

Developing an AI System for Answer Extraction from Quranic Verses

Team Members

- 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

General Topic & 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 Deep Learning 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, 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 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.

General Topic & Background

- **Advantages:**

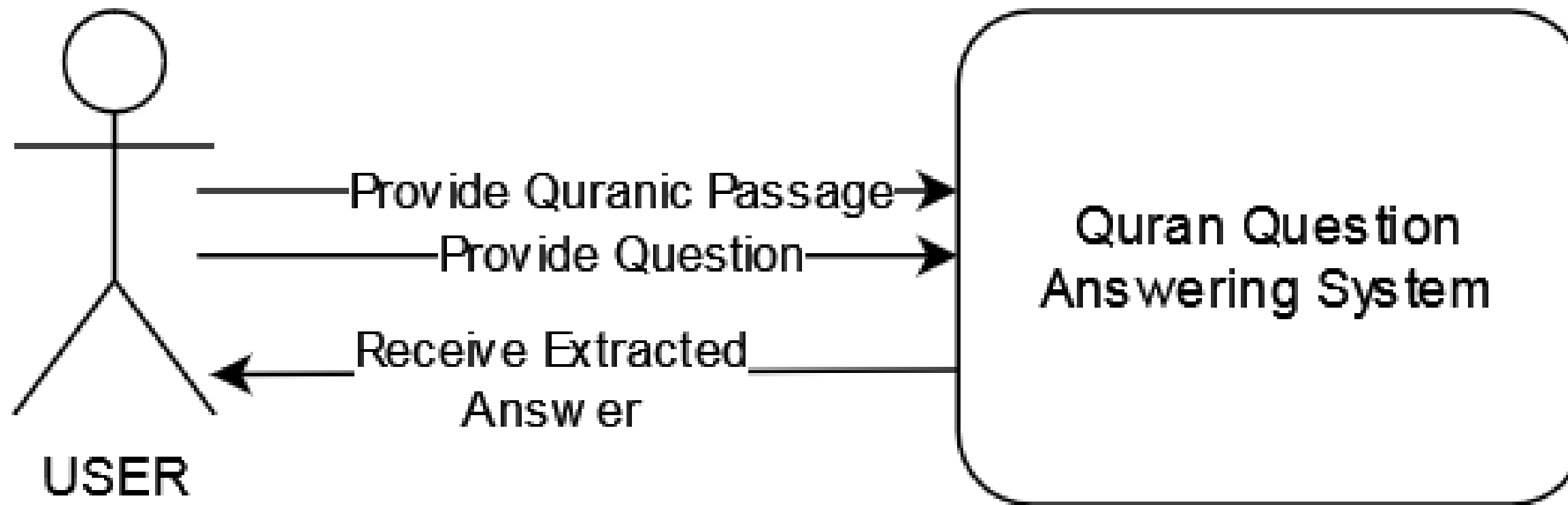
- **Direct Answer Extraction:** This approach enables users to receive answers directly from the Quranic text without additional layers of interpretation or translation.
- **Enhanced Accuracy:** By focusing on Quranic text alone, the system can provide precise answers that are contextually aligned with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

- **Disadvantages:**

- **Lack of Context:** the model take an irrelevant question to the given passage, the answer will be wrong.

System Usage

The program runs a Python cell where the user is prompted to input a Quranic passage and a related question. Upon receiving these inputs, the system processes the passage to extract and display the answer to the user's question. This interactive process ensures that users receive accurate answers based on the specific text they provide.



Approach & Methodology

- **Data Preparation & Analysis:**

Base:

- The dataset for this Arabic Reading Comprehension task, focused on the Holy Qur'an, is designed to evaluate language understanding in computer systems by posing questions over given text passages. The Holy Qur'an, written in Classical Arabic, contains 114 chapters and 6,236 verses, totaling over 80,000 words.

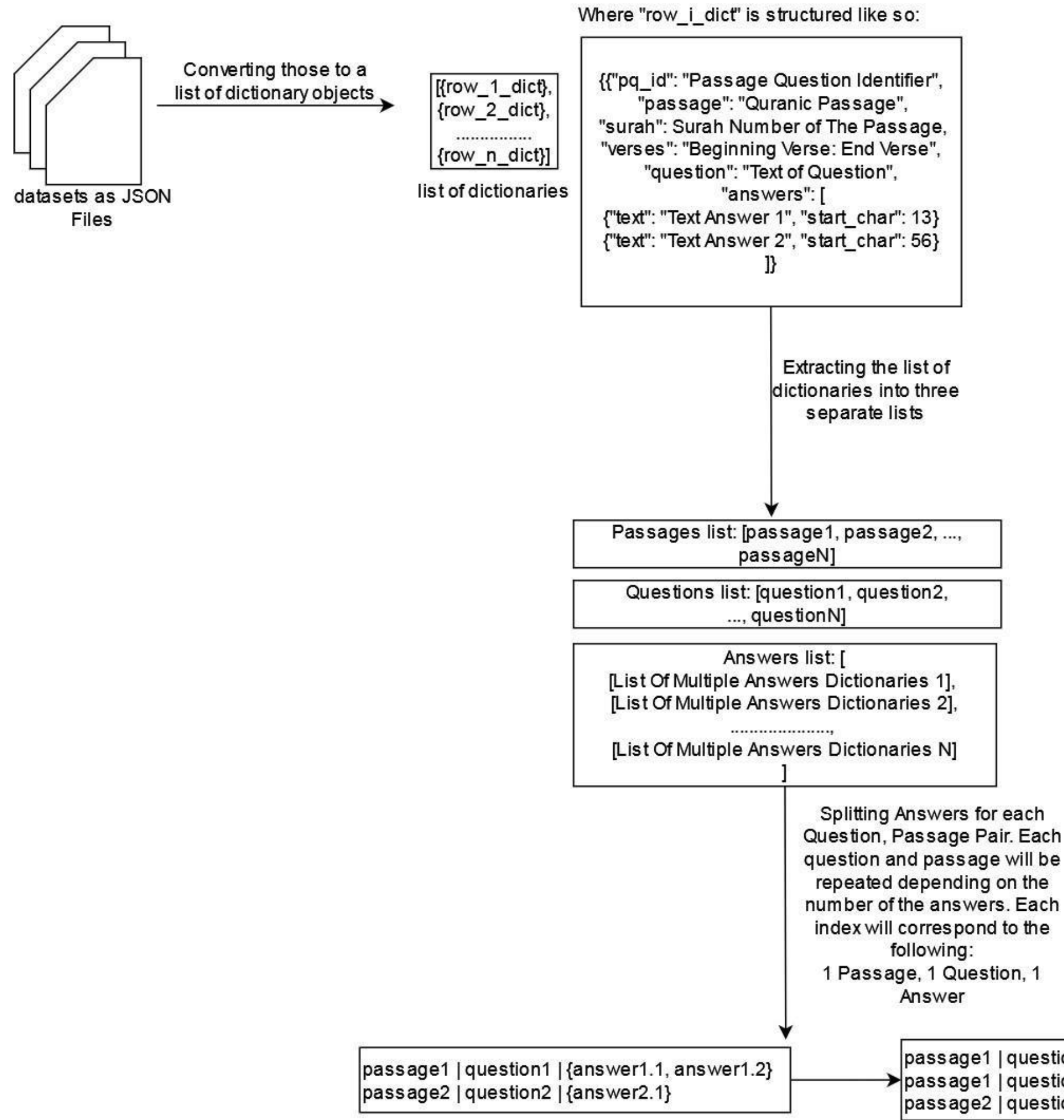
Dataset description:

- The dataset, called QRCD (Qur'anic Reading Comprehension Dataset), comprises 1,093 question-passage pairs and 1,337 question-passage-answer triplets, where each passage contains consecutive verses from the Qur'an and questions are posed in Modern Standard Arabic (MSA).

Steps for preparing the dataset:

- Converts JSON files to a Python dictionaries.
Extract passages, questions, and answers and store them in lists (One question and passage pair can have multiple answers, so we have to split them).
Split multiple answers per passage-question pair. So each row can have one passage, one question, and one answer.

Data Loading & Reformatting Process



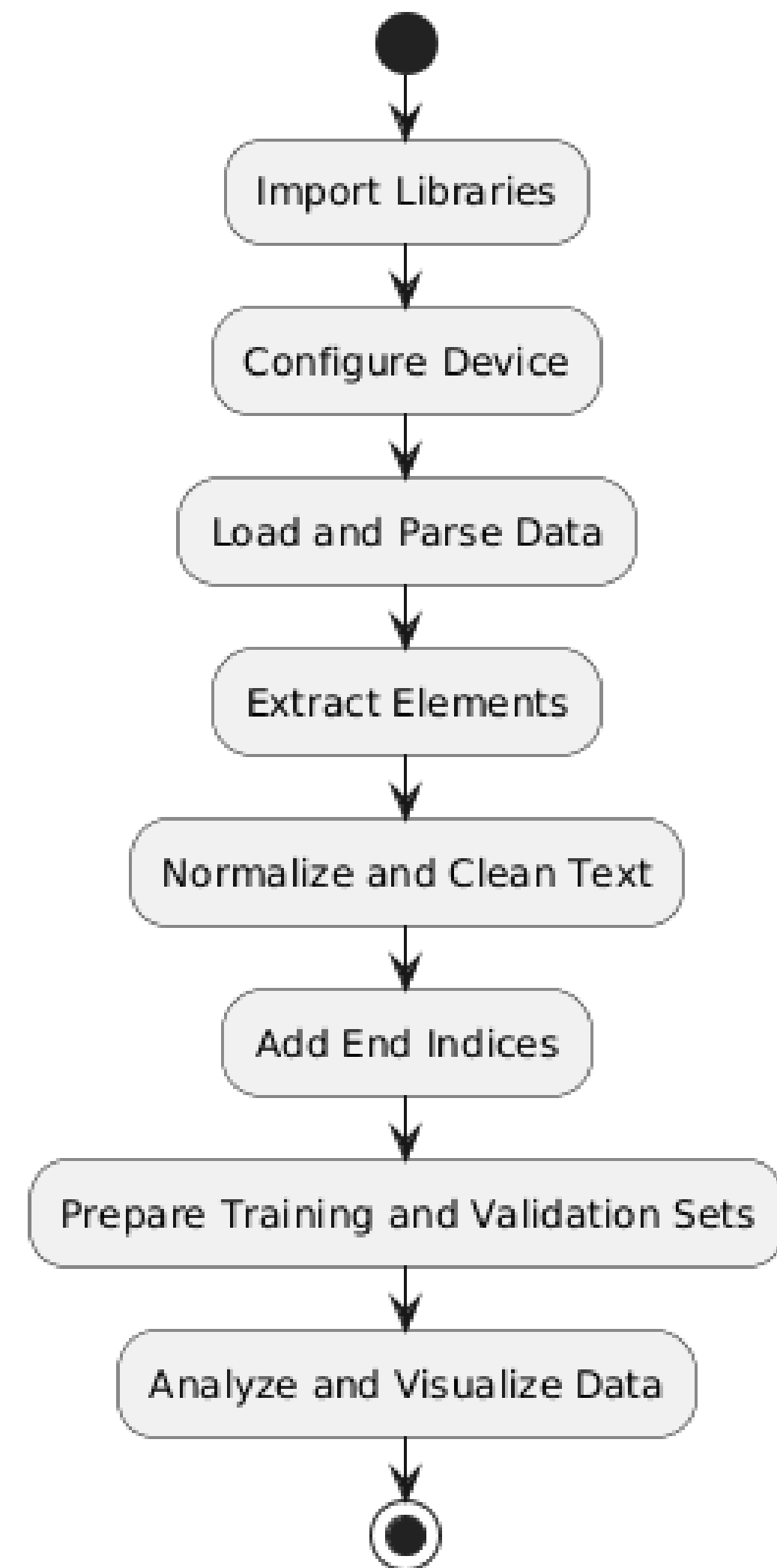
Preprocessing Steps

- **Removing Diacritics**
- **Purpose:** Simplify Arabic text by removing diacritics (Tashkeel) to avoid confusion.
- **Diacritics Removed:**
 - \u064B (ﺀ): Fathatan \u064C (ﻪ): Dammatan \u064D (ﻩ): Kasratan
 - \u064E (ﻪ): Fatha \u064F (ﻩ): Damma \u0650 (ﻩ): Kasra
 - \u0651 (ﻪ): Shadda \u0652 (): Sukun
- **Removing Digits and Punctuation:**

We remove all numbers [0-9] from all three lists in each individual string, and we remove the following punctuations:

! " # \$ % & ' () * + , / : ; < = > @ [\] ^ _ ` { | } ~ » « “ ”

The reason behind this is that we need to reduce the number of useless characters in our text that could potentially degrade the models' performances.



Preprocessing Steps

Adding End Indices to Answers

We need to calculate 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. We also need to ensure these character positions are correct.

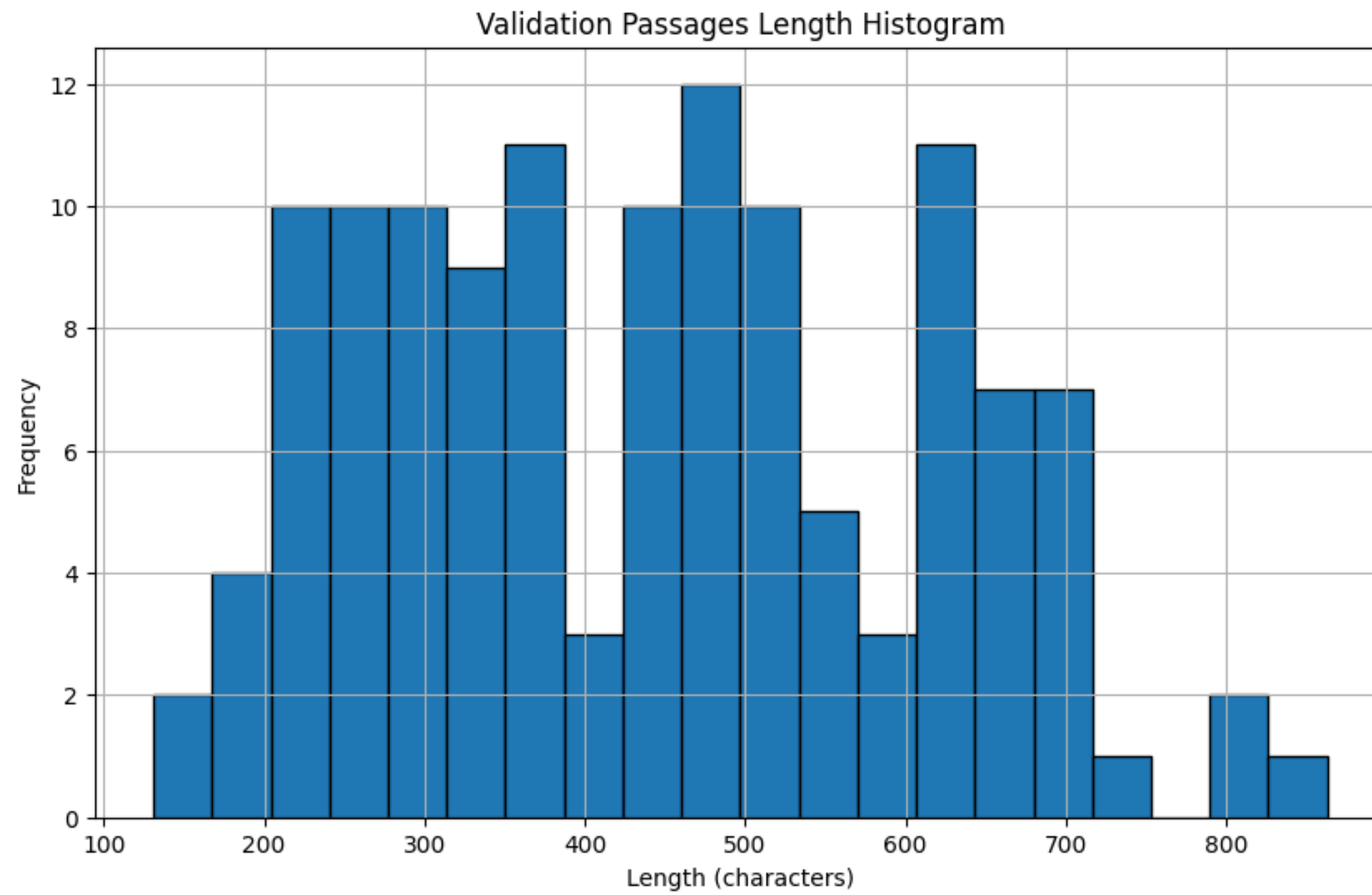
Modifying Text in Answers

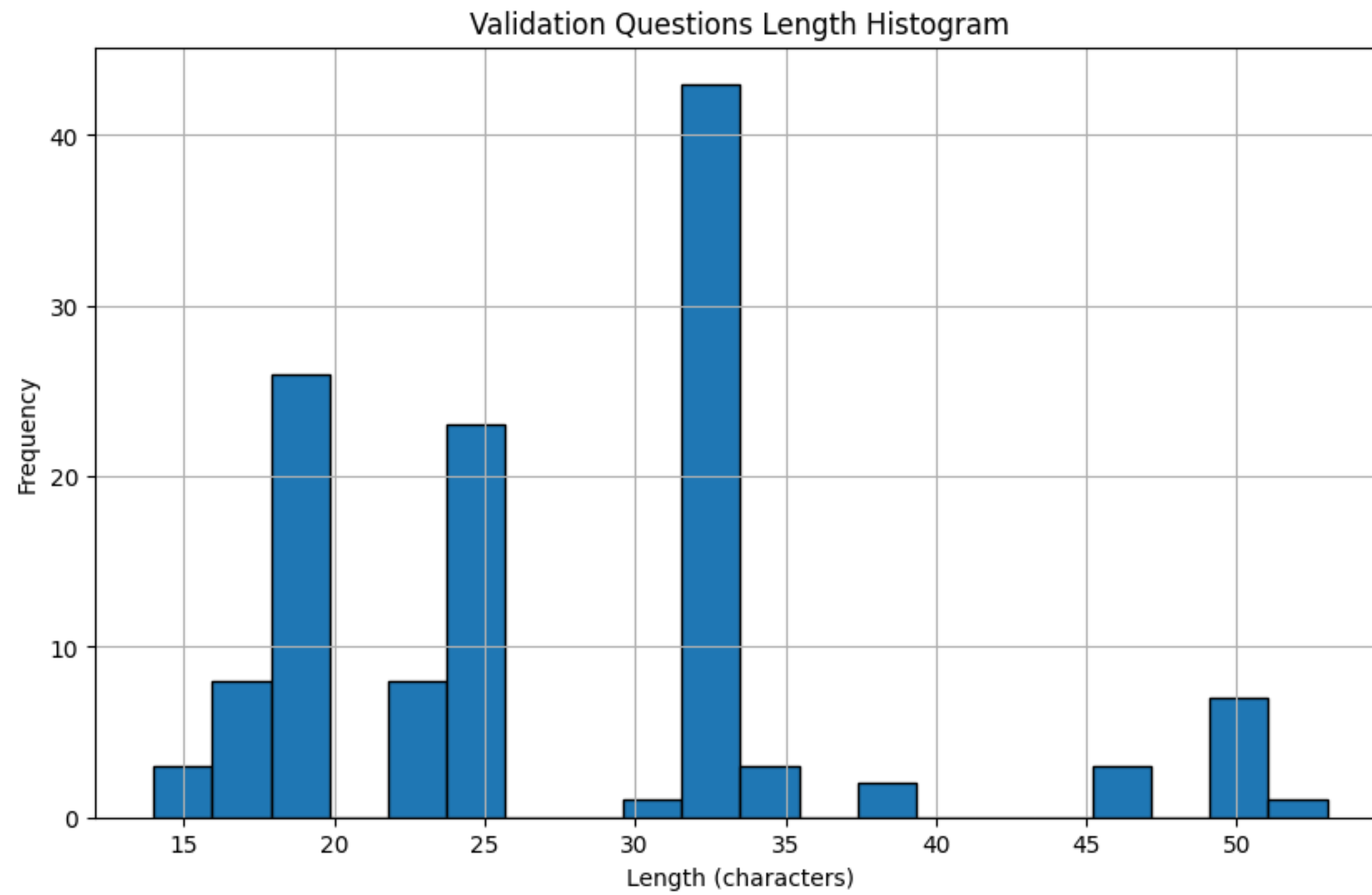
The answers are structured in a way that each item in the answers list is a dictionary of three keys. 'text' Which is the text of the answer, 'start_char' which is the index of the beginning character with respect to the context, 'end_char' is the end of the answer character. This function applies text modification functions (like normalization and punctuation removal) to the answer texts without altering the overall structure of the answer dictionaries.

Dataset Analysis

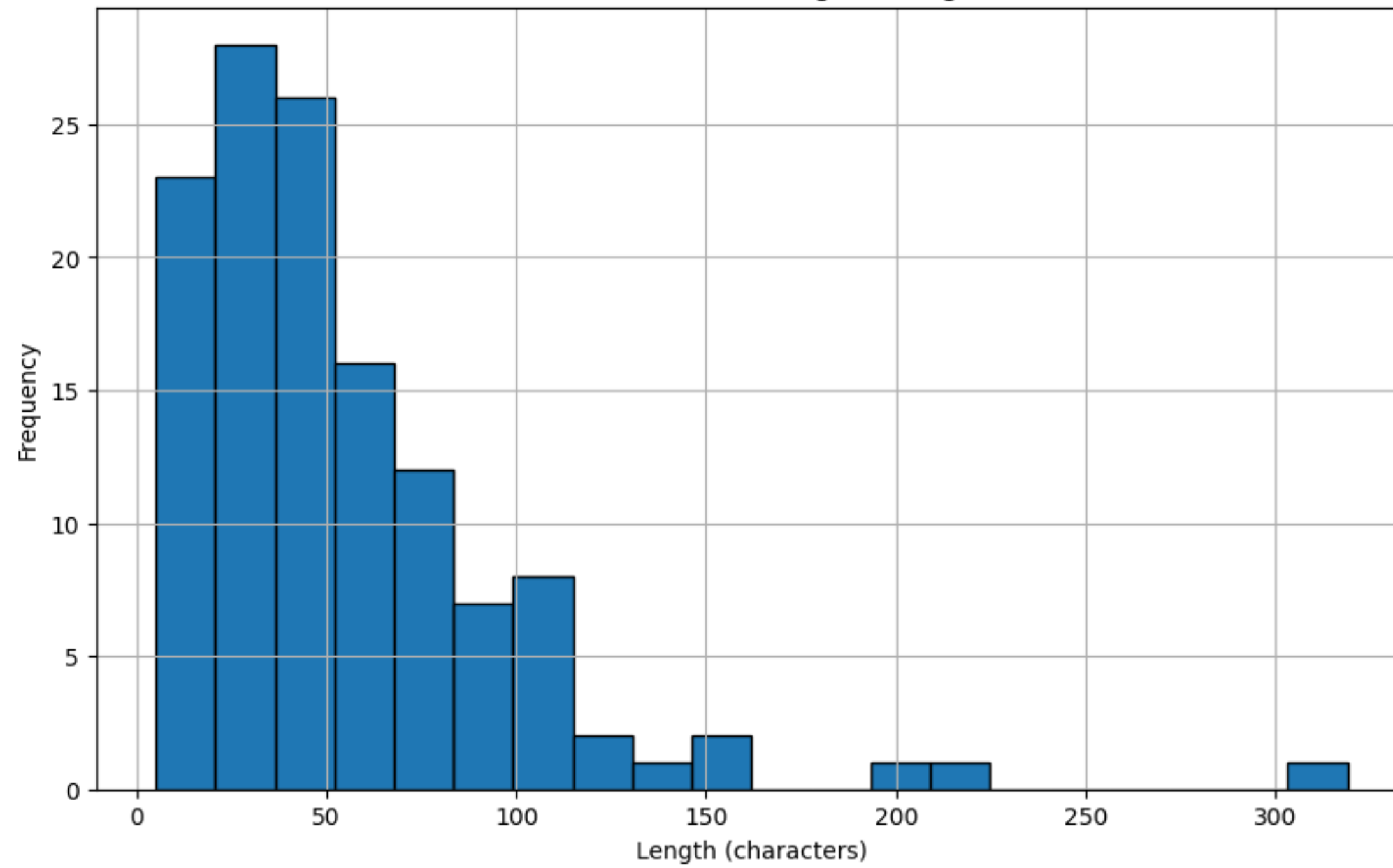
Text Length Statistics & Visualizing Text Length:

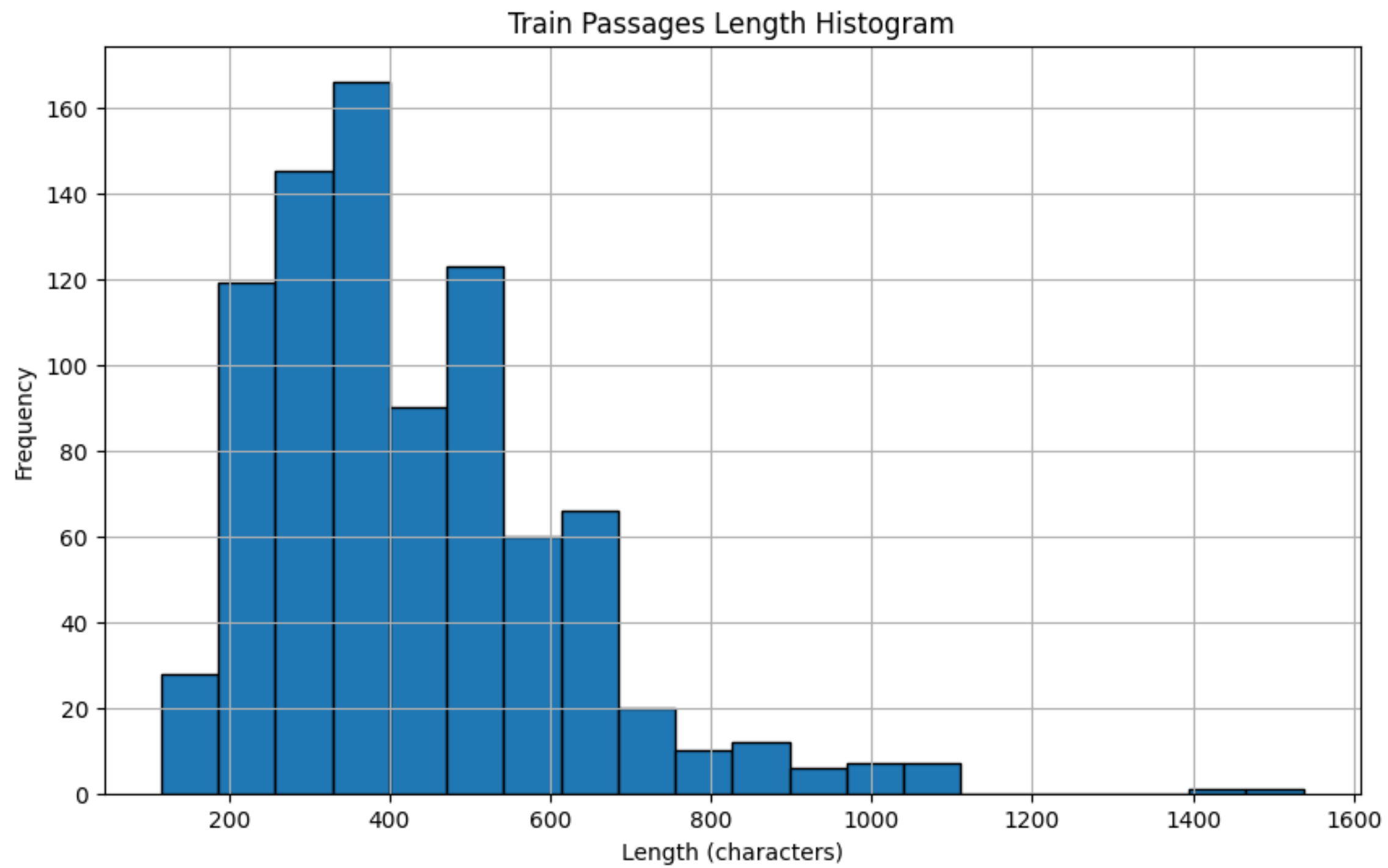
We use various ways to analyze our data in terms of the amount of characters in each of the three lists. These include finding the minimum, maximum, mean, and median of the lengths. The visualization are basically some histogram graphs of those lengths.

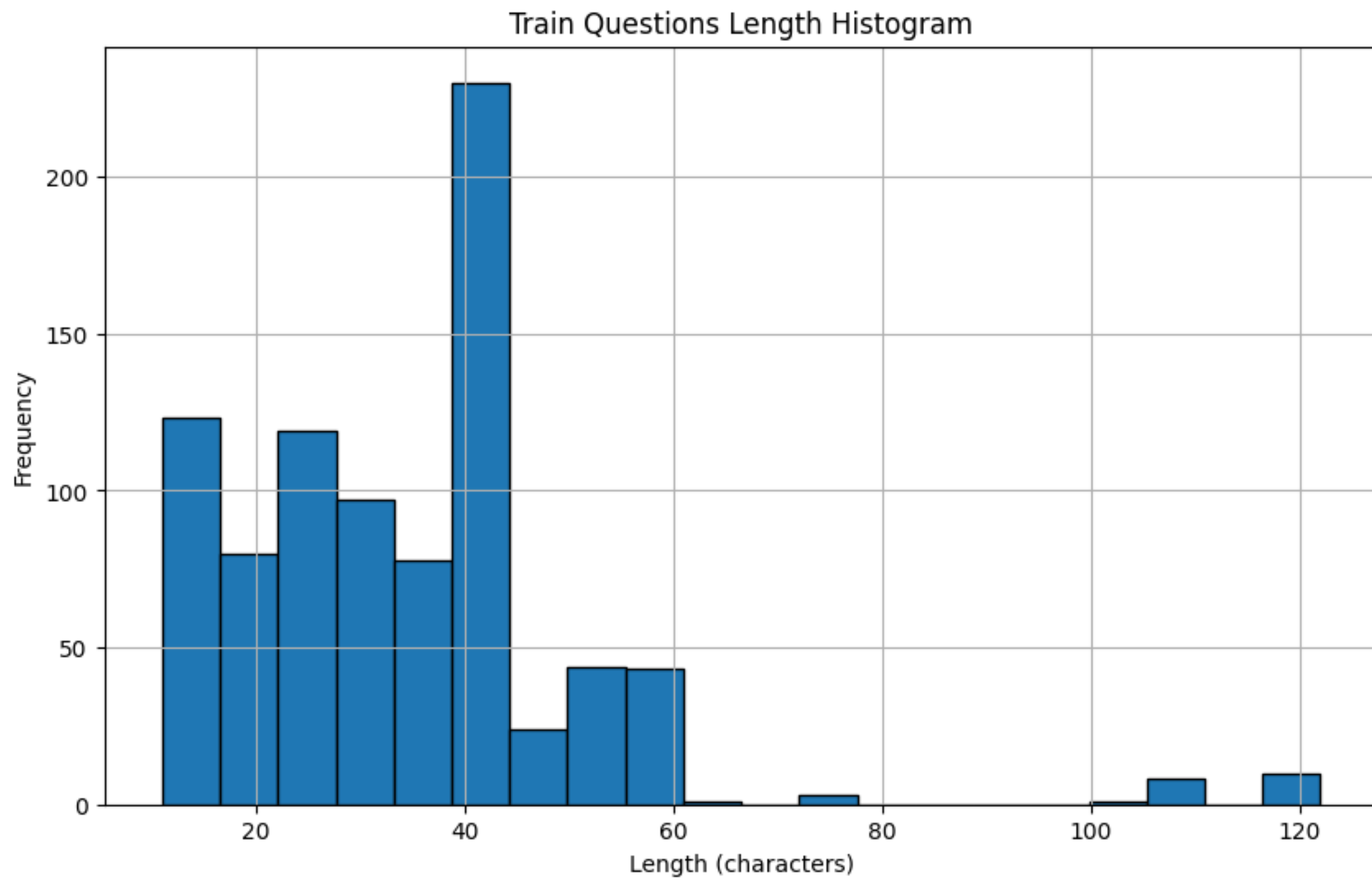


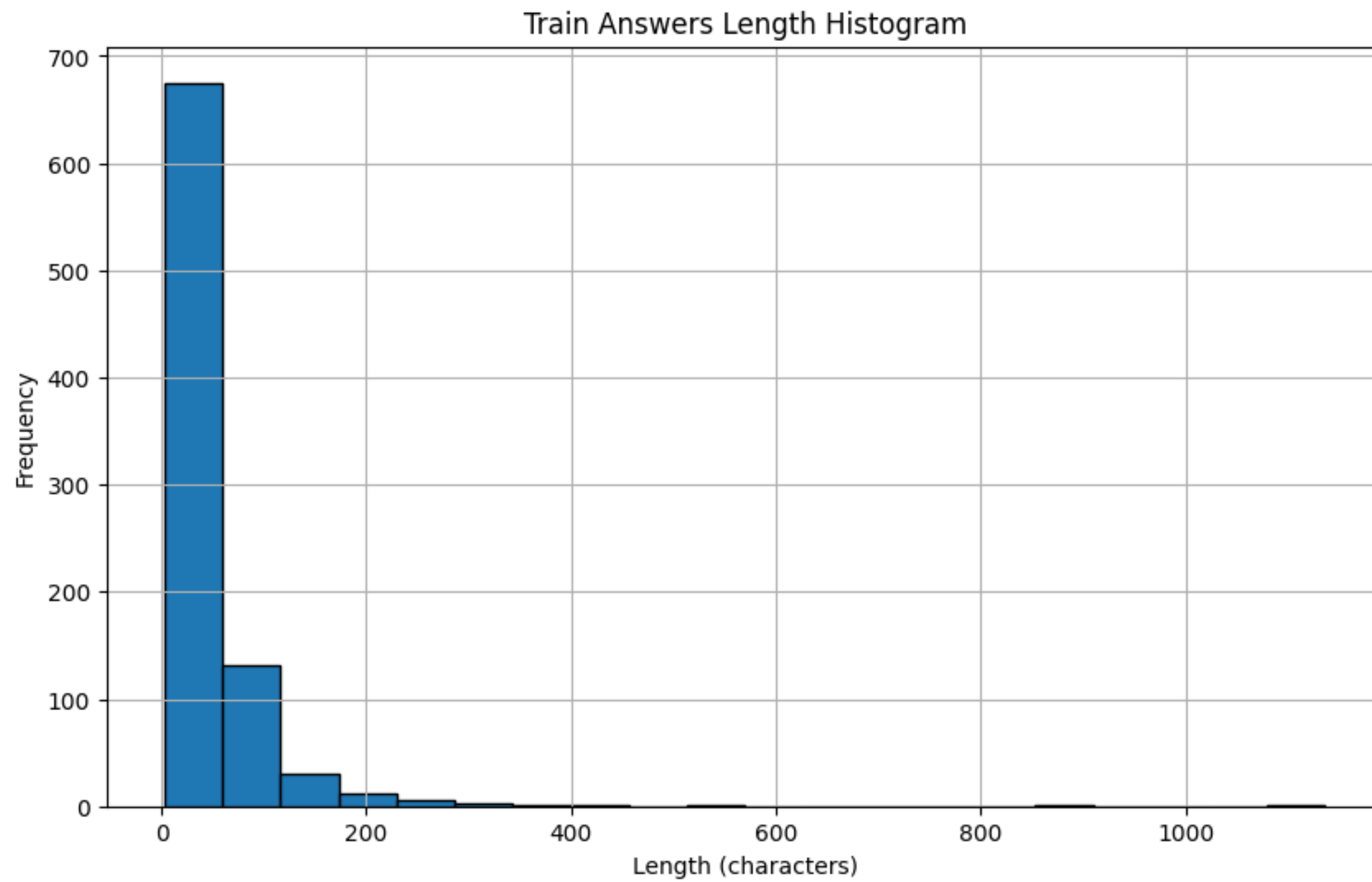


Validation Answers Length Histogram









Training and Validation Data Lengths Analysis

Validation Data Statistics

Stats	Passages	Questions	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

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

Text Tokenization/Encoding

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. Tokenization is needed in order to have a more unified look for the inputs.
- We essentially tokenize both the passages and questions and then append the tokens of both of them with the [SEP] token in between.
- The tokenizer function will return a dictionary of the final tokenization, the input ids form, and their attention mask.

Ex:

token_type_ids: [[CLS], I, love, reading, [SEP], what, do, you, love, [PAD],..., [PAD]]

input_ids: [2, 24, 15, 16, 80, 20, 34, 12, 15, 31, ..., 31]

attention_mask: [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, ..., 0]

Text Tokenization/Encoding

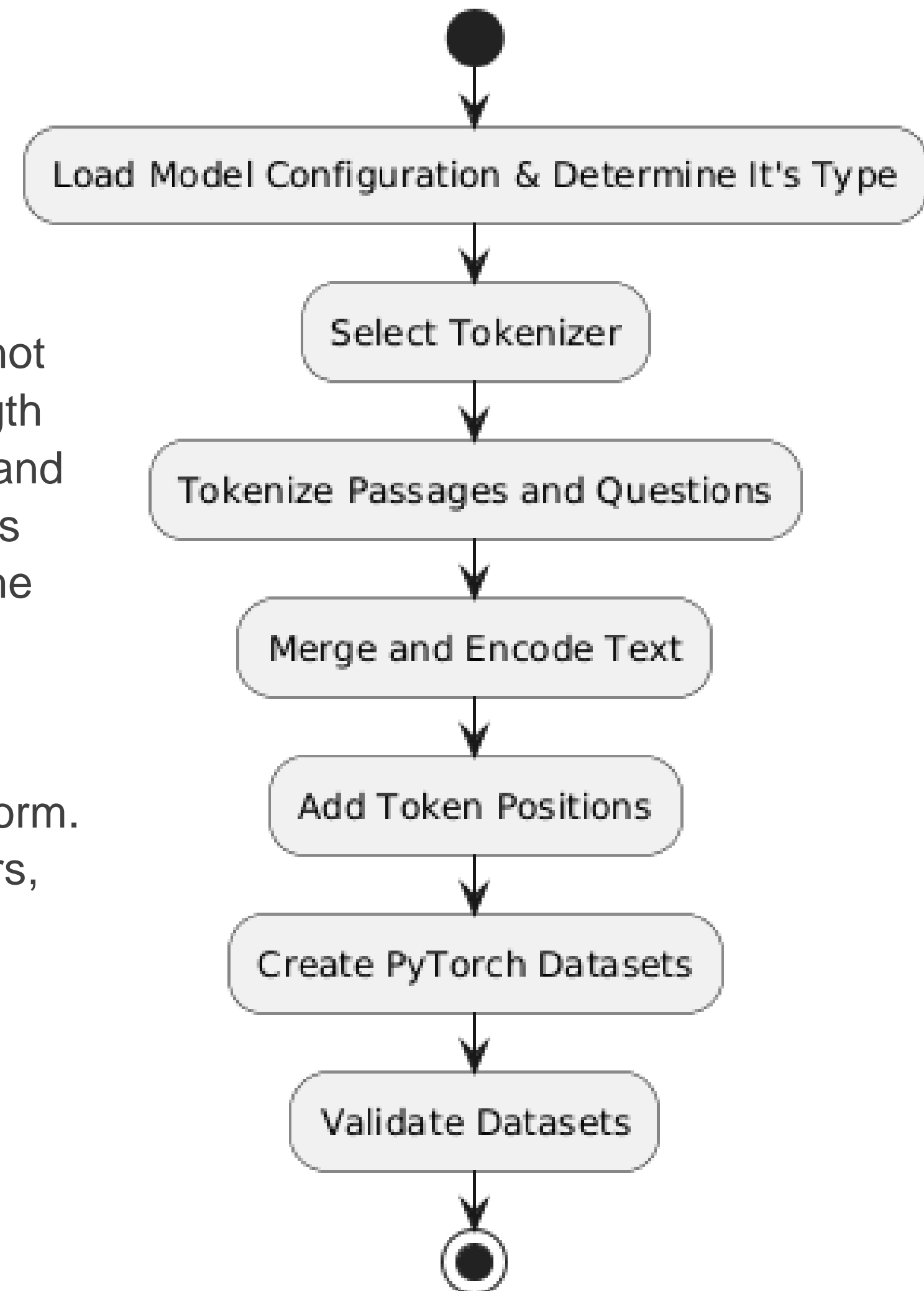
Handling Edge Cases

- 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. We need to convert the start and ending character indices for the answers to token indices. This shall insert those two tokens positions as key and values to the encodings dictionary.

Creating PyTorch Datasets

- We create a custom PyTorch dataset class initializes with encodings and implements methods to return data in tensor form. The tokenized encodings are transformed into PyTorch tensors, making them compatible with PyTorch models for training and evaluation.

Finally, we then integrate all of the encoding process in a single function.



Model Training/Evaluation Pipeline

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:

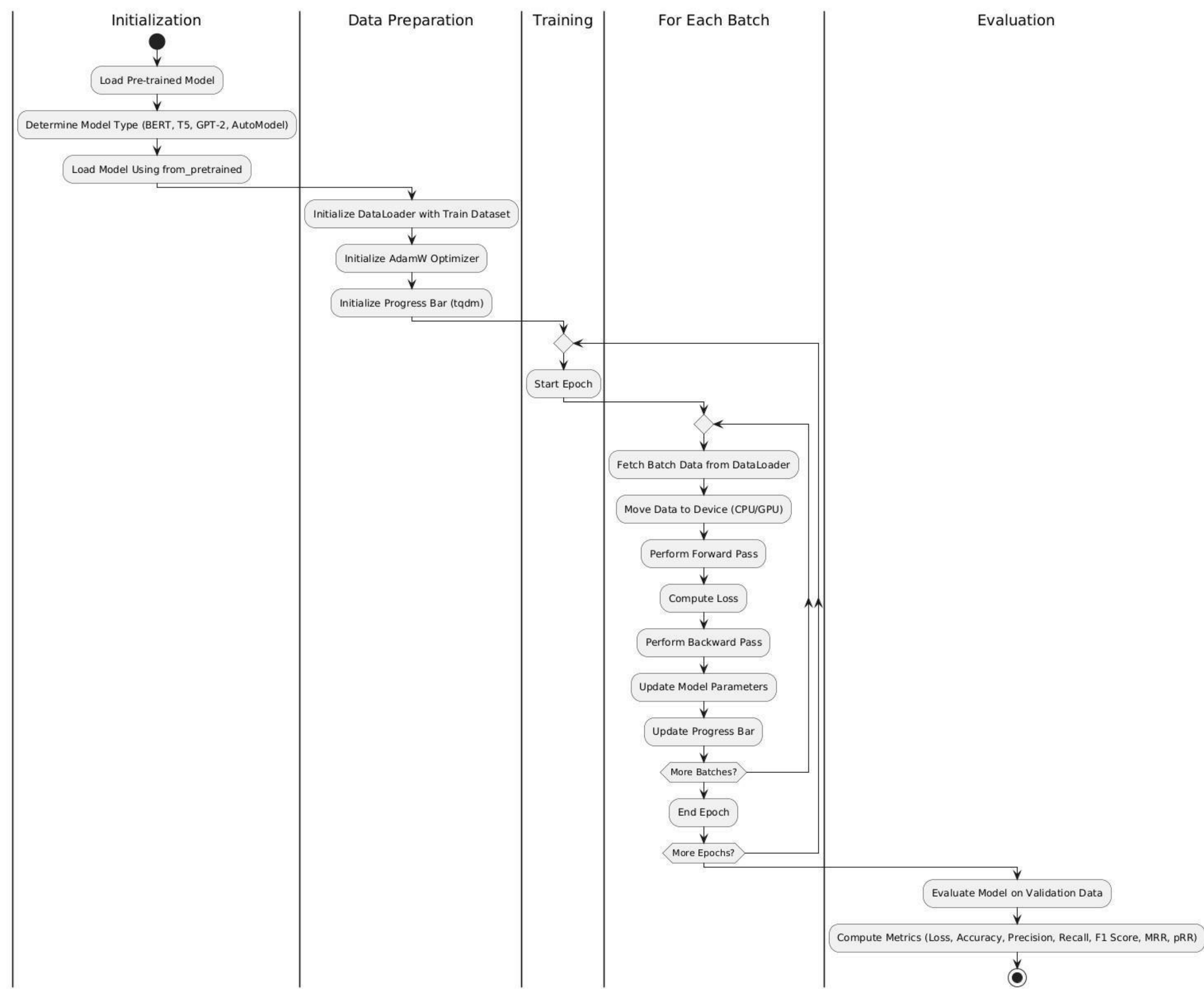
- The process begins with selecting a pre-trained model tailored for question-answering tasks. For this pipeline, several model types are considered: BERT, T5, GPT-2, and a more generic models. Each of these models comes with its own pre-trained weights and architecture, allowing them to be fine-tuned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

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.
 - 2.Optimizer: The AdamW optimizer with weight decay is chosen for its robustness and effectiveness in reducing overfitting.
 - 3.Progress Tracking: To monitor the training progress, the tqdm library is used to provide a visual progress bar.

Model Training/Evaluation Pipeline

- **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.



Training & Evaluation with BERT Models

- **General Overview**
- **BERT:** it's a Bidirectional Encoder based transformer model. The base version is comprised of 12 layers of encoders. every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection.
- **AraBERT:** is a pre-trained language model specifically designed for the Arabic language, based on Google's BERT architecture. AraBERT uses the same configuration as BERT-Base, which includes 12 layers, 768 hidden units, and 12 attention heads. It Primarily focused on Modern Standard Arabic (MSA).
- **MARBERT:** is another advanced Arabic language model, focusing on both Dialectal Arabic (DA) and Modern Standard Arabic (MSA). MARBERT is trained on a large-scale dataset of Arabic tweets (1 billion tweets). MARBERT aims to understand both Dialectal Arabic and Modern Standard Arabic, making it versatile for various Arabic language tasks.

Model Training Approaches

- **Training AraBERT V2 Without Preprocessing**
- **Process:** Convert data, train with default parameters (batch size: 2, learning rate: $2e-5$, epochs: 5).
- We then save the trained model for future use and to reimport for the evaluation and performance measures calculation.
- **Training AraBERT V2 With Preprocessing**
- **Process:** Apply preprocessing to data before training.
- **Evaluating Both Models**
- **Process:** Load models and evaluate on validation sets.
- **Metrics:** Compare validation loss, accuracy, precision, recall, F1 score, MRR.

Performance Metrics Comparison

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

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

Training & Evaluation with MARBERT

- **MARBERT Training**
- **Process:** Similar to AraBERT, convert data, train, and save model and tokenizer.
- **Evaluation:** Print performance metrics on the validation set.

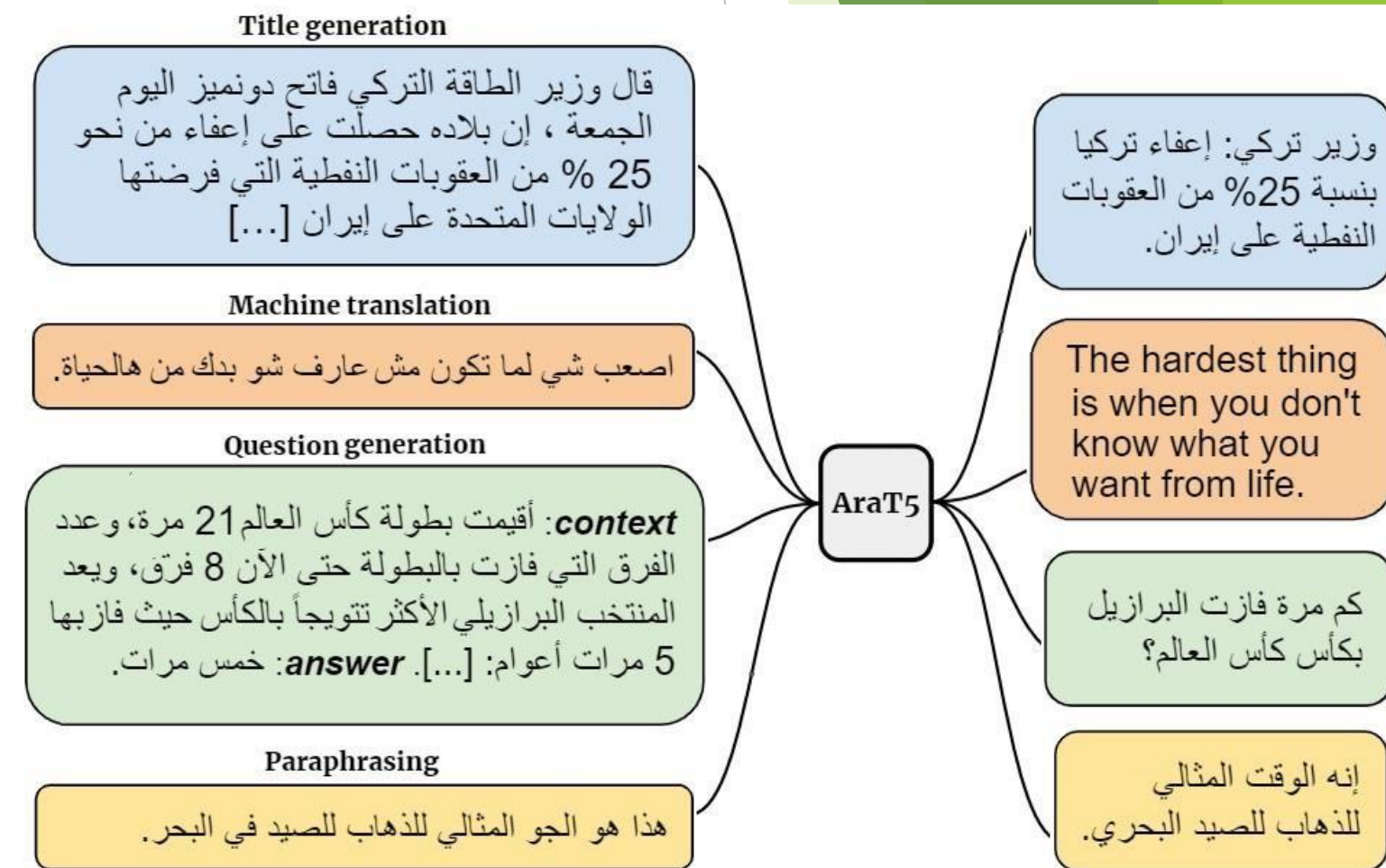
Metric	Value
Validation Loss	3.817220790017018
Validation Accuracy	0.33203125
Validation Precision	0.2471809158445987
Validation Recall	0.26254331528007624
Validation F1 Score	0.23122828578071547
Validation MRR	0.3515625

Hyperparameter Optimization with Grid Search

- Grid Search is the procedure in which we try to find the best hyper parameters, for a given model.
- For integrating the model training process with scikit-learn's API, a custom transformer class is implemented which inherit from BaseEstimator and ClassifierMixin allowing it to fit into the scikit-learn ecosystem.
- Using the GridSearchCV method we can hookup our training loop to it and define our search space like so:
- `param_grid = {'batch_size': [2, 4, 6], 'learning_rate': [2e-5, 1e-4, 2e-4], 'epochs': [2, 5, 10]}`
- Unfortunately, due to the limited budget and processing power we have, we weren't able to get the best hyperparameters or graph them. This thing took forever and the best computer we had crashed after 11 hours during this procedure.

Testing with Another Model: T5

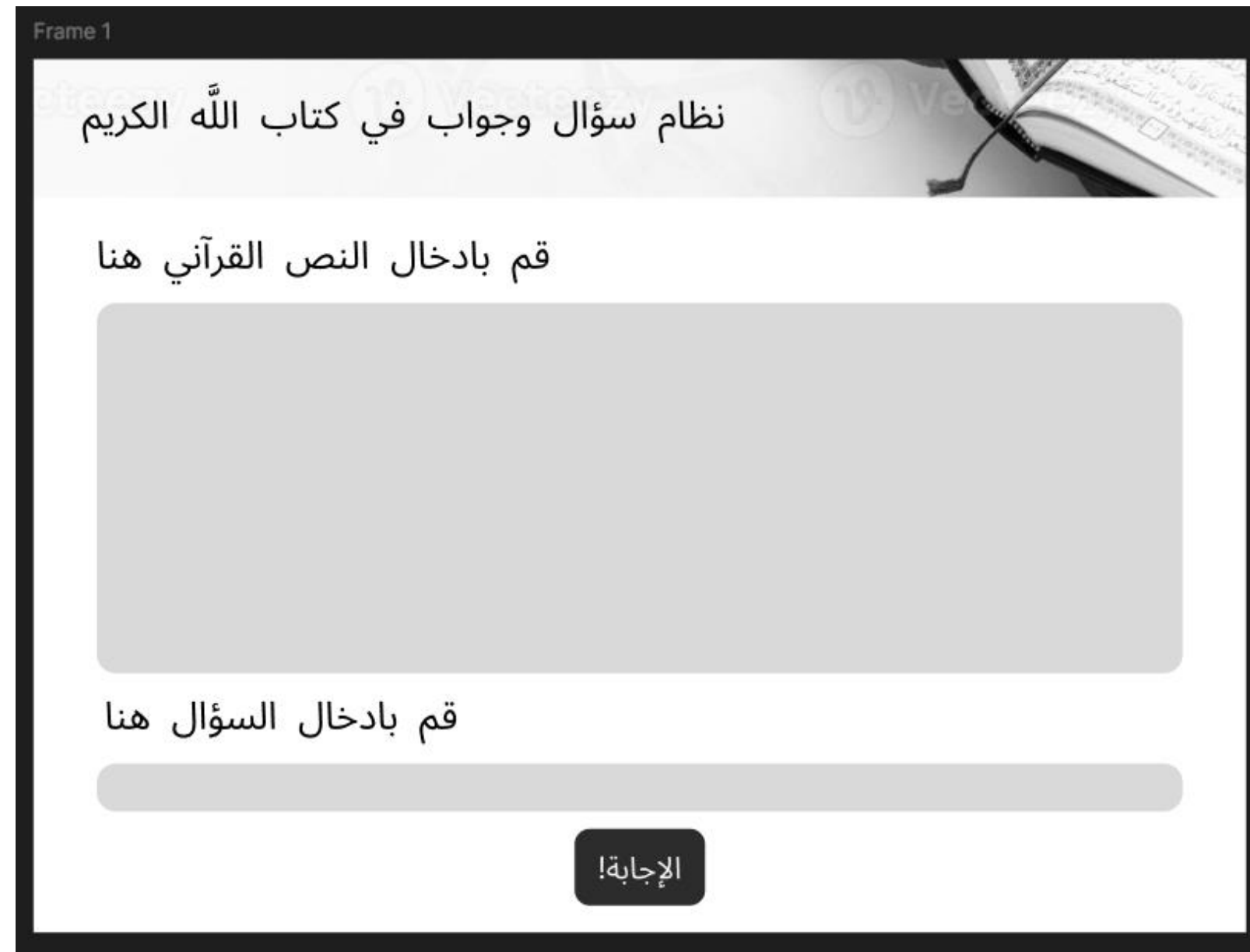
- **AraT5** is a multi tasking model optimized for the Arabic language. It can do Question Answering, Summarization, and translation tasks simultaneously for instance. It is an encoder-decoder model on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format.
- This one required a lot of memory which we didn't have on our computer. And all we could do was to train the model on the validation set to ensure that our pipeline is compatible with the majority of models we throw at it. And training it on Colab will take forever.



Part 2: GUI Application Development

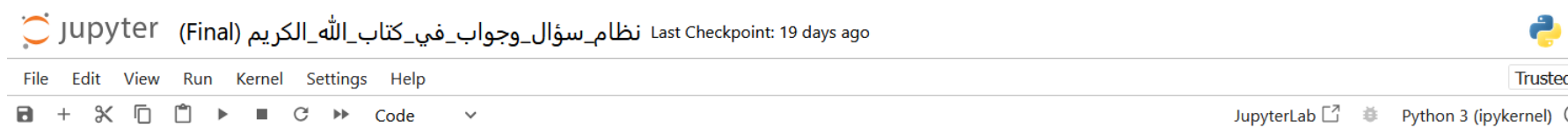
Tkinter GUI Design

- We used an online platform **figma.com** to model the shape of the main window of the program and add the banner picture above as well as defining the button shape, font style, text area and text box.
- Then we convert the design we made online into a Tkinter python code and extract the frontend images (assets) into a separate folder to be loaded using **Tkinter Designer**.



Integrating the QA Code to the GUI

We copied and pasted the code to the main development notebook for editing and refinement as well as integrating both the GUI components and the Question Answering code together. After fully testing the program, we copy the relevant cells to a .PY file to be run standalone and be used for compiling later on.



```
[1]: import torch
from transformers import BertTokenizerFast, BertForQuestionAnswering

def get_answer_from_model(model_path, passage, question):
    tokenizer = BertTokenizerFast.from_pretrained(model_path)
    model = BertForQuestionAnswering.from_pretrained(model_path)
    #Ensure the model is in evaluation mode
    model.eval()

    #Define the device
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
    model.to(device)

    #Tokenize the input context and question
    inputs = tokenizer.encode_plus(question, passage, return_tensors='pt')

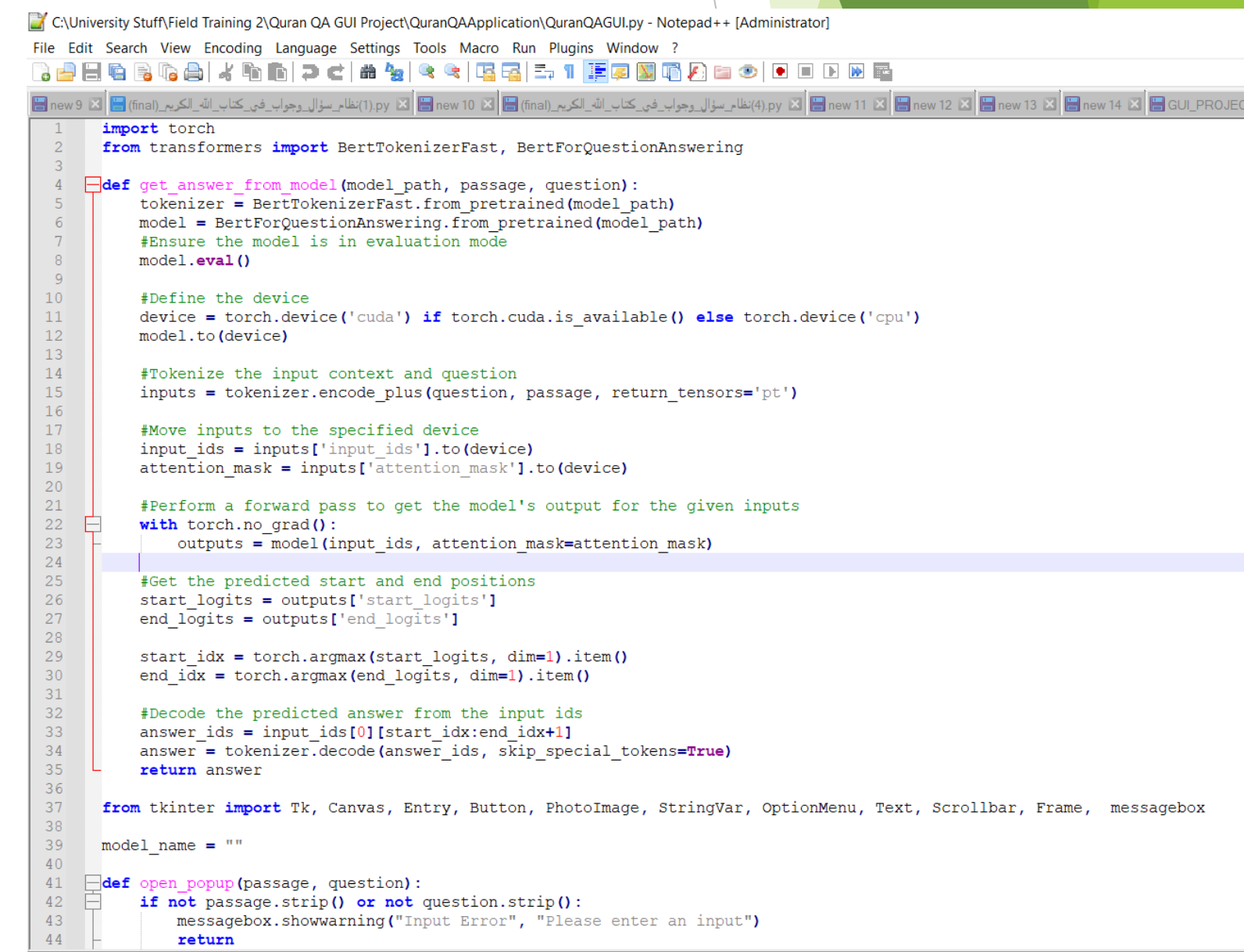
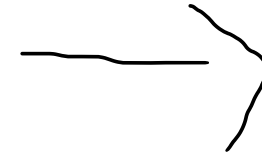
    #Move inputs to the specified device
    input_ids = inputs['input_ids'].to(device)
    attention_mask = inputs['attention_mask'].to(device)

    #Perform a forward pass to get the model's output for the given inputs
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)

    #Get the predicted start and end positions
    start_logits = outputs['start_logits']
    end_logits = outputs['end_logits']

    start_idx = torch.argmax(start_logits, dim=1).item()
    end_idx = torch.argmax(end_logits, dim=1).item()

    #Decode the predicted answer from the input ids
```



Converting the code to a desktop application

- We have included two important folders for the program to function properly, one is called the “model” folder, which contains the AraBERT & MarBERT models pretrained for the Quranic extractive question/answering task.
- The other folder is the “assets” folder which includes some gui graphic elements, we put that to the “_internal” folder for the program to work. There’s no need to install any other software as long as you have visual redistributable c++ 2022 installed on your PC because it contains necessary dependencies for the program to work, otherwise we haven’t encountered any issues on other machines. You can launch the program from **QuranQAGUI.exe**.




- **The Internal folder contains all the libraries, dependencies, and python runtime files.**


```
(quranEQA) PS C:\University Stuff\Field Training 2\Quran QA GUI Project\QuranQAApplication> pyinstaller QuranQAGUI.py --windowed
```









GitHub Repository Creation

We created a GitHub repository to publicly share the source code for the project and let the collaborators easily modify different elements of the project.

 **quranicEQA** Public Pin Unwatch

main 1 Branch 1 Tags t Add file <> Code

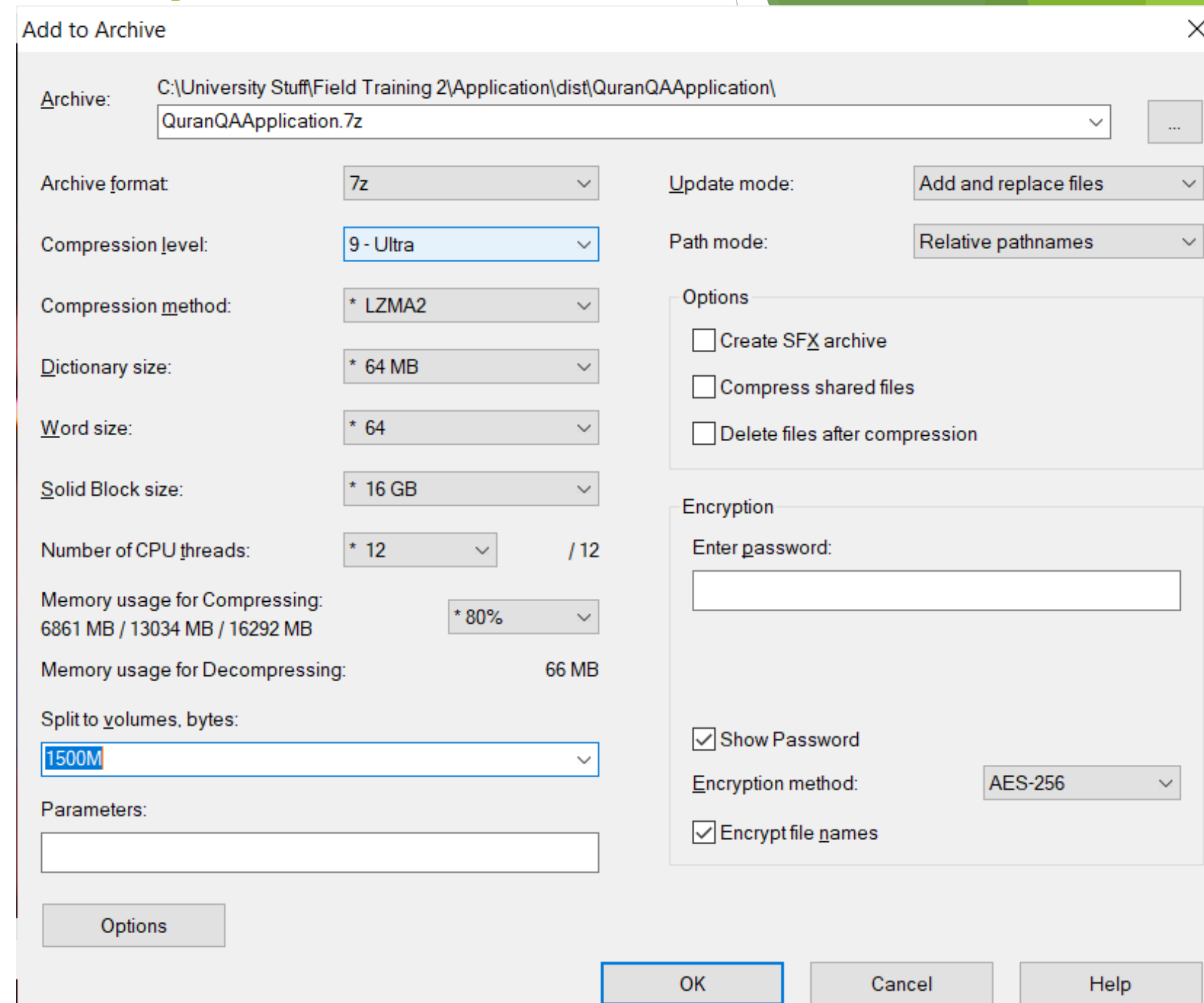
 **AbdelrahmanAbdelkader2055** Add files via upload 79cf0e3 · 2 hours ago 12 Commits

 (Final) نظام_سؤال_وجواب_في_كتاب_الله_الكريم	Add files via upload	5 days ago
 Commentray Demo Video.mp4	Add files via upload	2 hours ago
 LICENSE	Initial commit	5 days ago
 Project Documentation.pdf	Add files via upload	5 days ago
 QuranQAGUI.py	Add files via upload	yesterday
 ReadMe.txt	Update ReadMe.txt	yesterday
 requirements.txt	Add files via upload	yesterday

<https://github.com/AbdelrahmanAbdelkader2055/quranicEQA>

Program size reduction & compression

We have created a separate anaconda development environment for all the necessary libraries to be included to discard the irrelevant ones which we originally had for our development environment. That would end up saving over 1.5 GB of total program space. We then recompiled the code using Pyinstaller library command mentioned “Converting the code to a desktop application”. Finally, we heavily compressed the program folder using an archiver software known as **7zip** and we used those settings and split them to two parts. Reducing the overall file size from 5.35GB to 2.36GB compressed.





```
(quranEQA) PS C:\University Stuff\Field Training 2\Quran QA GUI Project\QuranQAApplication> pyinstaller QuranQAGUI.py --windowed
```





First Release Publication

We uploaded the two parts of the 7zip file to the GitHub repository's releases section so the end user can download, extract and use this program without much hassle. It took us time to upload those two file due to the relatively large file sizes.

v1.0

Latest

Compare  






 AbdelrahmanAbdelkader2055 released this yesterday · [5 commits](#) to main since this release  alpha  0e0d57c 


First Public Release

Added Hot Fix 1 (For Input Handler) [Just Replace the .exe file included]

▼ Assets

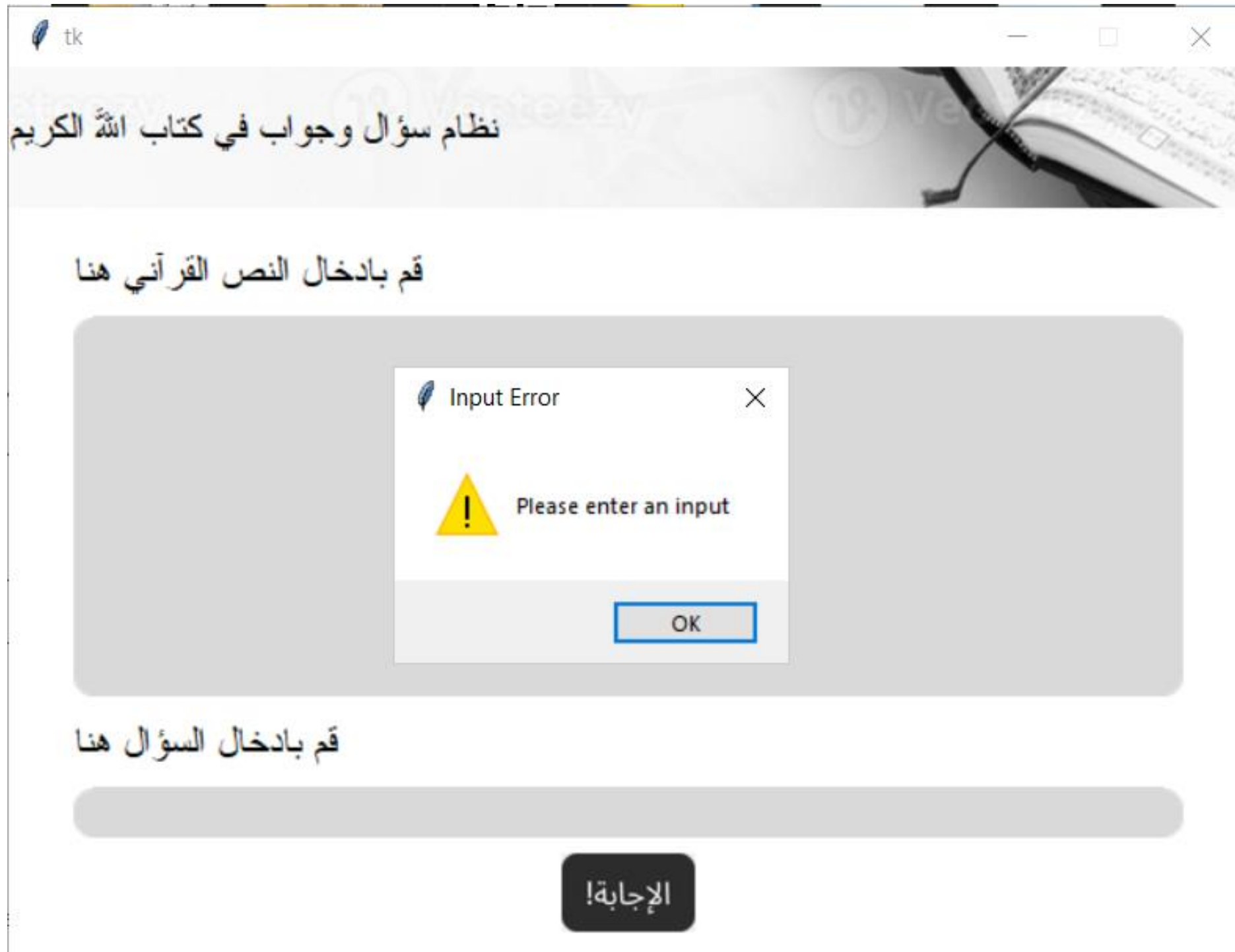
5

 Hotfix1.zip	44.1 MB	yesterday
 QuranQAGUIAPP.7z.001	1.46 GB	yesterday
 QuranQAGUIAPP.7z.002	918 MB	yesterday
 Source code (zip)		yesterday
 Source code (tar.gz)		yesterday



Input Handler Implementation

This feature was added late during development, and it checks for the inputs and see whether they're empty or not. Added as hotfix 1 in GitHub.



Suggested Future Work

- **Advanced Preprocessing Techniques**

- Explore different text normalization and tokenization strategies.
- Better handling of Quranic diacritics.

- **Model Refinement and Complexity Handling**

- Develop or refine models to handle Quranic Arabic complexities.
- Integrate additional NLP functionalities.

- **Expansion and Enhancement of Datasets**

- Expand the Quranic dataset.
- Incorporate additional Arabic text resources and transfer learning techniques.

- **Evaluation and Benchmarking**

- Continual evaluation and benchmarking against existing QA systems.
- Regularly update evaluation metrics specific to Quranic text.

Conclusion

- **Project Team Learning Outcomes**

- **Skills Developed:**

- Advanced NLP models: AraBERT, MARBERT, T5.
 - Machine learning techniques: Model fine-tuning, dataset preparation, performance evaluation.
 - Preparedness for future AI and NLP endeavors.

- **Technological Impact:**

- Improved efficiency and accuracy in information retrieval from religious texts.
 - Enhanced usability and accessibility of religious texts.
 - Contribution to AI by addressing challenges of classical texts.

References:

- 1.(2024, June) wikipedia [online] authors unicode for arabic
[Arabic script in Unicode - Wikipedia](#)
- 2.(2019) [online] Taha Zerrouki
[لوظائف-الولا — 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.

THANK YOU