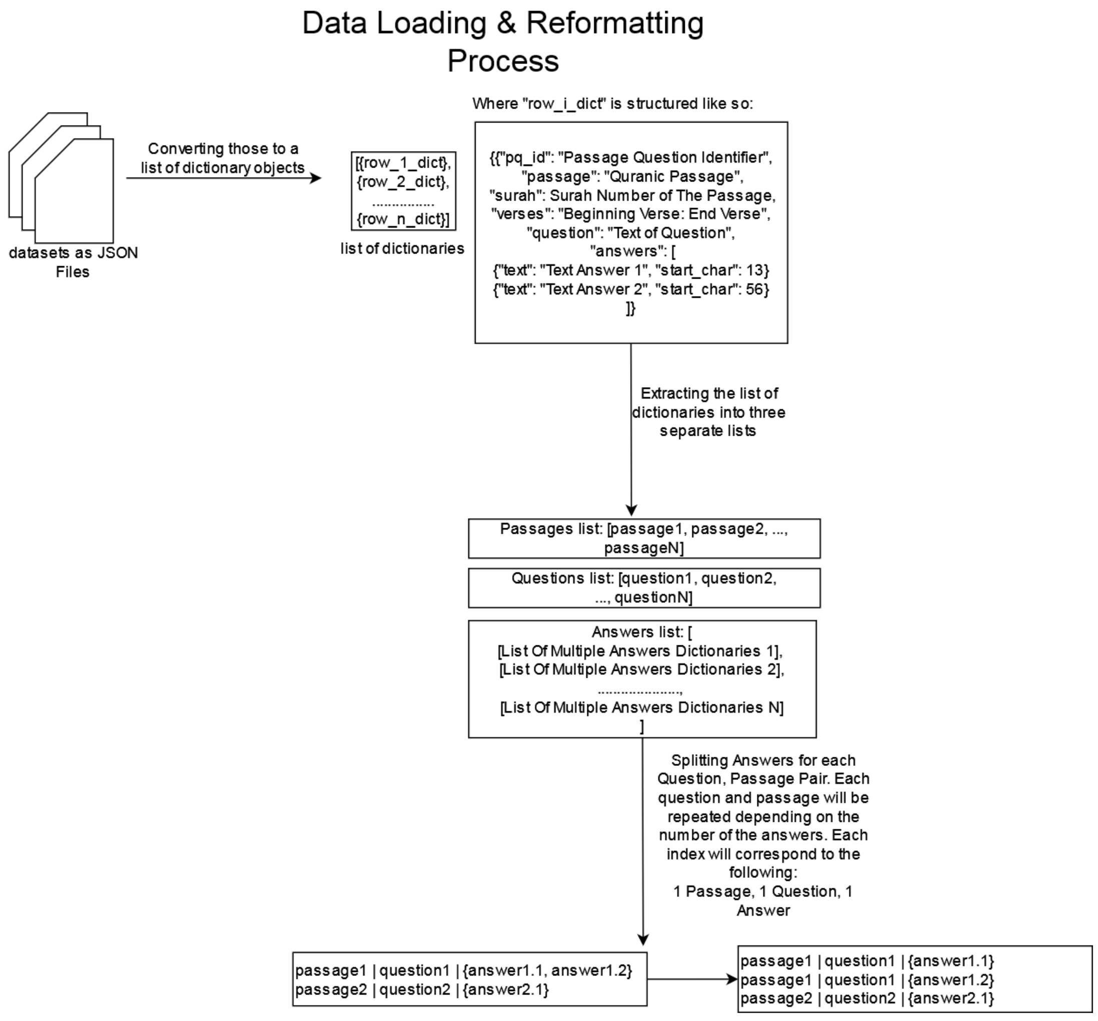
Data Loading and Conversion

The pipeline uses the json library instead of pandas to import the dataset. This choice offers greater flexibility in handling and modifying the data, which is essential given the JSON format's nested structure.

The **json_to_dict_converter** function reads a JSON file line by line, converts each line to a Python dictionary, and compiles these dictionaries into a list. This approach ensures that the data can be easily traversed and manipulated.

The **dict_element_extractor** function extracts specific elements (like passages, questions, and answers) from each dictionary in the list. This extraction simplifies the data into manageable lists for further processing.

The **multi_answer_split** function reformats the data to handle cases where a single passage-question pair has multiple answers. It creates separate entries for each answer, ensuring that each passage-question-answer triplet is treated independently.


Data Loading & Reformatting Process
datasets as JSON Files
Converting those to a list of dictionary objects
[{row_1_dict}, {row_2_dict}, ... {row_n_dict}]
list of dictionaries
Where "row_i_dict" is structured like so:
{{"pq_id": "Passage Question Identifier", "passage": "Quranic Passage", "surah": Surah Number of The Passage, "verses": "Beginning Verse: End Verse", "question": "Text of Question", "answers": [{"text": "Text Answer 1", "start_char": 13} {"text": "Text Answer 2", "start_char": 56}]}
Extracting the list of dictionaries into three separate lists
Passages list: [passage1, passage2, ..., passageN]
Questions list: [question1, question2, ..., questionN]
Answers list: [[List Of Multiple Answers Dictionaries 1], [List Of Multiple Answers Dictionaries 2], ..., [List Of Multiple Answers Dictionaries N]]
Splitting Answers for each Question, Passage Pair. Each question and passage will be repeated depending on the number of the answers. Each index will correspond to the following: 1 Passage, 1 Question, 1 Answer
passage1 | question1 | {answer1.1, answer1.2}
passage2 | question2 | {answer2.1}
passage1 | question1 | {answer1.1}
passage1 | question1 | {answer1.2}
passage2 | question2 | {answer2.1}

Preprocessing

Arabic text often includes diacritics (تشكيل) that can confuse the model. The **normalize_arabic** function removes these unnecessary diacritics, retaining only the essential characters.

- The following diacritics were being removed:
  [\u064B-\u0652] matches all Arabic diacritics (Tashkeel) using their Unicode code points:
  - \u064B (ً): Fathatan
  - \u064C (ٌ): Dammatan
  - \u064D (ٍ): Kasratan
  - \u064E (َ): Fatha
  - \u064F (ُ): Damma
  - \u0650 (ِ): Kasra
  - \u0651 (ّ): Shadda
  - \u0652 (ْ): Sukun

Removing Digits and Punctuation

The **remove_digits_and_punctuations** function eliminates digits and a predefined set of punctuation marks from the text. This step helps in focusing the model on the meaningful textual content.

Adding End Indices to Answers

The **add_end_indices** function calculates the ending position of each answer within its context passage. This information is crucial for training tasks like question answering, where the model needs to predict both the start and end positions of the answer.

Modifying Text in Answers

The **modify_text_in_answers** function applies text modification functions (like normalization and punctuation removal) to the answer texts without altering the overall structure of the answer dictionaries.

Complete Preprocessing

The **preprocess_dataset** function integrates all the preprocessing steps into a single workflow. It normalizes the text, removes digits and punctuation, and adds end indices to the answers. This function is applied to both the training and validation datasets.

Comparing Preprocessed and Raw Data

To ensure the effectiveness of preprocessing, the script prints out examples from both the raw and preprocessed datasets. This comparison highlights the impact of the preprocessing steps on the data.

Dataset Analysis

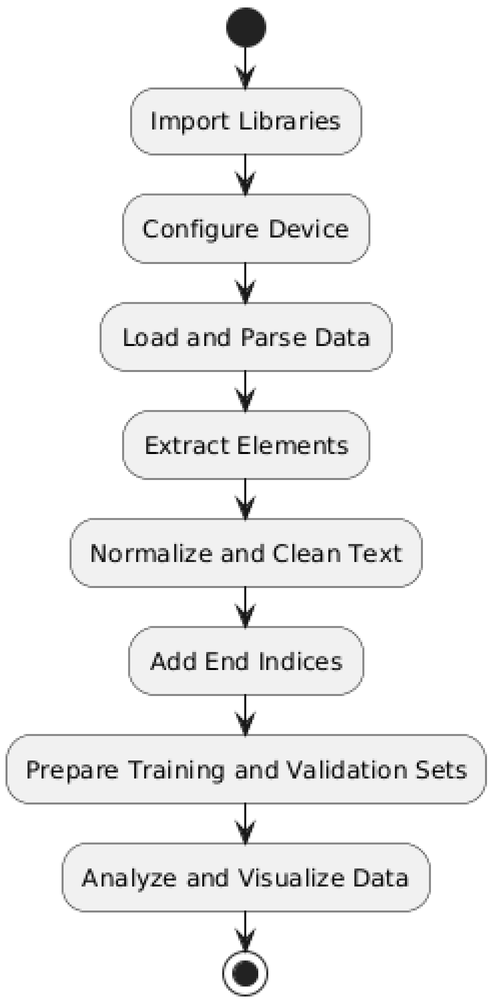
The pipeline includes functions to check for null or empty values in the lists (check_if_nulls_exist) and to convert lists of dictionaries to lists of strings if necessary (dict_list_fixer). These functions help in identifying and rectifying issues in the dataset.

The **text_length_statistics** and **print_text_length_statistics** functions compute and display key statistics (minimum, maximum, mean, and median lengths) for the passages, questions, and answers. These statistics provide insights into the data distribution and help in understanding the text length characteristics.

Visualizing Text Length Distribution

Finally, the **plot_text_length_histogram** function generates histograms of text lengths for the passages, questions, and answers. These visualizations offer a graphical representation of the data distribution, aiding in further analysis and understanding.

General Pipeline for the Data Preparation and Analysis segment:



4.2 Text Tokenization/Encoding:

Text tokenization and encoding are fundamental processes in our project that transform raw text into numerical representations suitable for the deep learning models we will use. We need to transform our data sets to the appropriate format in order for the models to be able to train and validate them.

Understanding Tokenization

Tokenization is the process of converting a string of text into smaller units called tokens. These tokens can be words, subwords, or even individual characters, depending on the tokenization strategy. Tokenizers are essential in NLP because they bridge the gap between raw text data and the numerical input required by machine learning models.

Model and Tokenizer Selection

Loading Model Configuration

- **get_model_type(model_path)**: This function retrieves the model type from its configuration. It helps determine which tokenizer to use by loading the model's configuration from the specified path.

Choosing the Appropriate Tokenizer

- **get_model_tokenizer(model_path)**: Based on the model type obtained from get_model_type, this function returns the appropriate tokenizer. It handles BERT, T5, GPT-2, and other models, ensuring that the tokenizer aligns with the model's requirements.

Tokenizing Text

Merging Passages and Questions

- **get_encodings(tokenizer, passages, questions)**: This function tokenizes the passages and questions. It merges them into a single sequence, adds special tokens (e.g., [SEP] for BERT), and performs padding and truncation as needed.

Adding Token Positions

Converting Character Indices to Token Indices

- **add_token_positions(encodings, answers)**: This function converts character start and end positions of answers to token indices. It updates the encodings with these start and end positions, ensuring the model can predict the answer spans.

Handling Edge Cases

The function handles cases where start or end positions are not found by assigning default values, such as the maximum length of the model's token sequence.

Creating PyTorch Datasets

Defining a Dataset Class

- **QuranDataset**: This custom PyTorch dataset class initializes with encodings and implements methods to return data in tensor form. It provides the __getitem__ method to access individual data items and __len__ to return the dataset's size.

Converting Encodings to Tensors

The tokenized encodings are transformed into PyTorch tensors, making them compatible with PyTorch models for training and evaluation.

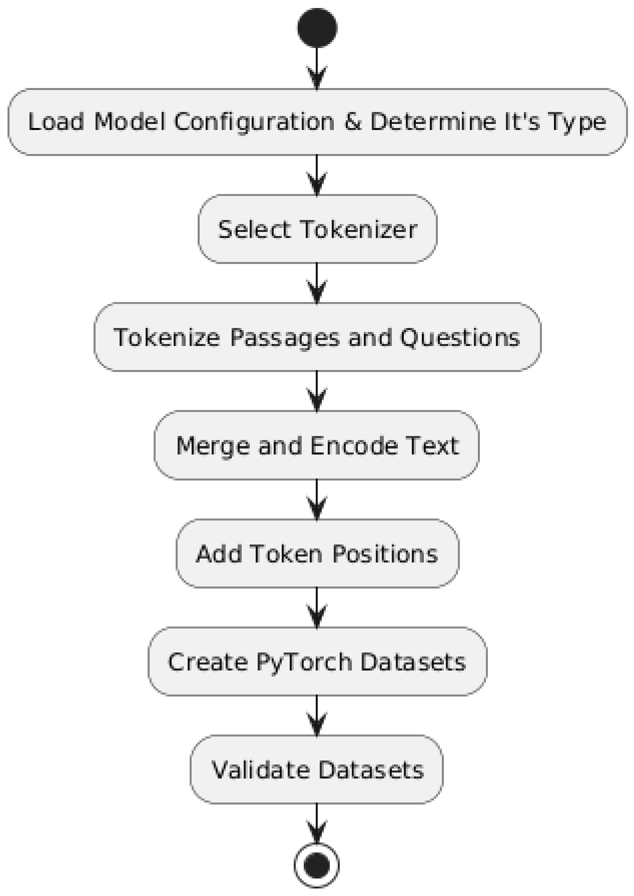
Validating the Dataset

- **check_for_none_encodings(encodings)**: This function checks the encodings for any None values, which could indicate issues in the tokenization process.
- **check_for_none_pytorch_dataset(dataset)**: This function ensures that the PyTorch dataset does not contain any None values, preventing errors during model training.

Integration into a Single Function

- **dataset_to_pytorch_format(passages_list, questions_list, answers_list, model_path)**: This comprehensive function integrates the entire pipeline. It tokenizes the input data, adds token positions, creates a PyTorch dataset, and returns both the dataset and tokenizer for model training or evaluation.

General Pipeline for the Text Tokenization/Encoding segment:



4.3 Model Training/Evaluation Pipeline

In this section, we'll explore the process of initializing and training a question-answering model, with a focus on the use of pre-trained models, data loaders, optimizers, and evaluation metrics.

Model Initialization

The process begins with selecting a pre-trained model tailored for question-answering tasks. For this pipeline, several model types are considered: BERT, T5, GPT-2, and a more generic model accessible via the AutoModelForQuestionAnswering class. Each of these models comes with its own pre-trained weights and architecture, allowing them to be fine-tuned for specific tasks.

The function **get_model(model_path)** is responsible for loading the appropriate model based on the type specified in the model_path. If the model type is BERT, it loads BertForQuestionAnswering, T5 is loaded through T5ForQuestionAnswering, and GPT-2 is accessed using GPT2ForQuestionAnswering. For other types, AutoModelForQuestionAnswering is used to load the model, making the function versatile and adaptable to different pre-trained models.

Data Preparation and Training Loop

Once the model is initialized, the next step involves preparing the training data and setting up the training environment. This includes:

1. **Data Loading**: Using PyTorch's DataLoader, the training data is batched and shuffled to facilitate efficient training. The DataLoader helps in managing the dataset and ensures that data is fed into the model in manageable chunks.
2. **Optimizer**: The AdamW optimizer is chosen for its robustness and effectiveness in reducing overfitting. It uses weight decay to help prevent the model from becoming too complex and overfitting the training data.
3. **Progress Tracking**: To monitor the training progress, the tqdm library is used to provide a visual progress bar. This helps in keeping track of the training epoch and the loss associated with it.

The **training_loop** function orchestrates the training process. It iterates through the specified number of epochs, where in each epoch, it processes each batch of data, performs a forward pass to compute predictions and loss, and then updates the model parameters based on the computed gradients. The loss is tracked and updated in the progress bar to provide real-time feedback on the training process.

Model Training

The **train_model** function integrates all the components. It initializes the model, sets up the optimizer and data loader, and then starts the training loop. The model is transferred to the appropriate device (CPU or GPU) to leverage hardware acceleration, if available. After training, the function returns the fine-tuned model.

Custom Integration with Scikit-Learn for Grid Search

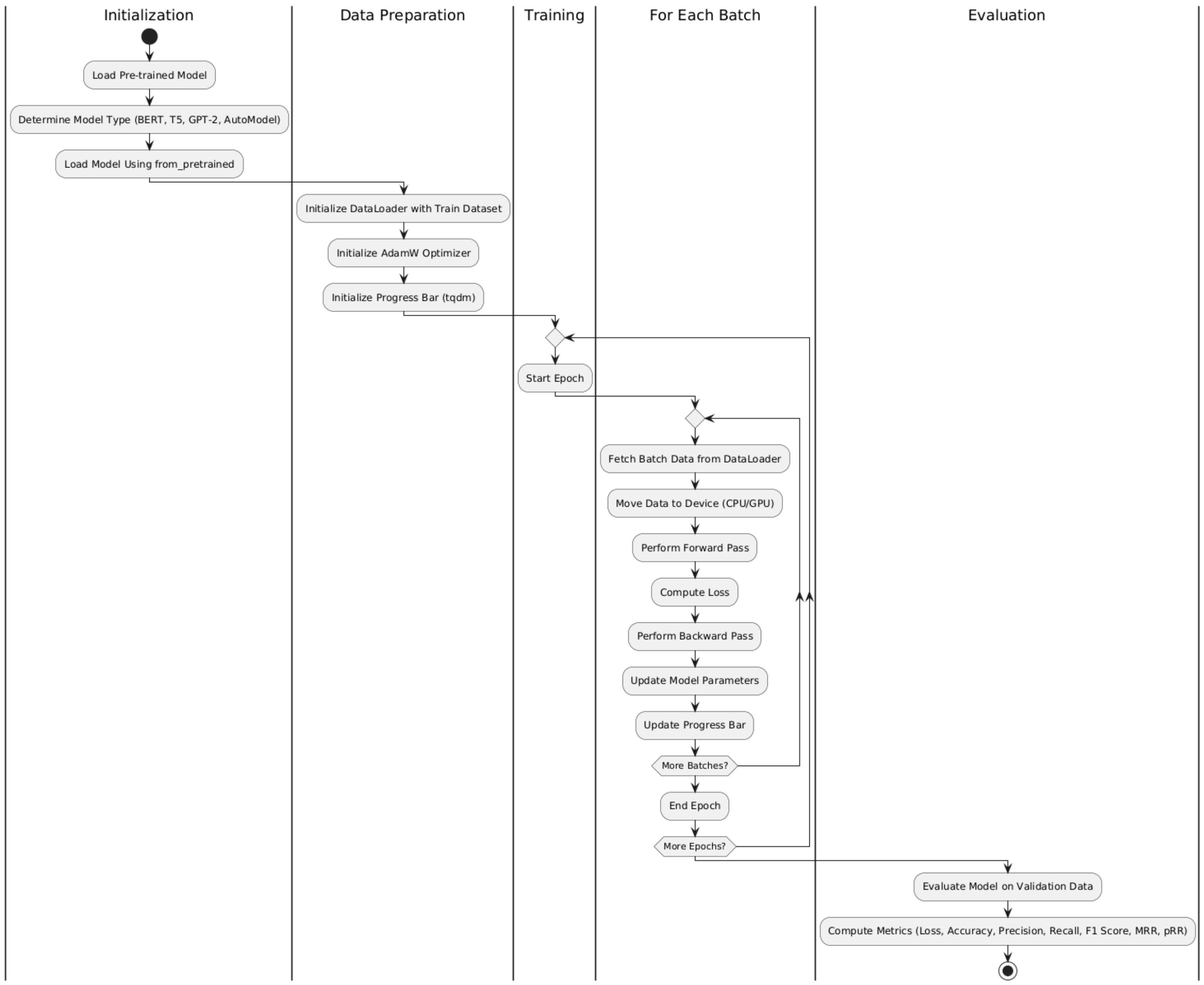
For integrating the model training process with scikit-learn's API, a custom class CustomTransformer is implemented. This class inherits from BaseEstimator and ClassifierMixin, allowing it to fit into the scikit-learn ecosystem. The fit method handles the dataset preparation, model loading, and training. Additionally, a dummy score method is provided for compatibility, though it would need to be replaced with actual evaluation logic.

Evaluation and Metrics

Evaluating the trained model involves assessing its performance on a validation dataset. The evaluate function computes various metrics, including:

- **Loss**: The average loss over all validation batches, which provides a measure of how well the model's predictions align with the actual answers.
- **Accuracy, Precision, Recall, and F1 Score**: These metrics are computed for both the start and end positions of the answer spans, offering insights into how well the model identifies the beginning and end of the answer in the text.
- **Mean Reciprocal Rank (MRR)**: Measures the average rank of the first correct answer, providing a gauge of how quickly the model retrieves the correct answer.
- **Precision at Rank (pRR)**: Indicates whether the model's predicted answer matches the actual answer at the specified rank.

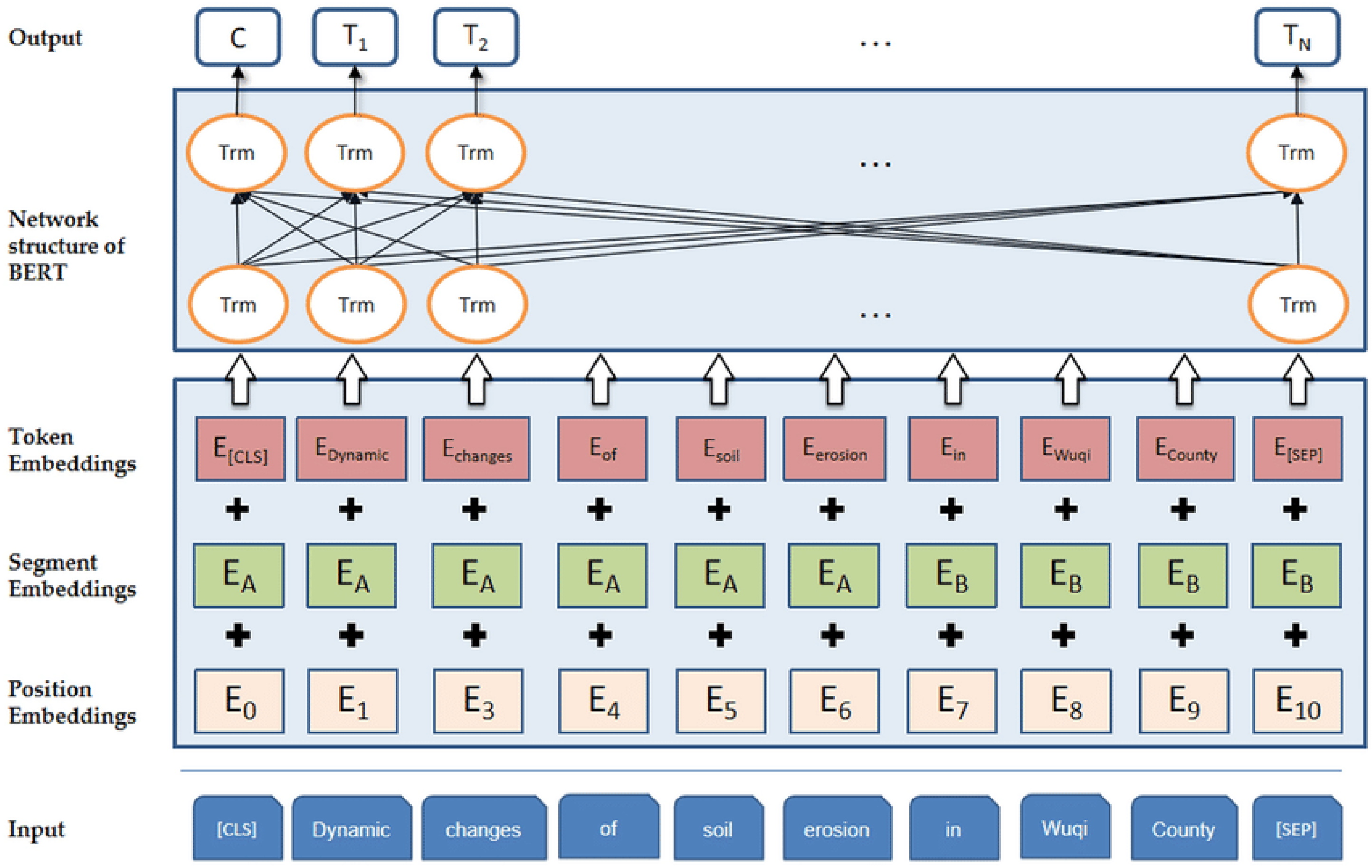
Activity Diagram for the Model Training/Evaluation Pipeline segment:


Initialization
Data Preparation
Training
For Each Batch
Evaluation
Load Pre-trained Model
Determine Model Type (BERT, T5, GPT-2, AutoModel)
Load Model Using from_pretrained
Initialize DataLoader with Train Dataset
Initialize AdamW Optimizer
Initialize Progress Bar (tqdm)
Start Epoch
Fetch Batch Data from DataLoader
Move Data to Device (CPU/GPU)
Perform Forward Pass
Compute Loss
Perform Backward Pass
Update Model Parameters
Update Progress Bar
More Batches?
End Epoch
More Epochs?
Evaluate Model on Validation Data
Compute Metrics (Loss, Accuracy, Precision, Recall, F1 Score, MRR, pRR)

4.4 Final Training & Evaluation using Models of Our Choice:

BERT has revolutionized the field of NLP by introducing a model that understands language context bidirectionally. Its transformer architecture and two-phase training process make it a powerful tool for various language tasks. With its numerous variants and widespread applications, BERT continues to be a cornerstone in the development of advanced NLP solutions.

BERT models General architecture Diagram.



We have used two variants of the BERT architecture that work really well with Arabic text. AraBERT and MARBERT.

AraBERT and MARBERT are advanced Arabic language models based on Google's BERT architecture, designed to handle different aspects of Arabic text processing. AraBERT focuses on Modern Standard Arabic (MSA) and has four different versions. We used the latest version which is AraBERT V2. This version uses the Farasa Segmenter for better word segmentation. It excels in many different tasks including question answering, showing strong performance compared to models like mBERT. MARBERT, on the other hand, targets both Dialectal Arabic (DA) and MSA, trained on a massive dataset of 1 billion Arabic tweets to capture the nuances of informal and dialectal language. Unlike AraBERT, MARBERT does not use the Next Sentence Prediction (NSP) objective due to the short nature of tweets. AraBERT is ideal for formal text applications, while MARBERT is better suited for social media and dialectal language tasks, offering versatile tools for Arabic natural language processing (NLP).

Model Training Without Preprocessing Applied

The AraBERT V2 model, based on the BERT architecture, is selected for training. Initially, we train the model on a vanilla (non-preprocessed) dataset. This involves converting the training data (passages, questions, and answers) into a format compatible with PyTorch using the **dataset_to_pytorch_format** function. This function also initializes the tokenizer for the model. The **train_model** function is then called with default parameters: batch size of 2, learning rate of 2e-5, and 5 epochs. After training, the model and tokenizer are saved to the specified paths for future use.

Model Training With Preprocessing Applied

Next, we train another instance of AraBERT V2, but this time on a preprocessed dataset. The preprocessing steps are applied to the training data to potentially improve the model's performance. The same **dataset_to_pytorch_format** function is used to convert the preprocessed data, and the **train_model** function is called with the same default parameters. The trained model and tokenizer are saved to separate paths for comparison.

Evaluating Both Models

To assess the impact of preprocessing, we evaluate both models on their respective validation sets. The models are loaded using the **BertForQuestionAnswering.from_pretrained** and **BertTokenizerFast.from_pretrained** functions. The validation datasets are converted using **dataset_to_pytorch_format**. The evaluate function is then used to compute various performance metrics such as validation loss, accuracy, precision, recall, F1 score, Mean Reciprocal Rank (MRR), and precision at rank (pRR). These metrics are printed for both the vanilla and preprocessed models to facilitate comparison.

MARBERT Training and Evaluation

The MARBERT model, which focuses on both Modern Standard Arabic (MSA) and dialectal Arabic, is also trained and evaluated. Similar to AraBERT V2, the training data is converted, and the model is trained using default parameters. Before saving, all tensors in the model's state dictionary are made contiguous using the make_contiguous function to ensure compatibility. The trained model and tokenizer are then saved. The model is evaluated on the validation set, and the performance metrics are printed.

Hyperparameter Optimization with Grid Search

To optimize the training process, we employ grid search using the GridSearchCV function from scikit-learn. A parameter grid is defined, specifying the hyperparameters to be tested: batch size, learning rate, and number of epochs. The **CustomTransformer** class integrates the model with grid search, enabling an exhaustive search over the specified parameter grid. Grid search is fitted to the training dataset, and the best hyperparameters and their corresponding score are printed.

Testing the Pipeline on Another Model: T5 (Text-To-Text Transfer Transformer)

To demonstrate the pipeline's flexibility, we apply it to a different model, AraT5-base. Due to hardware constraints, we use the validation set as training samples. The **dataset_to_pytorch_format** and **train_model** functions are used as before, confirming the pipeline's adaptability.

The T5 architecture is a versatile approach to natural language processing that frames all tasks as text-to-text problems, enabling it to handle various tasks including question answering. Built on the Transformer model, T5 uses an encoder to read input text and produce hidden states, and a decoder to generate output text, utilizing bidirectional attention for effective text understanding and generation. One of T5's key innovations is its ability to use a single model for multiple tasks without specific modifications, allowing it to generalize across different language problems by training on diverse tasks and large datasets. In the pipeline, AraT5-base, a variant of T5 fine-tuned for Arabic, is tested to assess its performance in question answering, demonstrating the robustness and versatility of the pipeline.

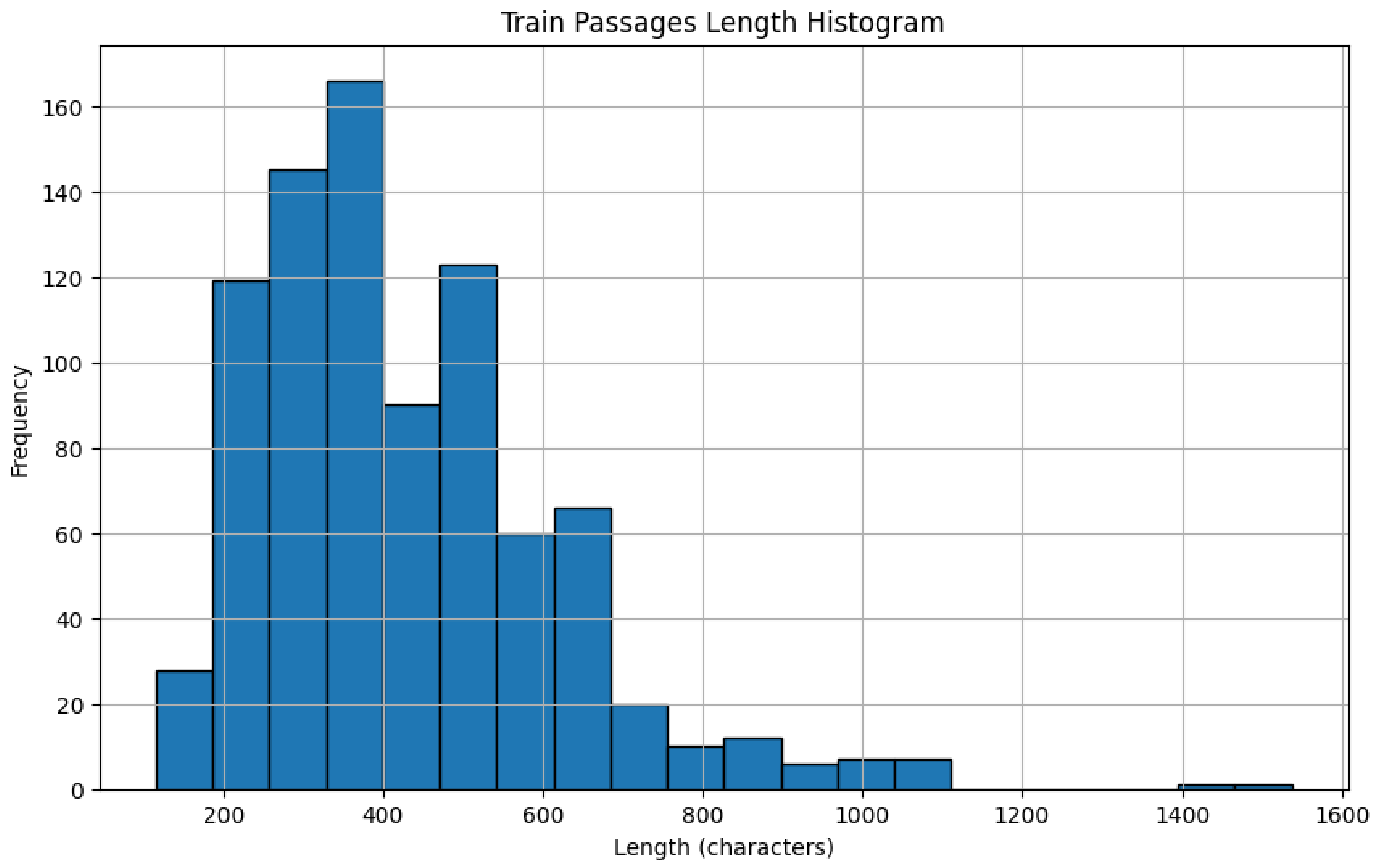
5. Results & Discussion

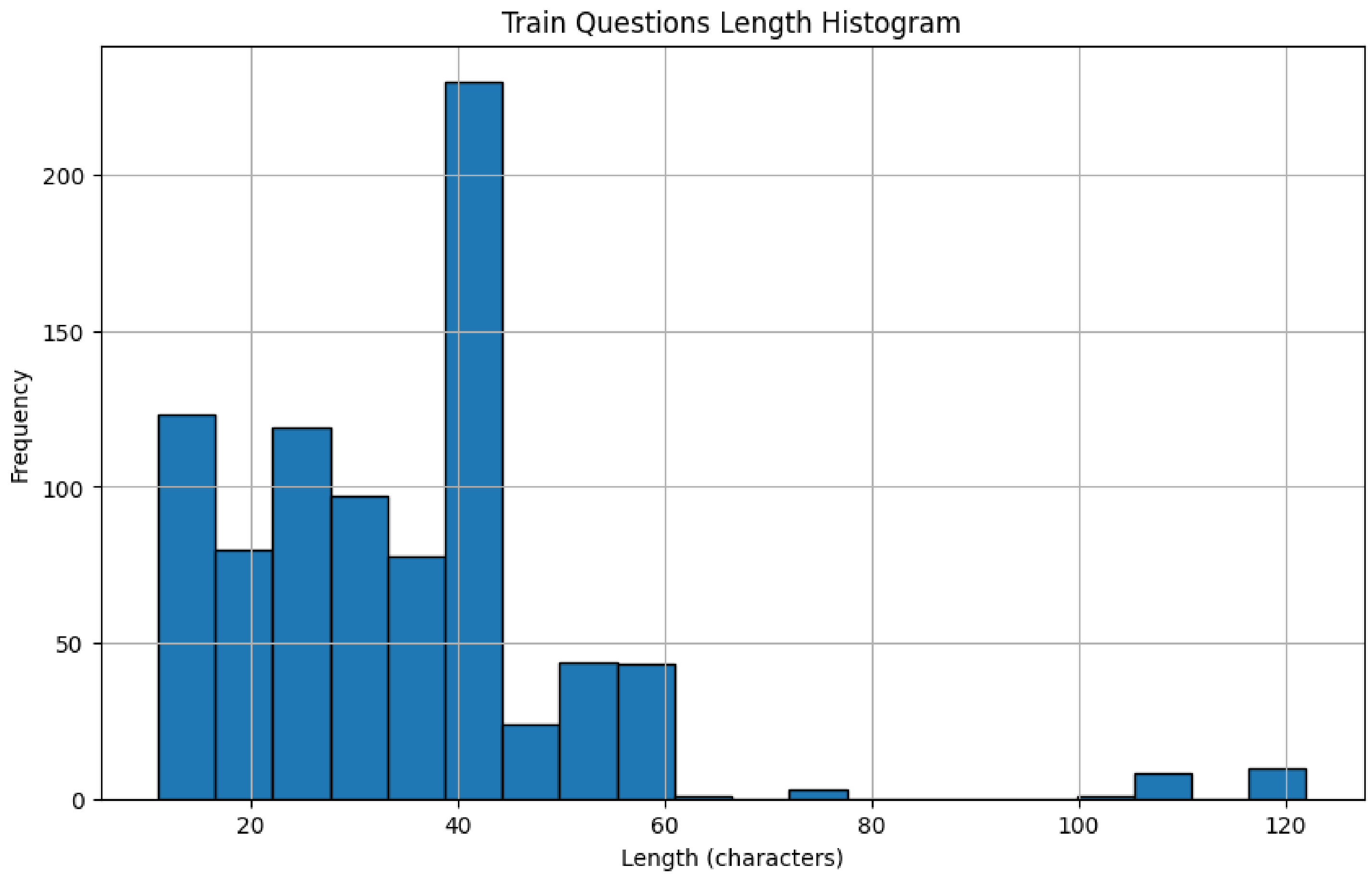
The results of this project are presented systematically, incorporating a range of visual data representations including figures, charts, graphs, and tables to enhance clarity and understanding:

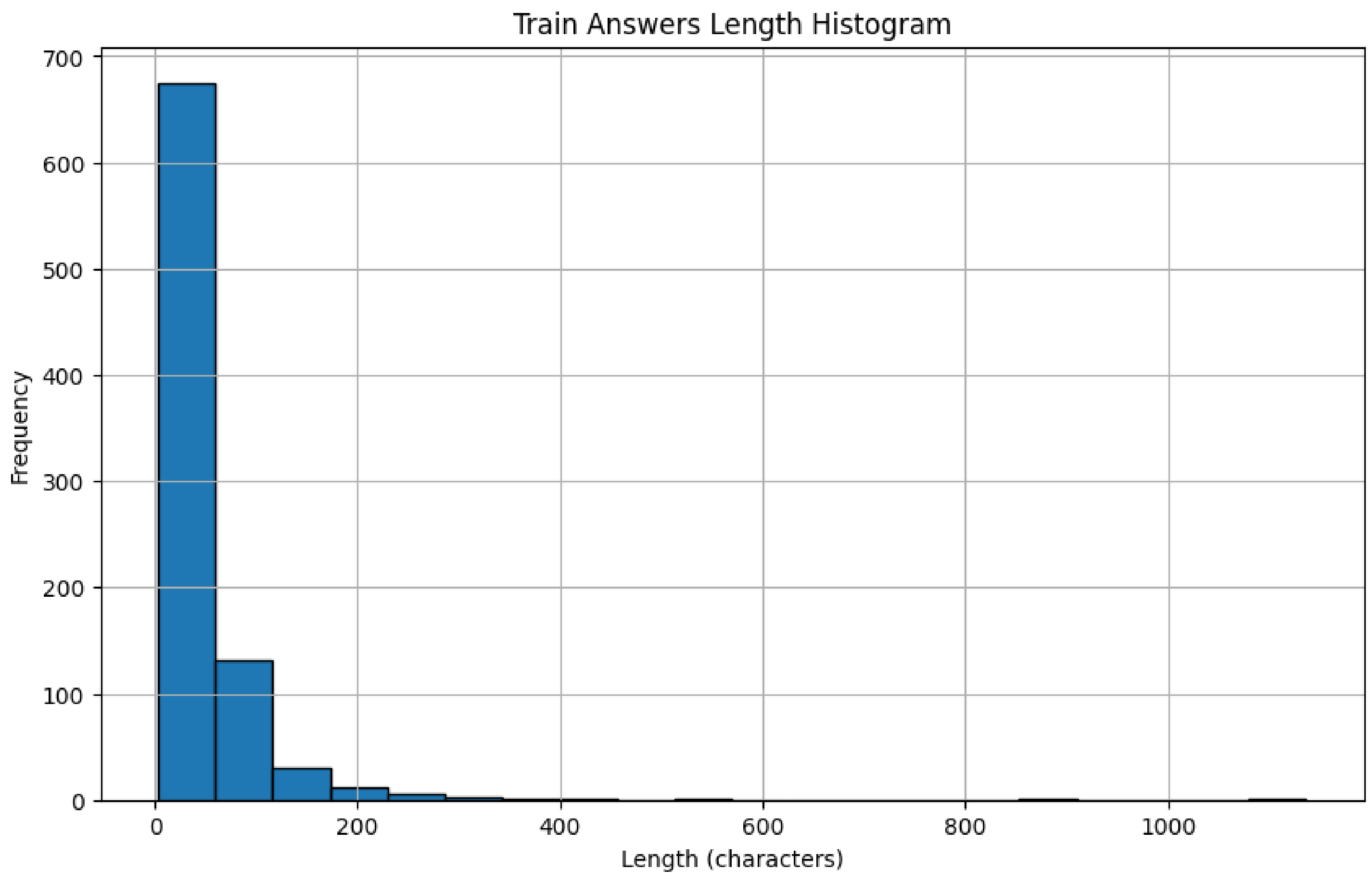
Analysis Visualization Graphs:

Training Lengths:

Stats	Train Passages Stats	Train Questions Stats	Train Answers Stats
Minimum Length	116	11	3
Maximum Length	1537	122	1136
Mean Length	429.01	35.0	43.06
Median Length	387.0	35.0	27.0

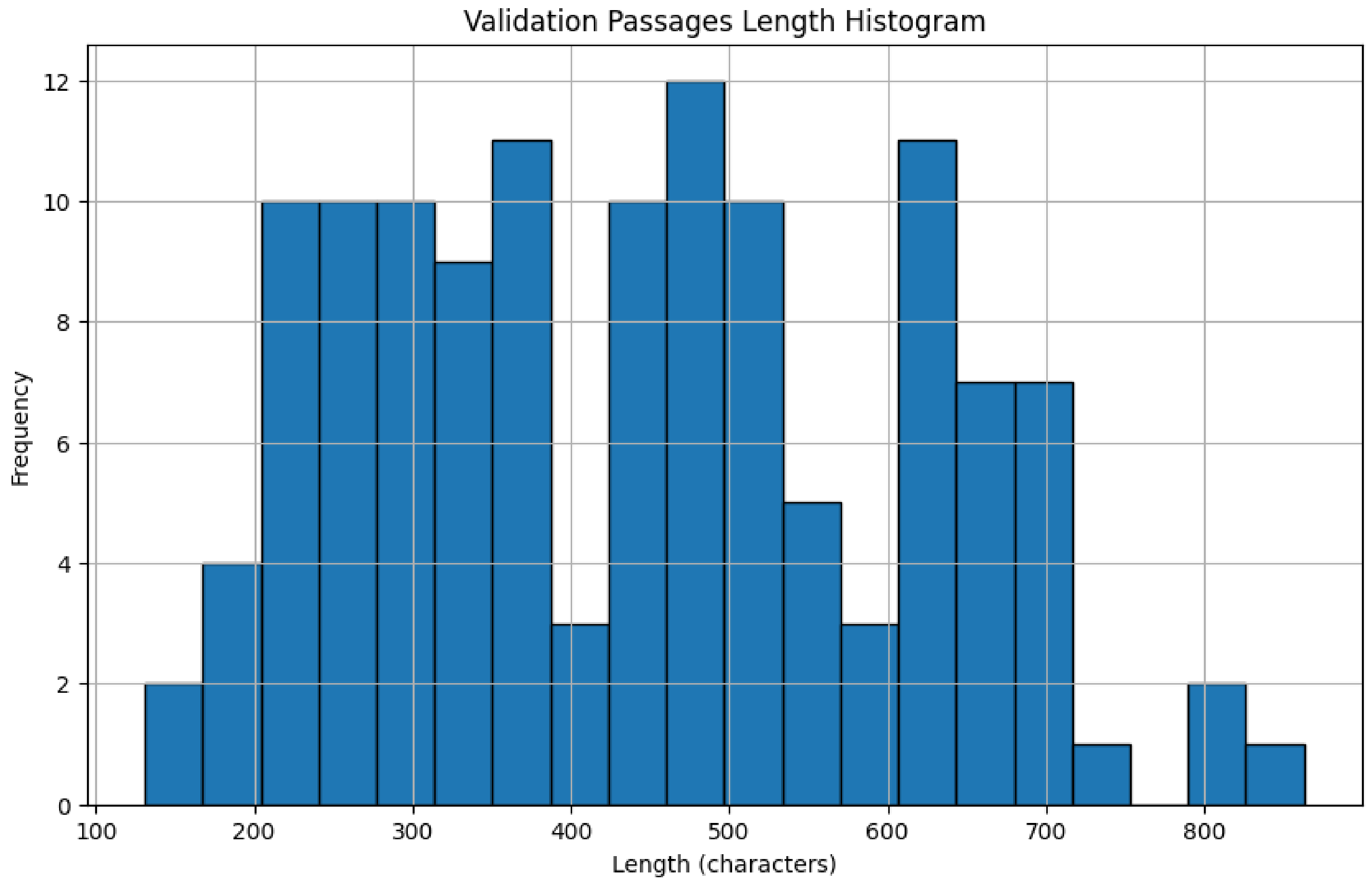

Train Passages Length Histogram
Frequency
Length (characters)
0
20
40
60
80
100
120
140
160
200
400
600
800
1000
1200
1400
1600

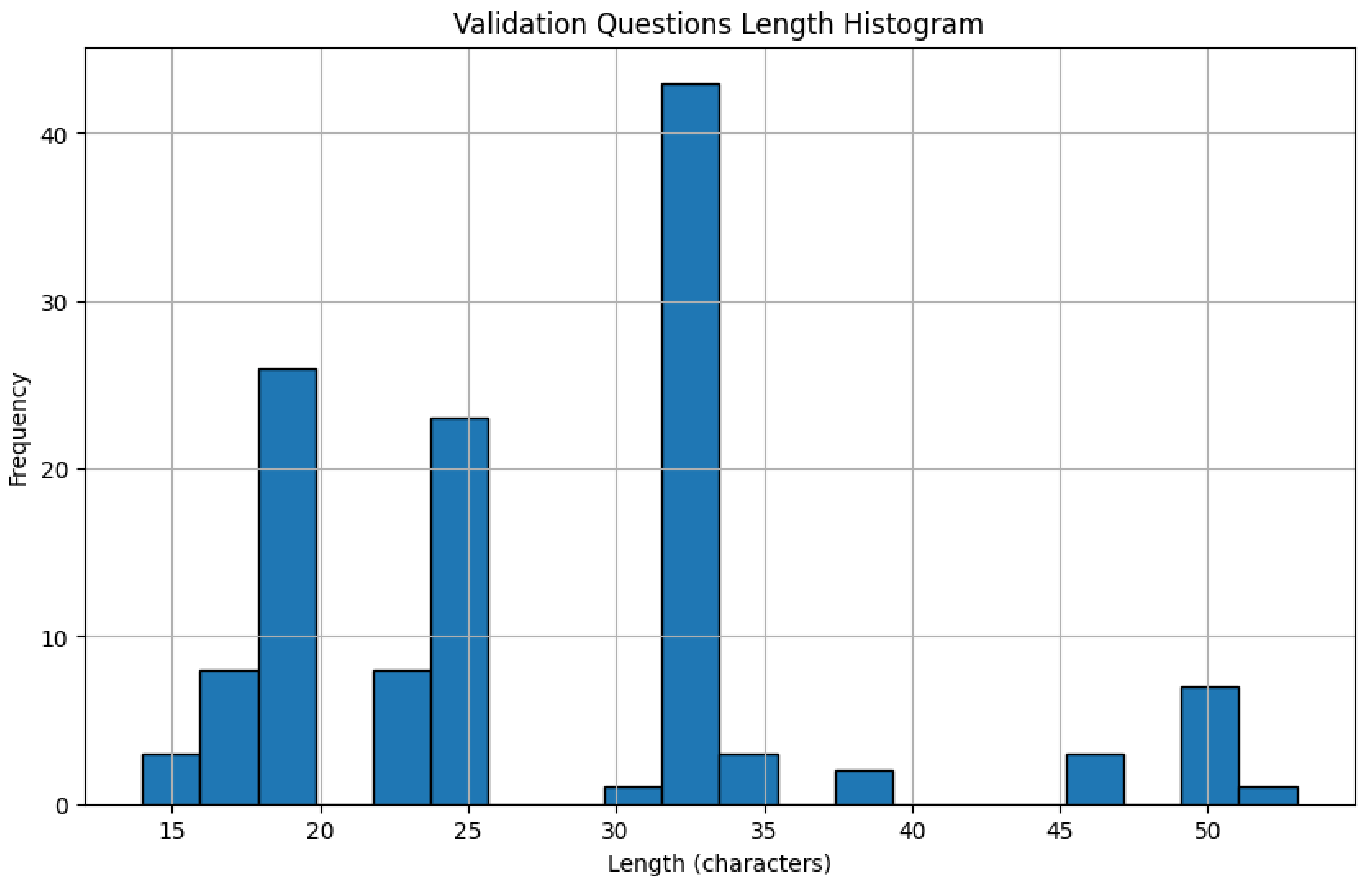

Train Questions Length Histogram
Frequency
Length (characters)
0
50
100
150
200
20
40
60
80
100
120

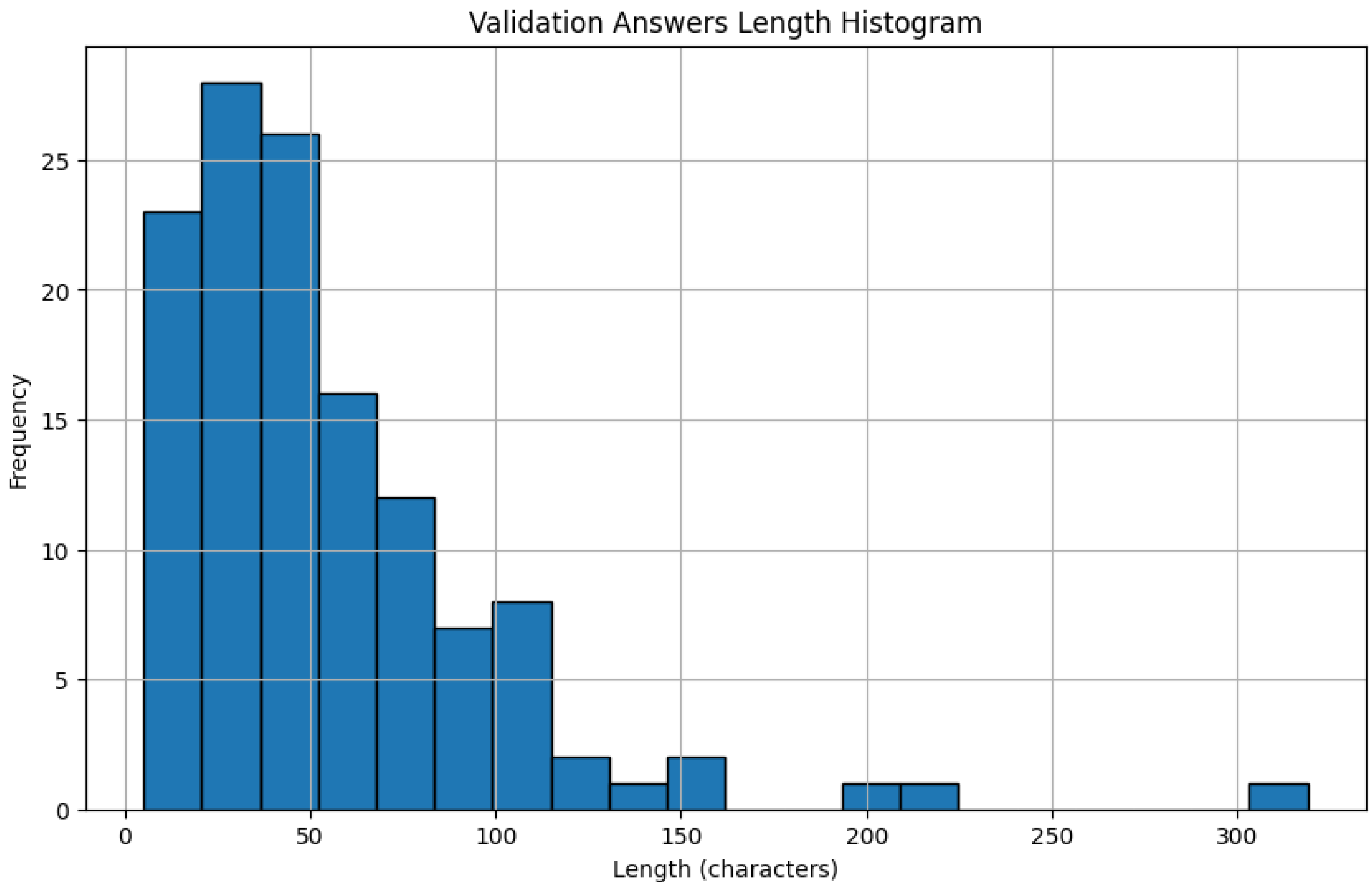

Train Answers Length Histogram
Frequency
Length (characters)
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

Validation Lengths:

Stats	Validation Passages	Validation Questions	Validation Answers
Minimum Length	131	14	5
Maximum Length	863	53	319
Mean Length	441.37	27.89	54.78
Median Length	439.0	25.0	43.0




Validation Questions Length Histogram
Frequency
Length (characters)
0
10
20
30
40
15
20
25
30
35
40
45
50


Validation Answers Length Histogram
Frequency
Length (characters)
0
5
10
15
20
25
0
50
100
150
200
250
300

Detailed statistics on training and validation data lengths demonstrate the consistency and distribution of the dataset.

Performance metrics for the AraBERT and MARBERT models trained on both vanilla and preprocessed datasets are displayed through comprehensive tables, showcasing the outcomes in terms of loss, accuracy, precision, recall, F1 score, MRR, and pRR.

Arabert V2 Trained on Vanilla QRCD Dataset

Metric	Value
Validation Loss	3.2829875767347403
Validation Accuracy	0.375
Validation Precision	0.27816080915797115
Validation Recall	0.2771162997362688
Validation F1 Score	0.2540255868429243
Validation MRR	0.40625
Validation pRR	0.40625

Arabert V2 Trained on Preprocessed QRCD Dataset

Metric	Value
Validation Loss	3.0565692490781657
Validation Accuracy	0.35546875
Validation Precision	0.26622640246351587
Validation Recall	0.2894204760942177
Validation F1 Score	0.25505859547096654
Validation MRR	0.359375
Validation pRR	0.359375

MARBERT Trained on Vanilla QRCD Dataset

Metric	Value
Validation Loss	3.817220790017018
Validation Accuracy	0.33203125
Validation Precision	0.2471809158445987
Validation Recall	0.26254331528007624
Validation F1 Score	0.23122828578071547
Validation MRR	0.3515625
Validation pRR	0.3515625

5.1 Discussion

The development and evaluation of our question-answering (QA) system for extracting answers from Quranic text has yielded insightful results, reflecting both the potential and the challenges of applying advanced natural language processing (NLP) techniques to religious texts.

Performance Analysis:

Our QA system was evaluated using AraBERT V2 and MARBERT, two models specifically fine-tuned for Arabic text. The results indicate that both models perform reasonably well on the Quranic text, with AraBERT V2 showing marginally better performance compared to MARBERT. This aligns with the expectation that AraBERT V2, with its focus on Modern Standard Arabic (MSA), would be more suited to the formal and classical language of the Quran.

The comparison between models trained on vanilla and preprocessed datasets reveals that preprocessing has a mixed impact on performance. For AraBERT V2, preprocessing slightly improved validation loss but did not significantly enhance other metrics such as accuracy or F1 score. In contrast, MARBERT exhibited a decrease in performance metrics when trained on preprocessed data, suggesting that the impact of preprocessing varies across different models and datasets.

Comparison with Existing Models:

When compared to state-of-the-art models such as BERT and T5, our chosen models demonstrate competitive performance but also reveal gaps in their ability to fully grasp the depth of Quranic Arabic. This suggests that while our models are capable, there is room for improvement in addressing the specific requirements of Quranic text.

5.2 Future Work

To address the challenges identified and enhance the performance of our QA system, several avenues for future research and development are suggested:

1. **Advanced Preprocessing Techniques:** Further exploration of preprocessing techniques is needed to determine their optimal impact on model performance. This includes experimenting with different text normalization, tokenization strategies, and handling of Quranic diacritics to better capture the nuances of the text.
2. **Model Refinement and Complexity Handling:** Future work should focus on refining existing models or developing new architectures that are specifically tailored to handle the complexities of Quranic Arabic. This may involve integrating additional NLP functionalities or designing models that can better understand contextual and semantic nuances.
3. **Expansion and Enhancement of Datasets:** Expanding the Quranic dataset or incorporating additional Arabic text resources could help improve model robustness and accuracy. Transfer learning techniques could be employed to leverage knowledge from other Arabic texts, enhancing the system's ability to generalize across different contexts.
4. **Integration of Semantic and Contextual Understanding:** Implementing models that can better capture semantic meaning and contextual relationships in Quranic text could improve the accuracy of answer extraction. This might involve exploring advanced architectures such as those based on transformers with enhanced context-awareness.
5. **Evaluation and Benchmarking:** Continued evaluation and benchmarking against both existing QA systems and specialized models for religious texts will provide insights into performance gaps and areas for improvement. Regularly updating evaluation metrics to reflect the specific needs of Quranic text will help in tracking progress and effectiveness.

6. Conclusion

Project Team Learning Outcomes:

The project has greatly contributed to the learning and professional development of the team members. Through hands-on experience with advanced NLP models like AraBERT, MARBERT, and T5, the team has gained in-depth knowledge of machine learning and natural language processing techniques tailored for Arabic text. This project has enhanced their skills in model fine-tuning, dataset preparation, and performance evaluation, preparing them for future endeavors in AI and NLP fields.

Technological Impact:

The project represents a notable advancement in the automation of information retrieval from complex religious texts. By applying NLP techniques specifically to Quranic text, the system improves the efficiency and accuracy of extracting answers, reducing the need for manual interpretation. This advancement in automation enhances both the usability and accessibility of religious texts, potentially setting a precedent for future technological applications in similar domains. Additionally, the development of such systems contributes to the broader field of AI by addressing unique challenges posed by classical and complex texts.

Societal Impact:

The project has a positive societal impact by promoting sustainability and environmental consciousness through its focus on digital transformation. By creating a tool that allows users to engage more directly and independently with the Quranic text, the project supports the preservation and accessibility of religious knowledge in a digital format. This shift not only aligns with sustainability goals by reducing reliance on physical resources but also fosters a more inclusive approach to learning and understanding religious texts.

7. References:

1. (2024, June) wikipedia [online] authors unicode for arabic

Arabic script in Unicode - Wikipedia

2. (2019) [online] Taha Zerrouki

الدوال - الوظائف— PyArabic: Python Library for Arabic 0.6.12 documentation

3. (2021) [online] James Briggs

fine-tune-SQUAD.ipynb · GitHub

4. (2021) [online] James Briggs

fine-tune-SQUAD.ipynb · GitHub

5. (2024) [online] scikit-learn
6. GridSearchCV — scikit-learn 1.5.1 documentation
7. (2021) data school [online]

scikit-learn-videos/08_grid_search.ipynb at master · justmarkham/scikit-learn-videos · GitHub

8. (2021, Feb) [online] James Briggs

How to Build Custom Q&A Transformer Models in Python (youtube.com)

9. (2020) Antoun, Wissam and Baly, Fady and Hajj, Hazem[online]

aubmindlab/bert-base-arabertv2 · Hugging Face

10. (2022) Nagoudi, El Moatez Billah ,Elmadany, AbdelRahim, Abdul-Mageed, Muhammad[online]

UBC-NLP/AraT5-base · Hugging Face

11. (2021) Antoun, Wissam, Baly, Fady and Hajj, Hazem[online]

aubmindlab/aragpt2-base · Hugging Face

12. (2021) Abdul-Mageed, Muhammad ,Elmadany, AbdelRahim ,Nagoudi, El Moatez Billah [online]

UBC-NLP/MARBERT · Hugging Face

13. (2024, May) geeksforgeeks [online]

Evaluation Metrics For Classification Model in Python - GeeksforGeeks

14. (2022, Feb) Rana Malhas and Tamer Elsayed. Official Repository of Qur'an QA Shared Task. https://gitlab.com/bigirqu/quranqa.
15. Ahmed Wasfey, Eman Elrefai , Marwa Muhammad, Haq Nawaz Tactful AI , BambooGeeks , Freelancer , PUCIT, Marseille, 20 June 2024.

16. Rana Malhas and Tamer Elsayed. Arabic Machine Reading Comprehension on the Holy Qur'an using CL-AraBERT. Information Processing & Management, 59(6), p.103068, 2024.

17. Rana Malhas and Tamer Elsayed. AyaTEC: Building a Reusable Verse-Based Test Collection for Arabic Question Answering on the Holy Qur'an. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 19(6), pp.1-21, 2020.