TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A PROJECT REPORT

ON

KEYPHRASE DETECTION AND QUESTION GENERATION FROM TEXT USING MACHINE LEARNING

SUBMITTED BY:

AAYUSH LAMICHHANE (PUL075BCT005)

BISHAL KATUWAL (PUL075BCT028)

BISHANT BANIYA (PUL075BCT030)

GOBIND PD SAH (PUL075BCT038)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

April 30, 2023

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"Keyphrase Detection and Question Generation from Text using Machine Learning"** submitted by **Aayush Lamichhane**, **Bishal Katuwal**, **Bishant Baniya**, **Gobind Prasad Sah** in partial fulfillment of the requirements for the Bachelor's degree in Electronics & Computer Engineering.

....
Supervisor
Santosh Giri
Assistant Professor
Department of Electronics and Computer
Engineering,
Pulchowk Campus, IOE, TU.

Copyright

Acknowledgments

We would like to express our gratitude to everyone who helped us to complete this project. First and foremost, we would like to acknowledge the crucial role of our project supervisor, **Er. Santosh Giri**, Associate Professor, Department of Electronics and Computer Engineering for their guidance, support, and feedback throughout the project. Next, we would like to give our gratitude to our classmates, for providing constructive feedback and engaging in thought-provoking discussions regarding our project. Next, we'd like to thank all lecturers of our department for guiding us through the beginning of our project to the end. Finally, a special gratitude to our family and friends, for their love, encouragement, and support throughout our academic journey.

Thank you all for your invaluable contributions to this project.

Sincerely,
Aayush Lamichhane
Bishal Katuwal
Bishant Baniya
Gobind Prasad Sah

Abstract

Question Generation may not be as prominent as Question Answering but it still remains a relevant task in NLP. The ability to ask meaningful questions provides evidence towards comprehension within an Artificial Intelligence (AI) model. This makes the task of question generation important in the bigger picture of AI. While existing question generation techniques rely on complex model architectures and additional mechanisms to boost performance, we show that transformer-based fine-tuning techniques can create robust question generating systems using only a single language model,, without the use of additional mechanisms, answer metadata, and extensive features. Some training parameters of our project are : epoch :10, batchsize : 4, learning rate 10e-3.Lastly, we also look into the model's failure modes and identify possible reasons why the model fails.

Keywords: *: Question Generation, Transformer, Automation, Question Generation, Randomization*

Contents

List of Figures

List of Tables

List of Abbreviations

NLP	Natural Language Proccessing
QA	Question Answering
QG	Question Generation
AI	Artificial Intelligence
ML	Machine Learning
BERT	Bidirectional Encoder Representations from Transformers.
T5	Text-to-Text Transfer Transformer

1. Introduction

In recent times, we have increased the interest in automated systems. One of the key field of automated system is use of Natural Language Processing(NLP) to understand and manipulate natural language text. Although, the biggest field in NLP is text summarization and question answering(QA), another equally important field is Question Generation(QG). Question generation (QG) aims to create natural questions from a given sentence, paragraph or a file. NLP QG is mostly applicable in education, chatbots, information retrieval, etc. Although, NLP QG, in itself, may not have a wide range of application, but it plays a vital role in NLP field as a whole. It is an important application of NLP because it allows computers to better understand and interact with human language, improving language comprehension, summarization, information retrieval, and dialogue systems. This project mostly deals with Question Generation in academics.

1.1 Background

In current education settings, question generation in academics has largely been limited to standardized tests with manual generation of questions. This leads the question generation tasks to be tiresome for both question setters and students. For question setters, it is an tiring task to generate appropriate question from a large valume of text with strict guidelines. For students, these methods often fail to capture the full range of a student's knowledge as a small set of setters are bound of set similar questions and thus questions can be repetitive. NLP QG can replace these processes by generating questions that can be used for formative assessment. These questions can be automated with NLP and tailored to individual needs. It can also be used to create personalized learning experiences for students by generating questions that are tailored to a student's individual needs based on their input text and question type. It can help ensure that students are being challenged appropriately and are not getting bored or frustrated with repetitive content.

1.2 Problem statements

Traditional methods of assessment in education, such as standardized tests and written assignments, often fail to capture the full range of a student's knowledge. There is a need for more diverse and personalized assessment methods that can help students identify areas where they need additional effort and adjust their learning accordingly. Natural Language Processing Question Generation (NLP QG) has the potential to address these issues by generating questions that are tailored to a student's level of understanding.

1.3 Objectives

- To implement an AI based question generator which generates both subjective and objective question along with answers.

1.4 Scope

The scope of this project is limited to academic application of NLP QG. Thus the scope extends to:

- It provdies an automatic, reliable and unbiased questions for evaluation.
- It helps students in self-evaluation.
- It separates the key theme of any excerpt.
- It aids in note keeping.

2. Literature Review

In this literature review, we will explore the various techniques used in NLP QG and their applications in the academic domain. Several techniques have been used in NLP QG, including rule-based, template-based, and machine learning-based approaches. Rule-based approaches involve using predefined rules to generate questions from given texts. Template-based approaches use predefined question templates to generate questions from given texts. Machine learning-based approaches, on the other hand, involve training models using large datasets to generate questions from given texts. NLP QG has gained significant attention in the field of education and academics in recent years. Many researchers have explored the potential of NLP QG for various educational purposes, such as formative assessment, personalized learning, and curriculum development.

2.1 Related work

Liu and Calvo (2012) proposed a system for generating questions to support academic writing using Wikipedia and conceptual graph structures.[1] The system was shown to be effective in generating relevant and useful questions for academic writing support. The authors constructed conceptual graphs for each sentence in the academic texts using a graph-based NLP tool called Text2Onto. They then used the conceptual graphs to generate questions by applying a set of graph transformation rules that convert the graphs into questions.

In another study, Zhao et al. (2018) proposed an adaptive question generation system that can generate questions of varying difficulty levels based on the student's prior knowledge and level of understanding.[2]. The study involved constructing a knowledge graph from the educational materials and domain-specific ontologies. It also involved implementing learning progress analysis algorithms, and developing adaptive question generation algorithms that utilized the knowledge graph and learning progress analysis results. The system was shown to improve student learning outcomes and engagement.

Other studies have also explored the use of NLP QG for curriculum development. For instance, Li et al. (2020) proposed a system for generating domain-specific questions based on textbooks and a knowledge graph.[3] The system was shown to be effective in generating high-quality questions that aligned with the curriculum. The system had the ability to generate domain-specific questions that are relevant to the content of the textbook and knowledge graph.

Although, these methods show promise in generating questions to support academic writing, there are limitations that need to be addressed to improve the quality and flexibility of the generated questions.

Liu and Calvo (2012)'s Text2Onto has issues in performance and scalability. Also, the focus has shifted from generic ontology models to task specific models.

Zhao et. al (2018) was heavily on Knowledge Graph Construction with isn't always feasible. It had limited generalization and was able to generate questions outside of the domain or topics covered by the predefined ontologies and templates.

Similarly, Li et. al (2020) also has some limitations, such as the potential bias introduced by the pre-defined question templates and the lack of flexibility to generate wide range of questions. Knowledge graph is based on entity relationship and thus cannot ask open ended question(How/Why).

Thus, in current scenario, transformer based question generation model is best suited. One example of a transformer-based question generation (QG) model is "Transformer-based Question Generation with Self-Supervised Learning" by Dai et al.[4]. The paper proposes a transformer-based QG model that uses self-supervised learning to improve the quality of generated questions. The model is trained on a large corpus of text data using a masked language modeling objective, which encourages the model to learn to predict missing words in sentences. The authors show that the model is able to generate high-quality questions for a variety of text genres and domains. One limitation of this approach is that it requires a large amount of text data to train the model effectively. Additionally, the model may generate questions that are too similar to the input text, which can be problematic in some contexts.

2.2 Related theory

2.2.1 Different approaches of QG

There are several approaches to QG in natural language processing (NLP). One of the approach is rule-based approach, which involves developing sets of linguistic rules to generate questions based on specific patterns or structures in the input text. Another approach is template-based approach, which involves predefining question templates and filling them in with relevant information from the input text. The third approach is Machine learning-based approach, which includes learning. Recently, transformer-based models such as T5 have shown promising results in QG by using the power of self-attention mechanisms and pre-training on large amounts of text data.

Rule based approach

The rule-based approach to question generation (QG) involves developing sets of linguistic rules or patterns to generate questions from input text. These rules can be based on both syntactic and semantic structures of the text. For example, a simple rule for generating questions could be to identify a declarative sentence and transform it into an interrogative sentence by reversing the subject-verb order and adding a question word such as "what," "who," or "when".

```
Text : ' This is a car.'
Generated Question : 'Is this a car?'
```

More complex rules can be developed to handle more nuanced aspects of the input text, such as identifying implicit information, handling word sense disambiguation, and generating appropriate question types based on the discourse context. Rule-based approaches can be limited by the complexity of the rules and the difficulty of encoding all possible patterns in the input text.

Template based approach

The template-based approach to QG involves creating a set of predefined question templates that can be applied to input text to generate questions. The templates can be customized to suit different types of texts and domains, and can include variables that can be filled in based on the input text.

Consider the following input text:

```
Muna Madan is a book by Laxmi Prasad Devkota famous for its tragic ending.
```

A template-based approach might use a "what" question template to generate the following question.

```
What is the Muna Madan?
```

A "what" template can also be designed to extract the following question:

```
What is a book by Laxmi Prasad Devkota famour for its tragic ending?
```

Template-based approach is more efficient and flexible than rule-based approaches. It can also generate questions that are syntactically and semantically correct. However, it can be limited by the number and diversity of templates available, and may not be able to generate questions that require more complex reasoning or understanding of the input text. For example, it is near impossible to create a template to generate following question.

```
Why is Muna Madan famous?
```

Machine Learning based

The machine learning approach to QG involves training the model on a large dataset of question-answer pairs, and it learns to generate questions by identifying patterns and features in the input data. For example, one of the popular algorithms used in ML-based QG systems is Seq2Seq model based on Encoder-Decoder architecture to generate questions from answers. The Encoder takes the input answer and produces a hidden representation of it. The Decoder takes the hidden representation and generates the corresponding question.

For example, let's say we have a QG model that is trained on a dataset of movie reviews. The model is given an input sentence

```
The acting in the movie was superb.
```

It generates a question,

```
Was the acting in the movie good?
```

The model has learned from the training data that phrases like

```
Was [aspect] [quality]?
```

2.2.2 Machine Learning

Machine learning is a subfield of artificial intelligence that involves training computer systems to learn patterns in data and make predictions or decisions based on that learning. Machine learning algorithms can be categorized into three broad categories: supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning

Supervised learning involves training a model to predict a target variable based on input data and a set of labeled examples. This type of learning is commonly used for tasks like image classification, speech recognition, and natural language processing.

Unsupervised Learning

Unsupervised learning involves training a model to find patterns in unlabeled data without explicit guidance from a target variable. This type of learning is commonly used for tasks like clustering, anomaly detection, and dimensionality reduction.

Semi supervised Learning

Reinforcement learning involves training a model to make decisions based on feedback from an environment, with the goal of maximizing a reward signal. This type of learning is commonly used for tasks like game playing and robotics.

2.2.3 Transformers

Transformers are a type of neural network architecture that has become increasingly popular in natural language processing (NLP) tasks. The key innovation of the transformer architecture is the use of self-attention mechanisms to process input data. Self-attention allows the model to weigh the importance of different parts of the input sequence when making predictions, which can be especially useful for NLP tasks where the meaning of a sentence can depend on the context in which it appears.

Transformers are made up of multiple layers of self-attention and feedforward neural networks, with residual connections and layer normalization to improve performance and reduce overfitting. They can be trained using large amounts of data and fine-tuned on specific tasks using transfer learning techniques.

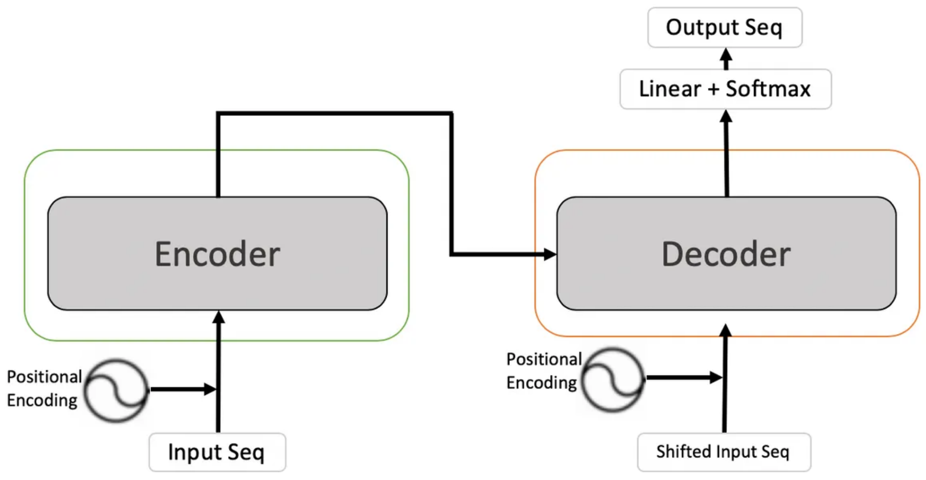


Figure 2.1: High level transformer architecture

The core mathematical operation in the transformer architecture is the self-attention mechanism, which can be expressed as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

where Q, K, and V are matrices representing queries, keys, and values, respectively. K^T represents transpose of Key, d_k is the dimensionality of the key vectors. The attention mechanism calculates a weighted sum of the values based on the similarity between the queries and keys, with the softmax function ensuring that the weights add up to one. This operation can be computed in parallel for all positions in the input sequence, allowing for efficient processing of long sequences.

2.2.4 Attention

Attention is a mechanism used in neural network architectures that allows the model to selectively focus on different parts of the input data when making predictions. Attention was first introduced in a 2014 paper by Bahdanau et al. for machine translation tasks, and has since been applied to a wide range of natural language processing (NLP) tasks and other domains. The core idea of attention is to compute a set of attention weights that reflect the importance of different parts of the input data for the current prediction. These weights are then used to compute a weighted sum of the input data, which is fed into the rest of the neural network architecture. By focusing on the most relevant parts of the input data for each prediction, attention can improve the performance of neural network models on complex tasks like machine translation and text classification.

Equation: The attention mechanism can be expressed mathematically as:

$$a_i = softmax(e_i) \tag{2.2}$$

$$c = sum(a_i * h_i) \tag{2.3}$$

where a_i is the attention weight for the i-th element of the input sequence, e_i is a score calculated based on the current state of the model and the i$^{th} element,$ h$_i$ is the hidden state of the i^{th} element, and c is the context vector, which is a weighted sum of the input sequence. The softmax function is used to ensure that the attention weights add up to one.

2.2.5 Softmax

Softmax is a mathematical function that is commonly used in machine learning, which takes a vector of real numbers as input and normalizes it into a probability distribution, such that the output values are between 0 and 1 and sum up to 1. The softmax function can be defined mathematically as follows:

$$softmax(x_i) = exp(x_i)/sum(exp(x_j)) \tag{2.4}$$

where x_i is the i^{th} element of the input vector x, and the sum is taken over all elements of the vector. The exponentiation and normalization operations ensure that the output values are positive and sum up to 1.

2.2.6 Encoder Architecture

In the context of neural networks, an encoder is a type of architecture that takes input data and transforms it into a lower-dimensional representation that can be used for downstream tasks like classification, clustering, or generation.

One popular type of encoder architecture is the convolutional neural network (CNN), which typically consists of several convolutional layers followed by pooling layers and a fully connected layer that produces the encoded representation. Another type of encoder architecture is the recurrent neural network (RNN), which processes the input sequence one element at a time, and uses a hidden state to maintain a memory of the previous elements in the sequence. The final hidden state of the RNN can be used as the encoded representation.

A more recent and highly popular encoder architecture is the Transformer, which was introduced in a 2017 paper by Vaswani et al.[5] The Transformer is a self-attention based neural network architecture that has achieved state-of-the-art results on a wide range of natural language processing tasks. The Transformer encoder consists of several self-attention layers followed by feedforward layers that produce the encoded representation. .

2.2.7 Decoder Architecture

The decoder architecture is typically used for tasks such as image or speech generation, language translation, or text generation. It takes the encoded representation produced by the encoder as input and produces an output that is similar to the original input data.

2.2.8 Encoder-Decoder Architecture

The encoder-decoder architecture is a type of neural network architecture that combines an encoder and a decoder to solve a wide range of tasks. The encoder processes the input data and produces an encoded representation, while the decoder takes the encoded representation as input and generates an output that is similar to the original input data. The encoder and decoder are typically two separate neural networks that are trained jointly using a supervised learning approach. During training, the encoder takes the input data and produces the encoded representation, which is then fed to the decoder to generate the output. The model is optimized to minimize the difference between the output generated by the decoder and the actual output data.

One common type of encoder-decoder architecture is the sequence-to-sequence (seq2seq) model. In a seq2seq model, the encoder processes the input sequence of words and produces an encoded representation, typically in the form of a fixed-length vector. The decoder then takes the encoded representation and generates the output sequence of words in the target language.

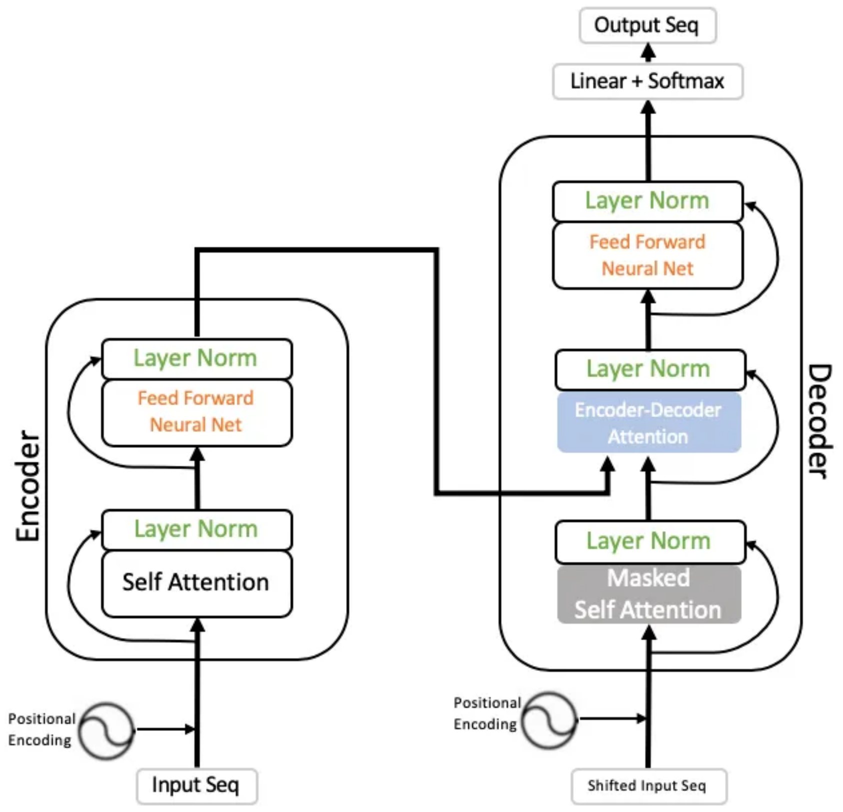


Figure 2.2: Encoder Decoder Architecture

2.2.9 T5 transformer

T5[6] (Text-to-Text Transfer Transformer) is a state-of-the-art transformer-based language model developed by Google. It is based on the same transformer architecture as BERT and GPT-2 but is designed for a specific task of text-to-text transformation. T5 is pre-trained on a large corpus of text and can be fine-tuned on a specific task such as language translation, question answering, or summarization. Unlike other pre-trained language models that are trained for a specific task, T5 is trained to perform a wide range of text-to-text transformations.

T5 uses a variant of the transformer architecture called the Transformer-XL, which is designed to handle longer sequences of text than the original transformer architecture. The Transformer-XL uses a segment-level recurrence mechanism that allows it to handle sequences of arbitrary length, making it well-suited for tasks such as language modeling and text generation.

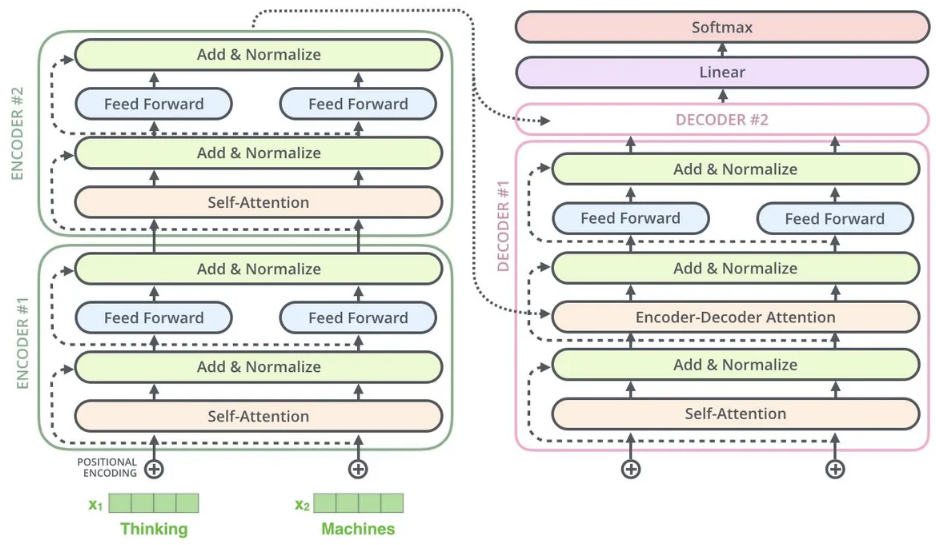


Figure 2.3: T5 Architecture

T5 has a total of 12 layers, all of which are transformer layers. These transformer layers can be divided into two categories: encoder layers and decoder layers. The encoder layers process the input text, while the decoder layers generate the output text. Within the 12 layers of the T5 model, there are 6 encoder layers and 6 decoder layers.

1. **Encoder layer**: The encoder layer processes the input text and consists of the following sub-layers:
   (a) **Multi-Head Attention Layer**: It performs attention mechanism on the input sequence to get a weighted representation of each token, taking into account its relationship with other tokens in the sequence.
   (b) **Feedforward Layer**: It applies a point-wise feedforward network to each position of the sequence independently and identically.
2. **Decoder layer**: The decoder layer generates the output text and consists of the following sub-layers:
   (a) **Masked Multi-Head Attention Layer**: It performs attention mechanism on the output sequence, but it is masked to ensure that tokens can only attend to previous tokens in the output sequence.
   (b) **Multi-Head Attention Layer**: It performs attention mechanism on the encoded input sequence, allowing the decoder to focus on relevant parts of the input when generating the output.

(c) **Feedforward Layer**: It applies a point-wise feedforward network to each position of the output sequence independently and identically.

Both the encoder and decoder layers use residual connections and layer normalization to stabilize the training process. The residual connections allow information to pass through the layers easily, while the layer normalization helps in normalizing the inputs to each layer.

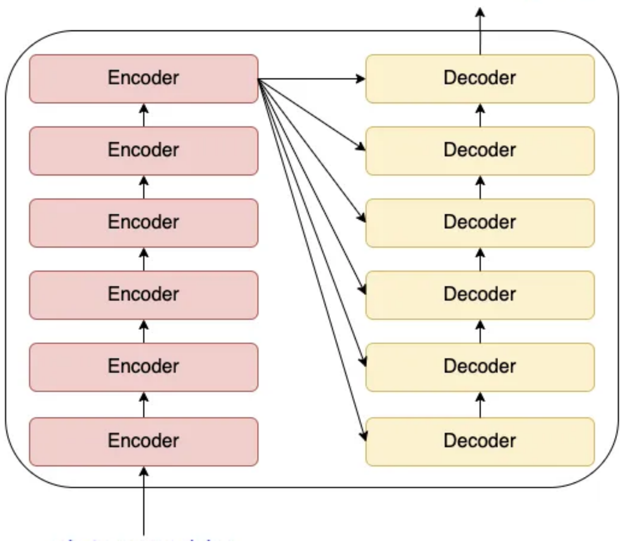


Figure 2.4: Transformer Layers

2.2.10 BLEU Score

BLEU (Bilingual Evaluation Understudy) score is a metric used for evaluating the quality of machine translation outputs. It measures the similarity between a machine-generated translation and one or more human-generated reference translations. The BLEU score works by comparing the n-gram sequences in the machine-generated translation to those in the reference translations. The BLEU score considers the precision of the n-gram sequences in the machine-generated translation by comparing them with the reference translations. It calculates a modified precision score for each n-gram sequence, which is the number of times the n-gram occurs in the machine-generated translation that also appears in any of the reference translations. This modified precision score is then weighted based on the n-gram length and summed to give a cumulative score. The cumulative score is then normalized by dividing it by the total number of n-grams in the machine-generated translation. The resulting value ranges from 0 to 1, with higher values indicating a better quality translation.

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right) \tag{2.5}$$

where:

- BP: the brevity penalty term that penalizes generated sentences that are shorter than the reference sentences, calculated as $\min\left(1, \exp\left(1 - \frac{\text{reference length}}{\text{output length}}\right)\right)$
- N: the maximum n-gram order to consider
- w_n: the weight assigned to the n-gram precision, with equal weights typically used (i.e., $w_n = \frac{1}{N}$)
- p_n: the n-gram precision, calculated as the count of n-grams in the generated sentence that also appear in the reference sentence, divided by the total count of n-grams in the generated sentence

The unnormalized BLEU score is a variant of the BLEU score that does not use any length normalization when calculating the score. Unlike the standard BLEU score, which divides the geometric mean of the n-gram precisions by a brevity penalty term, the unnormalized BLEU score simply calculates the geometric mean of the n-gram precisions. While the standard BLEU score is generally preferred due to its ability to handle different-length reference and generated sentences, the unnormalized BLEU score can be useful in certain situations where length normalization may not be necessary or desired, such as when comparing sentence pairs that have the same length. To calculate the unnormalized BLEU score, the formula is the same as that of the standard BLEU score, except that the brevity penalty term is not used.

2.2.11 AdamW

The Adam optimizer is widely used in optimizer to adapt the learning rate for each parameter based on the estimate of the first and second moments of the gradients. This makes it possible to use a high learning rate without causing the model to diverge.

However, there is a problem with the Adam optimizer when it comes to weight decay. Weight decay is a regularization technique used in deep learning to prevent overfitting. It works by adding a penalty term to the loss function that encourages the model to have smaller weights. The problem with Adam is that it applies weight decay to all parameters equally, including the ones that shouldn't be regularized, such as the bias terms. This can lead to suboptimal performance.

AdamW is a modification of this Adam optimizer. AdamW solves this problem by decoupling weight decay from the gradient-based optimization step. It achieves this by applying weight decay directly to the weights after each optimization step, rather than including it in the update rule. This means that weight decay is only applied to the parameters that should be regularized, such as the weights, and not to the ones that shouldn't, such as the bias terms. This results in improved performance and better convergence.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.7}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m} t - \alpha \lambda \theta t - 1 \tag{2.8}$$

where:
t is the current iteration
α is the learning rate
β_1 and β_2 are exponential decay rates for the first and second moments of the gradients, respectively
g_t is the gradient at iteration t
m_t and v_t are the first and second moment estimates, respectively
$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ are bias-corrected estimates
ϵ is a small constant to prevent division by zero
θ_t is the model parameter at iteration t
λ is the weight decay coefficient

2.3 Frontend Theory

Frontend web development has evolved considerably over the years, with many frameworks and libraries available to help developers create more dynamic and responsive user interfaces. Vue.js and Tailwind CSS are two such technologies that have gained popularity among frontend developers in recent years.

2.3.1 Vue.js

Vue.js is a progressive JavaScript framework that is designed to help developers build scalable and maintainable user interfaces. It is a popular choice for building single-page applications and allows developers to create reusable components that can be easily integrated into their projects. Vue.js also offers a number of powerful features, such as reactive data binding, computed properties, and directives, which make it easier for developers to create dynamic and responsive user interfaces.

2.3.2 Tailwind CSS

Tailwind CSS, on the other hand, is a utility-first CSS framework that allows developers to create custom designs quickly and efficiently. It provides a set of pre-defined utility classes that can be combined to create complex designs without the need for custom CSS. Tailwind CSS also includes a number of features, such as responsive design utilities, hover and focus states, and custom color palettes, that make it easier for developers to create visually appealing designs.

2.3.3 Tailwind CSS with Vue.js

When used together, Vue.js and Tailwind CSS can provide developers with a powerful toolkit for creating modern, responsive web applications. Vue.js can be used to create the core application logic and user interface components, while Tailwind CSS can be used to style and design those components. This allows developers to focus on the functionality of their applications without having to worry about the intricacies of CSS.

One of the key benefits of using Vue.js and Tailwind CSS together is the ability to create modular, reusable components. Vue.js components can be easily styled with Tailwind CSS classes, allowing developers to create custom designs that can be reused throughout their applications. This can save a significant amount of development time and effort, as developers do not have to create custom CSS for each component.

Another benefit of using Vue.js and Tailwind CSS together is the ability to create responsive designs quickly and efficiently. Tailwind CSS includes a number of responsive design utilities, such as breakpoints and screen size classes, that can be used to create designs that adapt to different screen sizes and device types. When combined with Vue.js, developers can create responsive user interfaces that are both functional and visually appealing.

In addition to these benefits, using Vue.js and Tailwind CSS together can also improve the maintainability and scalability of frontend applications. Vue.js allows developers to create clean, organized code that is easy to maintain and update, while Tailwind CSS provides a consistent set of design patterns and styles that can be easily scaled and modified over time.

2.4 Technical Details

2.4.1 Python

Python is a high-level, interpreted programming language. It is widely used for various purposes, including web development, data analysis, artificial intelligence, scientific computing, and more. Python is known for its simplicity, ease of use, and readability, making it an ideal language for beginners as well as experienced programmers. Python is generally used in the field of artificial intelligence, with popular machine learning frameworks like TensorFlow and PyTorch built on top of Python. Python's simplicity and ease of use make it an ideal language for prototyping and testing machine learning models.

2.4.2 Numpy

NumPy is a Python library for numerical computing, specifically designed for working with arrays and matrices. It provides a powerful set of tools for performing mathematical operations on large datasets. NumPy arrays are stored in memory in a contiguous block, which makes it faster to perform operations on them compared to Python lists. NumPy also provides a set of built-in functions for performing common mathematical operations, such as matrix multiplication, dot products, and trigonometric functions.NumPy also provides functions for indexing and slicing arrays, making it easy to extract specific data from a larger dataset.

2.4.3 Huggingface

Hugging Face is a company that provides a suite of natural language processing (NLP) tools and libraries, including pre-trained models, datasets, and training pipelines. The company is best known for its work on the Transformers library. The Hugging Face Transformers library provides a wide range of pre-trained models for tasks such as text classification, machine translation, question answering, and more. In addition to pre-trained models, the Transformers library also provides a range of tools for fine-tuning and training models on custom datasets. This includes data preprocessing tools, training pipelines, and evaluation metrics, making it easier for researchers and developers to build and train their own NLP models.

2.4.4 Scipy

SciPy is a Python library for scientific and technical computing, built on top of the NumPy library. It provides a range of tools for performing scientific computations, including optimization, integration, linear algebra, signal processing, and more. One of the key features of SciPy is its integration with NumPy. SciPy provides a range of functions for performing numerical computations on NumPy arrays, making it easy to perform complex mathematical operations on large datasets.

3. Methodology

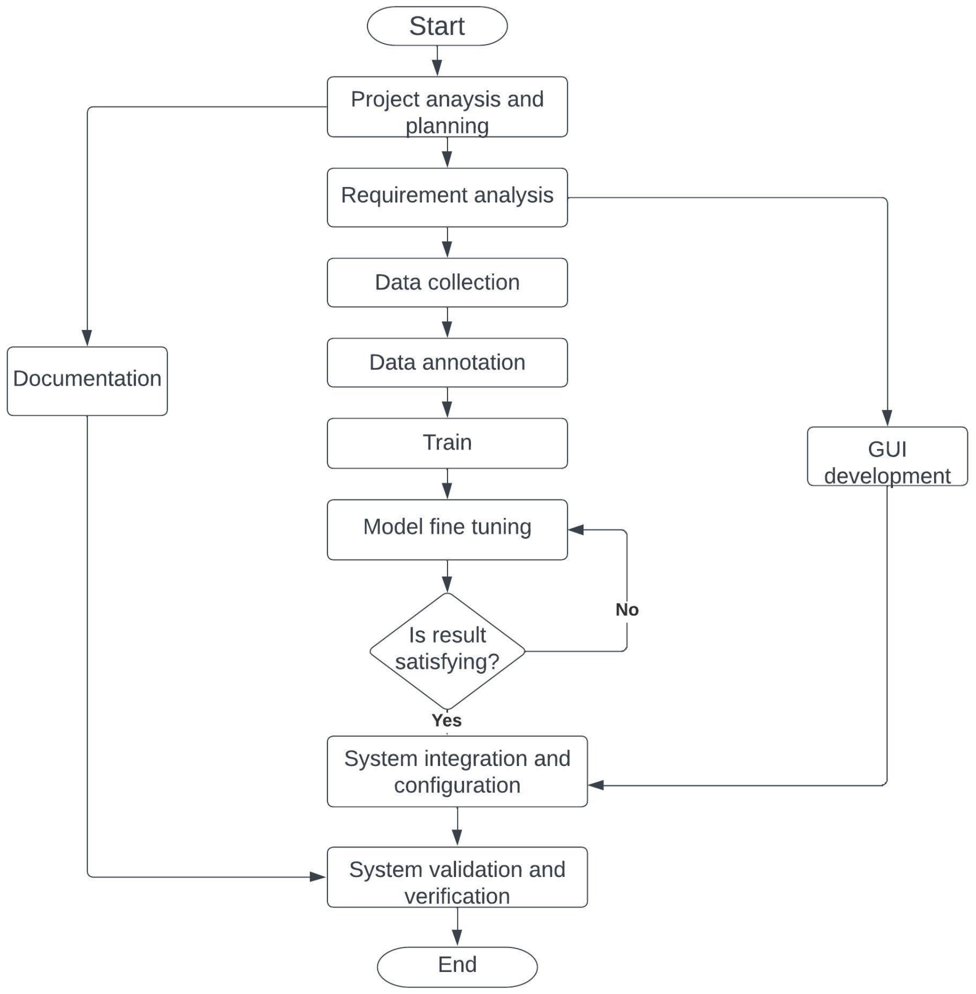


Figure 3.1: Methodology

3.1 Project Analysis and Planning

This was the first step of the project. We started analysis of the project. We researched some papers and articles useful for the project which are listed in refrences below. We also make a proper planning and schedule to complete the project. We created a Gantt chart to guide us through the other phases of the project in order to stick with the time available with us to complete the project.

3.2 Feasibility Analysis

The purpose of this feasibility analysis step is to assess the practicality and viability of a T5 transformer model for generating questions from text. The T5 transformer is a state-of-the-art language model that has shown promising results in various natural language processing tasks. The T5 transformer model requires significant computational resources to train and fine-tune. The model architecture is complex and requires access to specialized hardware such as GPUs or TPUs. We have assessed the availability of these resources and have found that they are readily available in the market. We used Google Colab as training and fine-tuning platform. Additionally, a large corpus of high-quality text data is required to train and validate the model. We have evaluated the availability and quality of such data and have found that there are various publicly available datasets that can be used for this purpose.

3.3 Requirement Analysis

Requirement analysis was the second step of the project. It is the process of identifying and documenting the needs and expectations of stakeholders for our software project. Early on, we decided to focius on academic application of this project. Thus our key stakeholders were mainly teachers and students. Next, we gathered, categorized, prioritized and validate requirements. This left us with following requirements :

- Generate question from text
- Generate question set of selected subjects
- Document the project

3.4 Data collection

For this project, we collected data from several sources, including the SQuAD dataset, notes, textbooks, and question banks. The Stanford Question Answering Dataset (SQuAD) is a popular benchmark dataset for natural language processing (NLP) research. It consists of a collection of Wikipedia articles and their associated questions and answers, and is widely used for training and evaluating models that perform question answering (QA) tasks. The dataset contains over 100,000 question-answer pairs that cover a broad range of topics, including history, science, literature, and more. Each question-answer pair is associated with a specific paragraph from a Wikipedia article, which provides context for the question and answer. One of the unique features of the SQuAD dataset is that the questions and answers are created by human annotators, rather than generated automatically. This ensures that the dataset contains high-quality, accurate information that is representative of how humans ask and answer questions.

In addition to the SQuAD dataset, we also used textbooks and question banks as a source of additional training data. We selected a set of textbooks that covered following subjects :

- Organization and Management
- Engineering Professional Practice
- Energy, Environment and Society
- Software Engineering
- Object Oriented Analysis and Design

These subjects were chosen as a composition that provides a mix of both computer-based and non-computer based subjects. Also, some of these subjects show the limitations of our model. We used them to extract additional question-answer pairs that were not present in the SQuAD dataset. We also used question banks from various sources to supplement our training data. To collect the data from the textbooks and question banks, we employed a combination of manual and automated methods. We manually reviewed the textbooks and question banks to identify relevant questions and answers, and then used automated tools to extract the data and format it in a way that could be used for training our model.

3.5 Data Annotation

In order to maintain quality of generated questions, data annotation is necessary. Annotation involves the process of checking the quality of questions. First part of our data annotation was to remake our subjectwise data into SQuAD format to ensure consistency in training and fine tuning data. In our project, we utilized manual annotation to annotate the collected data. Since, SQuAD data was handpicked, we also did the same to keep the quality of questions. The second part of the project required generating questions from subjects in form of IOE based question set. It meant we had to determine the types of questions to be generated from the given text. Just because a question is sound and valid doesn't mean it could be asked in the exams. The annotation process was done by a team of trained annotators who were provided with clear guidelines and instructions for labeling the data. Ensuring consistency and accuracy in the annotation process was a critical component of this project. To achieve this, we provided our annotators with a detailed set of guidelines and rules for labeling the data. Additionally, we implemented regular quality checks to ensure that the annotation was being done correctly and consistently. This involved reviewing a sample of the annotated data on a regular basis to check for any errors or inconsistencies. If issues were identified, we would provide additional training to the annotators to ensure that they were following the guidelines correctly.

3.6 Data Split

In this step, the dataset was split into two parts. 90% of the dataset was used to train the model and the rest 10% of the data was used to validate the model. AdamW optimizer was used to optimize the model.

3.7 Pretraining and Finetuning

We used the T5 model for this project because is well-suited for question generation task due to its ability to perform both sequence-to-sequence and text-to-text tasks. To pretrain a T5 model for question generation, we needed a large corpus of text data that includes both source text and target questions. Thus, we chose SQuAD. SQuAD was chosen ahead of its peers due to the combination of high-quality data, diverse topics, challenging questions, and a standardized evaluation metrics. Datasets like NewsQA don't cover a variety of topics. Similarly, SQuAD v2.0 includes unanswerable questions which aren't being dealt in this project. Similarly, TriviaQA is sourced from quiz bowl competitions, which are known for their difficult and esoteric questions. Thus, TriviaQA may not be representative of the types of questions that people ask in real-world settings.

Once we had SQuAD, you used the T5 model to pretrain on the text-to-text task of generating questions from source text. We used the "text-to-text" version of T5 and trained it on a combination of question-answer pairs and source-answer pairs. The training of model had following parameters.

```
dataloader_workers=4
epochs= 10
learning_rate = 1e-3
max_length = 512
train_batch_size = 4)
valid_batch_size = 32)
```

It is important to fine-tune the pre-trained model on the specific downstream task of question generation using supervised learning. Thus, we used a smaller, more targeted dataset for fine-tuning, manually generated from aforementioned subjects, to adapt the pre-trained model to our specific use case.

3.8 Evaluation

To evaluate the quality of the generated questions, we used the Bilingual Evaluation Understudy (BLEU) score, which is a widely used metric for evaluating the similarity between the generated questions and the reference questions. The following BLEU score was obtained by taking arithmetic mean of 10 comparison between generated questions and reference questions for validation set.

```
BLEU_1 = 54.98
BLEU_2 = 30.13
BLEU_3 = 16.56
BLEU_4 = 7.74
```

This gives the unified BLEU score 0.208. This result may look bad at first glance but if we compare it to the best OQPL models, we aren't very far off.

```
BLEU_1 = 55.60
BLEU_2 = 31.37
BLEU_3 = 16.79
BLEU_4 = 8.27
```

This gives the unified BLEU score 0.219.[7]

3.9 System Integration and Configuration

During this step, all the components of the system were integrated to form a single program. The integration testing was also performed to ensure that the system as a whole works fine.

3.10 Documentation

We began documentation of our project at the very start. By documenting the project from start to finish, we ensured that everyone involved in the project understands the goals, requirements, and processes. This documentation will also serve as a reference for future projects, making it easier to build on the success of the current project.

3.11 GUI Development

GUI development was its own project. We took it parallelly along with system.

3.11.1 Requirements Gathering

The first step in the development process was to gather requirements for the web application. The requirements were defined by us and documented in a requirements document. The document included design and branding guidelines.

3.11.2 Wireframing

Once the requirements had been defined, a wireframe was created to provide a visual representation of the user interface. The wireframe was created using Figma and included the main components and layouts of the application, such as the , main content areas, and form inputs. The wireframe was reviewed and approved by the team members before proceeding to the next step.

3.11.3 Planning the Architecture

The next step was to plan the architecture of the web application. The architecture was designed to be scalable and maintainable, with reusable components and modules. Vue.js was chosen as the frontend framework, and Tailwind CSS was chosen as the CSS framework. The API was developed using Node.js and Express.js.

3.11.4 Developing the Frontend

The frontend was developed using Vue.js and Tailwind CSS. The components and modules were developed to be reusable, allowing for efficient development and maintenance. Vue.js was used to handle the application's state, and Tailwind CSS was used to style the components and layouts. The frontend was tested extensively to ensure that it met the requirements and was visually appealing.

3.11.5 Integration and Testing

Once the frontend and backend had been developed, they were integrated and tested. The frontend was connected to the API using fetch api, which allowed for easy data fetching and posting. The integration was tested to ensure that the frontend and backend were communicating correctly and that data was being displayed and updated correctly.

3.12 Model Architecture

To generate questions from text using the T5-base model, the input to the model is a concatenation of the text to generate questions from and a special separator token "¡sep¿". The T5 model then applies a series of transformer layers to the input to generate a sequence of output tokens, which can include both question words and question marks.

Each transformer layer in the T5-base model consists of three sublayers: multi-head self-attention, a feedforward neural network, and layer normalization. The multi-head self-attention layer allows the model to attend to different parts of the input sequence and capture long-range dependencies. The feedforward neural network applies non-linear transformations to the output of the self-attention layer, while layer normalization ensures that the output of each sublayer has a consistent distribution.

The T5 model is trained using a combination of maximum likelihood estimation and self-supervised learning objectives. During training, the model learns to generate questions from text by predicting the next token in the output sequence given the input sequence and previous tokens. The model is optimized to minimize the negative log-likelihood of the correct output sequence given the input. So the number of parameters to be learned are:

The number of parameters in a single transformer layer is:

$$n_{params} = (4 * d_{model}^2 * n_{heads}) + (4 * d_{model} * d_{ff}) + (2 * d_{model}) \tag{3.1}$$

where:

d_{model} is the dimensionality of the model's hidden state

n_{heads} is the number of attention heads

d_{ff} is the size of the feedforward neural network

$$n_{params} = (4 * 768^2 * 12) + (4 * 768 * 3072) + (2 * 768)$$

For T5-base with 12 layers, the total number of learnable parameters is:

$$n_{params} = 12 * [(4 * 768^2 * 12) + (4 * 768 * 3072) + (2 * 768)] = 220,027,520$$

.

4. System design

4.1 Overview of System Architecture

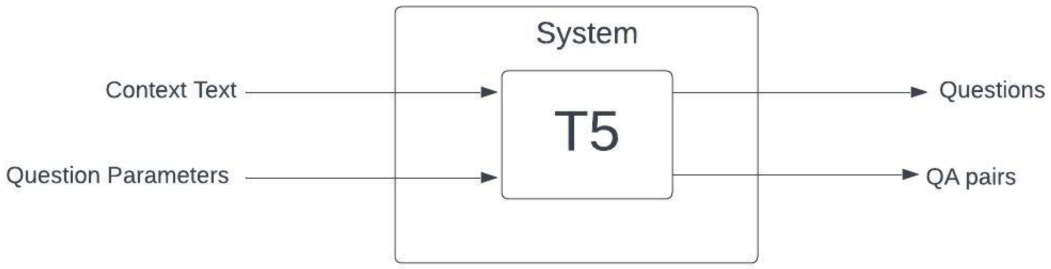


Figure 4.1: Overview of system architecture

The user provides context text and question parameters to the system. The system consists of a single T5 model.The model does the pre-processing of inputs. This includes tokenization and encoding of the inputs. The pre-processed input is then used by the T5 model for generating the output questions and/or question-answer pairs. The output generated by the T5 model is then post-processed to format it appropriately. The final output is returned to the user.

4.2 Use Case Diagram

User first gives an text as input to the system. The text is saved by the system while the user enters question parameters. User can then select to generate question n using the trained model. User selects view result to see the questions and question-answer pairs of the input text. Alternatively, the user can also select a subject and modify distribution of marks to generate a question set.

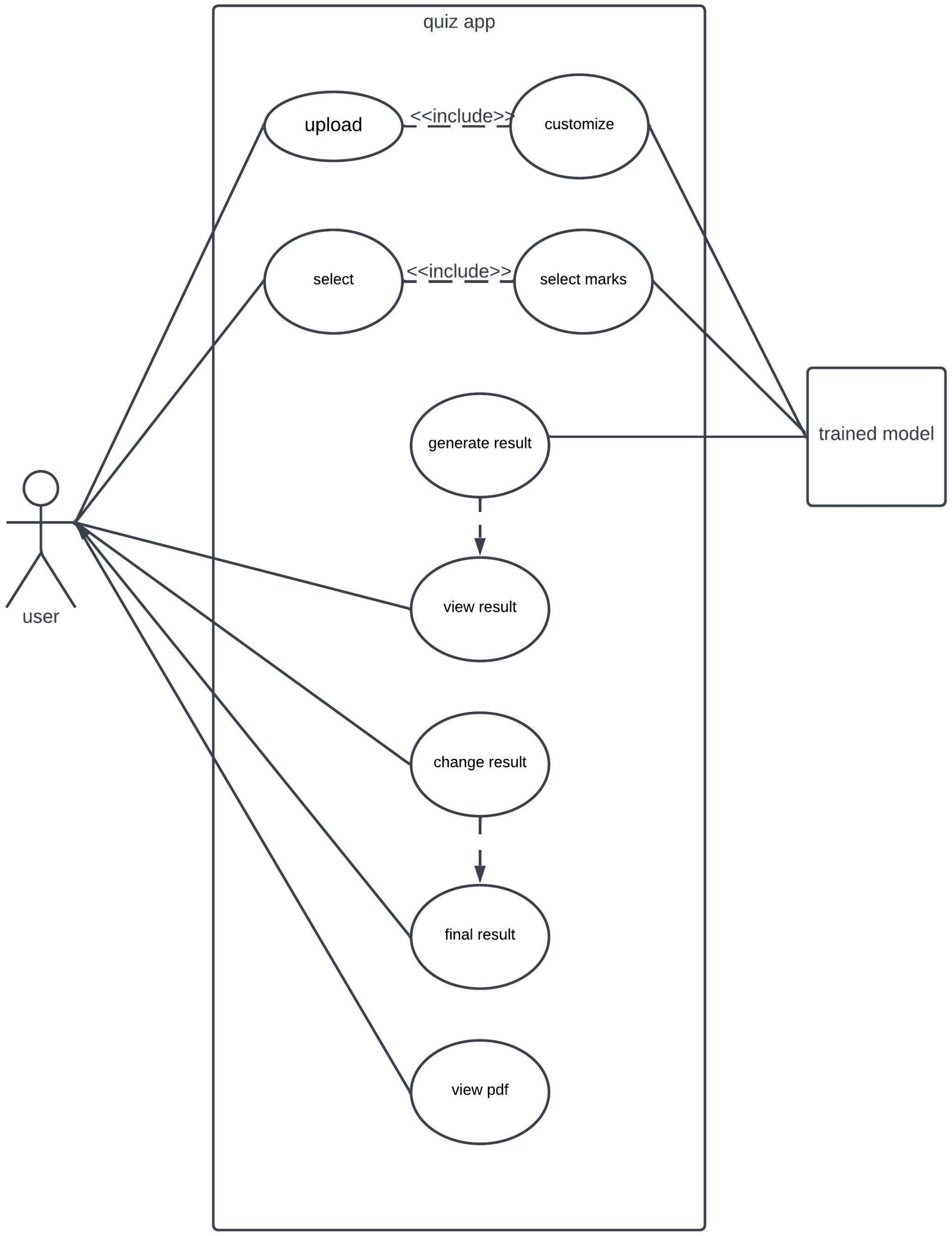


Figure 4.2: Use Case Diagram

4.3 System Context Diagram

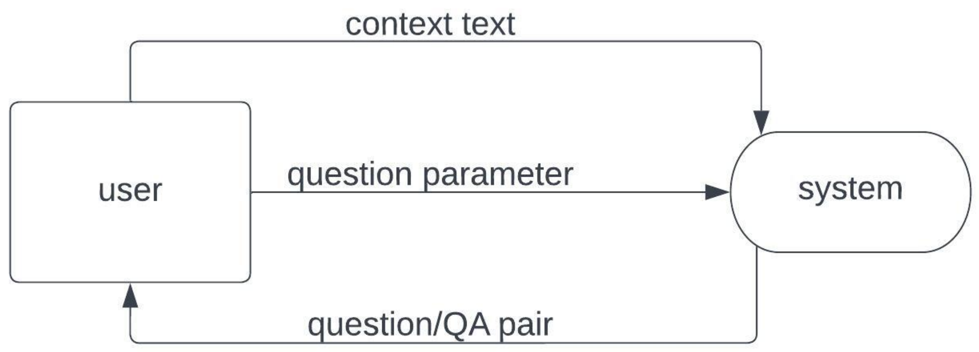


Figure 4.3: System Context Diagram

The system context diagram provides a high-level view of a system and shows its interactions with user(external entity). System represents our system and user is the only external entity that interacts with the system. There are three basic interactions. Use provides context text to the system along with question parameters. The system then returns questions and/or question-answer pairs. This context holds true for generating subjectwise question where the only difference being that the system already has context text and just needs name of subject that points to the text.

4.4 Data Flow Diagram

The data flow diagram of the system shows the user input as the source of the data flow, with the context text and question parameters flowing into the inference for processing. Similarly, the trained model can also viewed as an input for inference. The system then uses trained model along with text and parameters to provide output. The output of the system would be either questions or question-answer pairs, which would be returned to the user as the final output of the system.

Similarly, for the generation of subjectwise questions, the data flow diagram of the system shows the user input as the source of the data flow, with the subject information flowing into the inference for processing. Similarly, the trained model can also viewed as an input for inference. The system then uses trained model along with inputs to provide output. The output of the system would be a questionset, which would be returned to the user as the final output of the system.

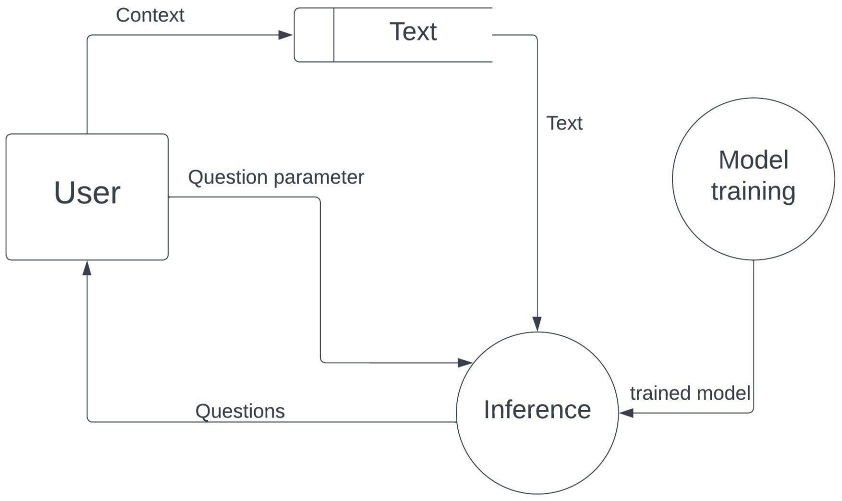


Figure 4.4: DFD for questions from text

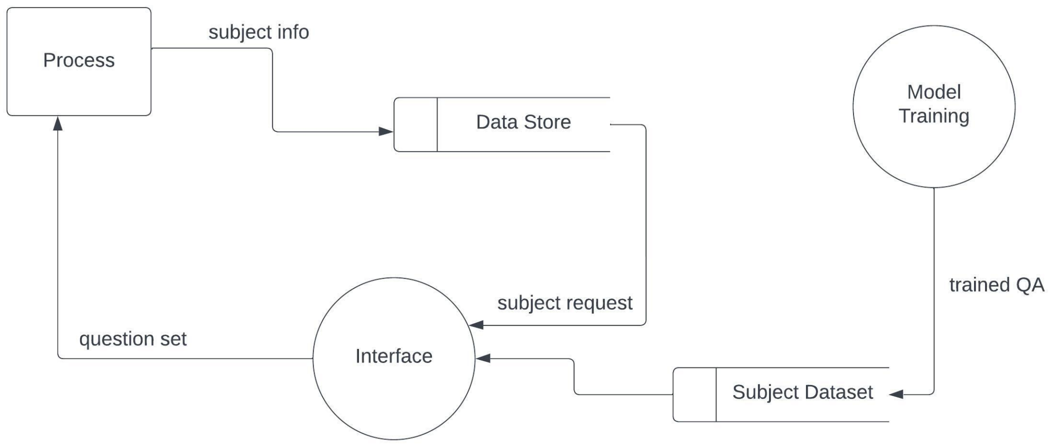


Figure 4.5: DFD for question set from subject

4.5 Sequence Diagram

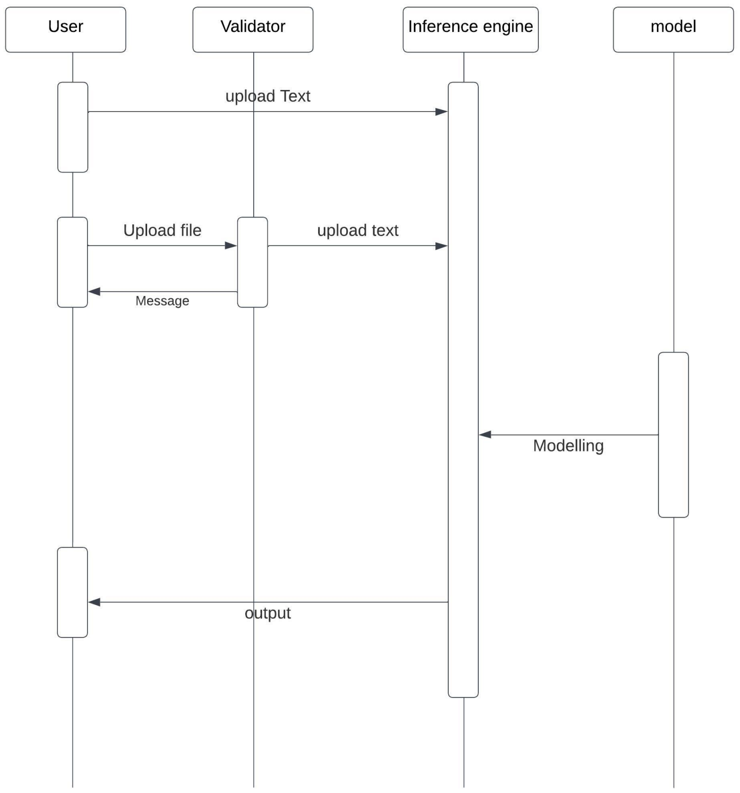


Figure 4.6: Sequence Diagram for question from text

The diagram depicts the sequence of events in our system. The user inputs text or a textfile. In case of textfile, there is an extra validation step that ensures the text file is acceptable. Next the inference engine gets the input from user and trained model and uses those to infer the output.

4.6 Activity Diagram

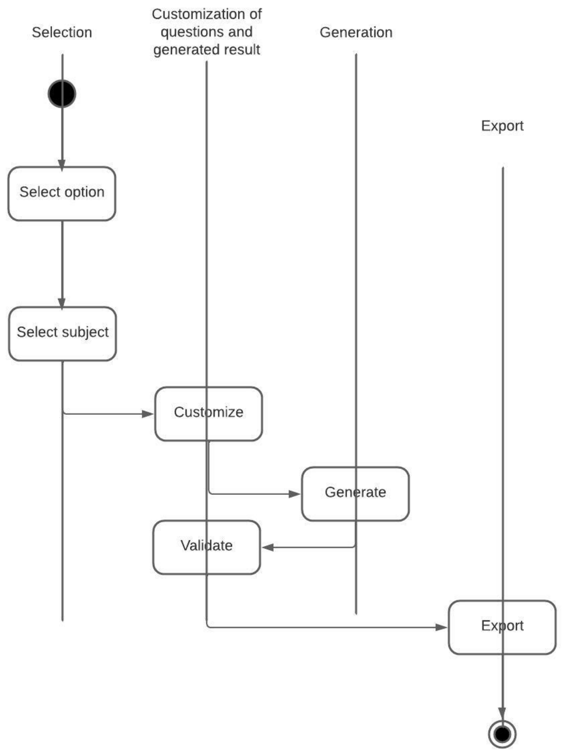
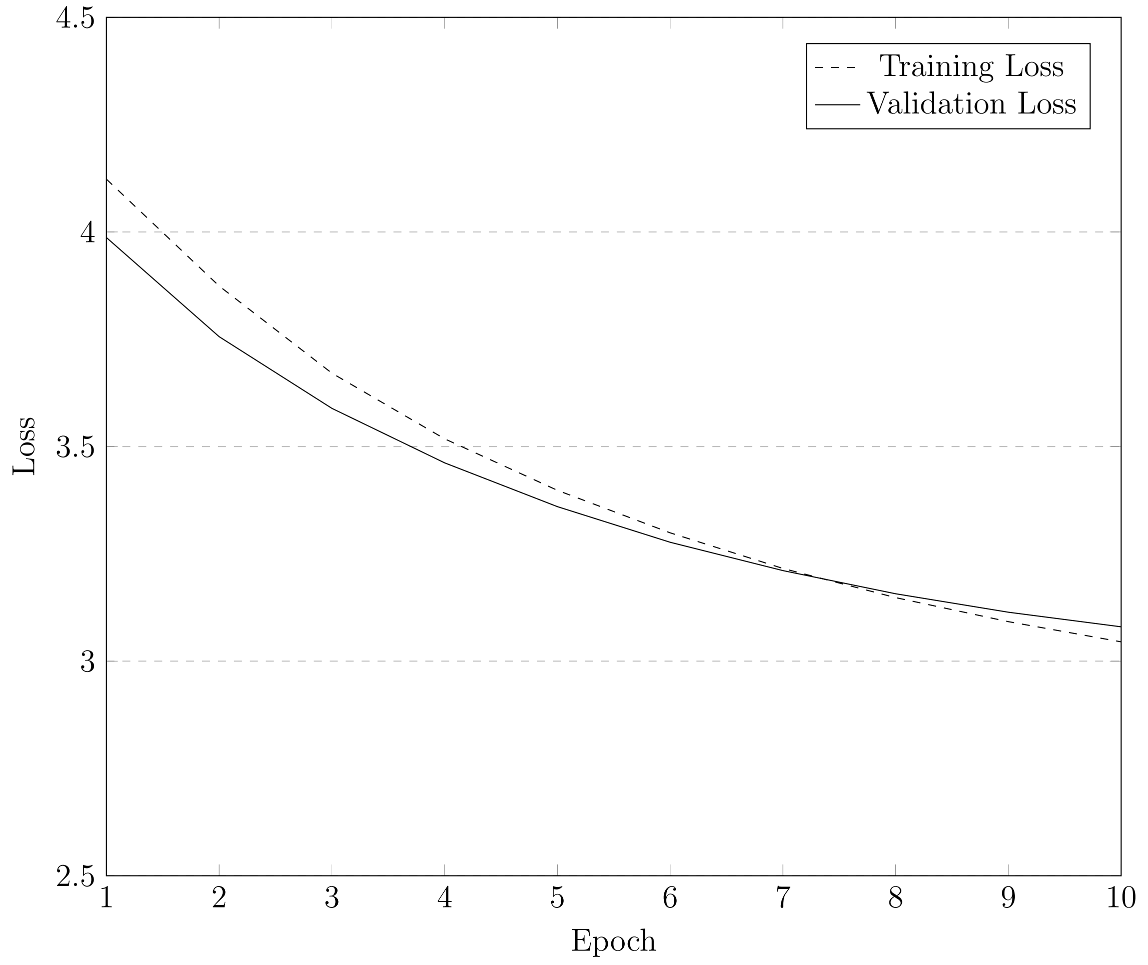


Figure 4.7: Activity Diagram for subjectwise questions

The diagram depicts the activities within our system. The user interacts with a select option and selects a subject, User then customizes mark distribution. The system then generates the question set and validates the formatting. Finally, the system can export the text as pdf.

5. Results & Discussion

5.1 Model Results



From the given curve, it can be seen that the model has reached a saturation point after the 7th epoch. Beyond the 7th epoch, the decrease in the loss is not significant, and the training and validation losses are not improving by a large margin. This suggests that the model has already learned most of the relevant patterns in the data and further training may not improve its performance significantly. Moreover, continuing training for more epochs might lead to overfitting, where the model starts fitting the training data too closely and loses its ability to generalize to new data.

Therefore, it was reasonable to stop training the model at the 10^{th} epoch, as it has already learned most of the relevant patterns in the data, and continuing training beyond that point did not significantly improve its performance.

5.1.1 Question Quality Evaluation

There is no metric for evaluating the quality of question. Thus, the evaluation were done manually by team members. The model was trained on a particular subject until the question quality was satisfactory,

5.1.2 Model Performance Assessment

There is a distinct lack of metric to assess the performance of a model that generates questions from text. The BLEU score test is the closes metric which still does not provide satisfactory results. For the following text,

```
System design is the process of defining the architecture, components, modules,
interfaces, and data for a system to satisfy specified requirements
```

Following questions are possible:

```
    "What is system design?"
    "Define system design."
    "What is the definition of system design?
```

These three questions have the same meaning and are arguably of equal quality. However, the BLEU score of these questions are:

```
BLEU score for sentence 1: 0.48
BLEU score for sentence 2: 0.57
BLEU score for sentence 3: 0.37
```

Thus the system lacks a quality model performance assessment metric.

5.1.3 Result Comparison

BLEU	BLEU1	BLEU2	BLEU3	BLEU4	unifiedBLEU
Our Model	54.98	30.13	16.56	7.74	0.208
best OQPL model	55.60	31.37	16.79	8.27	0.219

Table 5.1: Comparison of BLEU scores

For the best OQPL models, only BLEU scores of 1gram to 4-gram were available.[7]. We assumed the BP for it to be 1.

5.2 Project Result

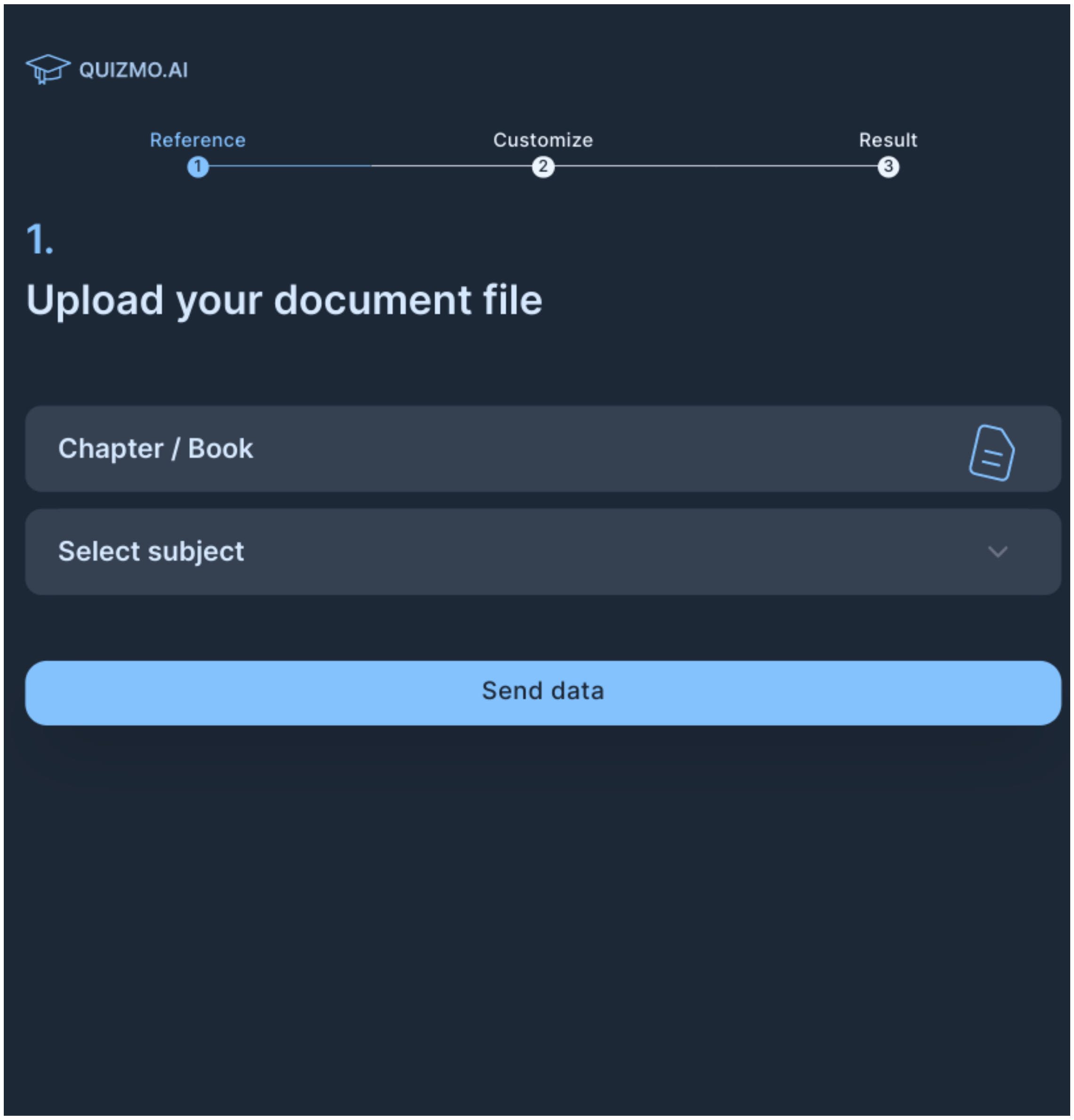


Figure 5.1: Result: HomePage

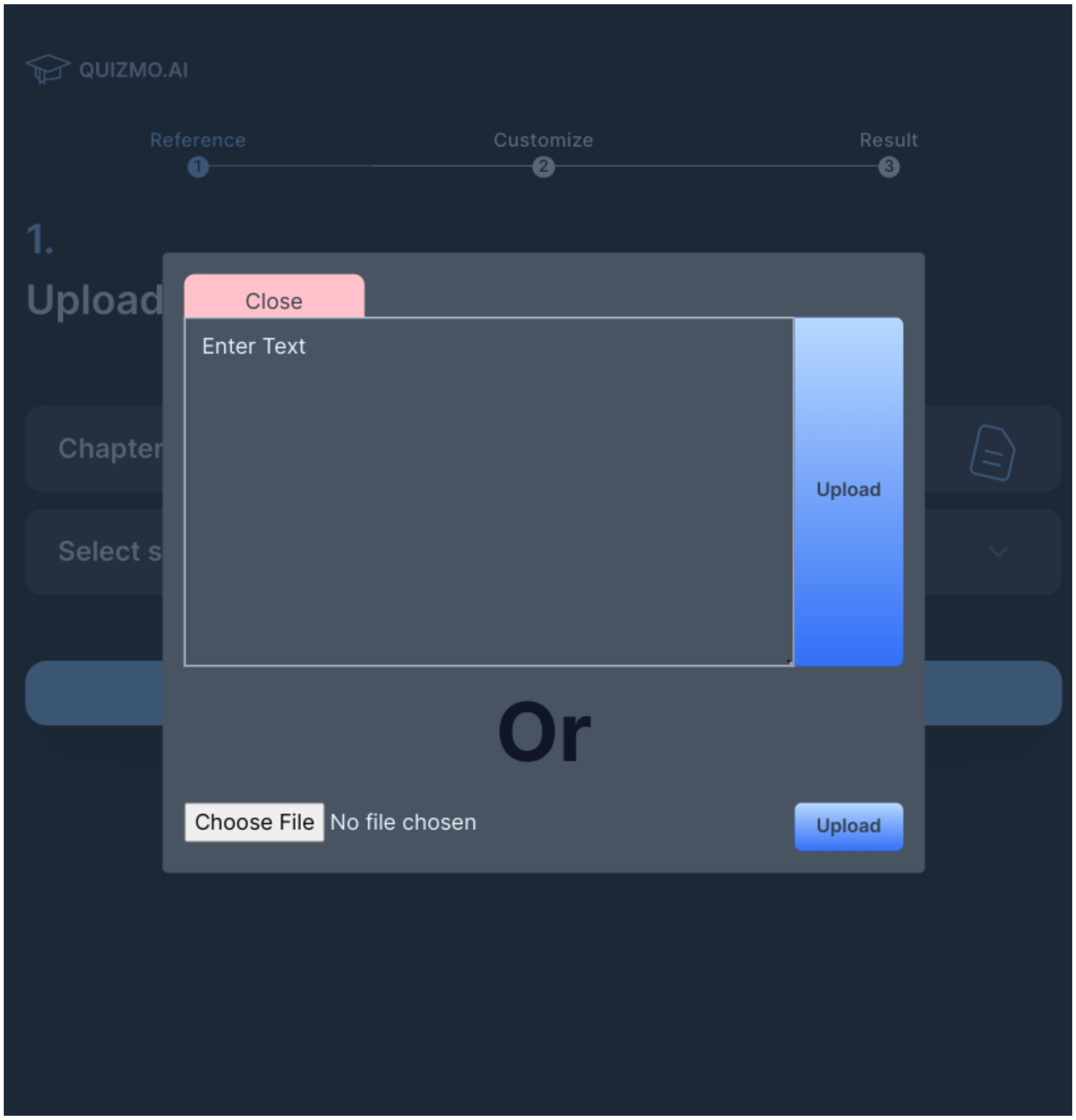
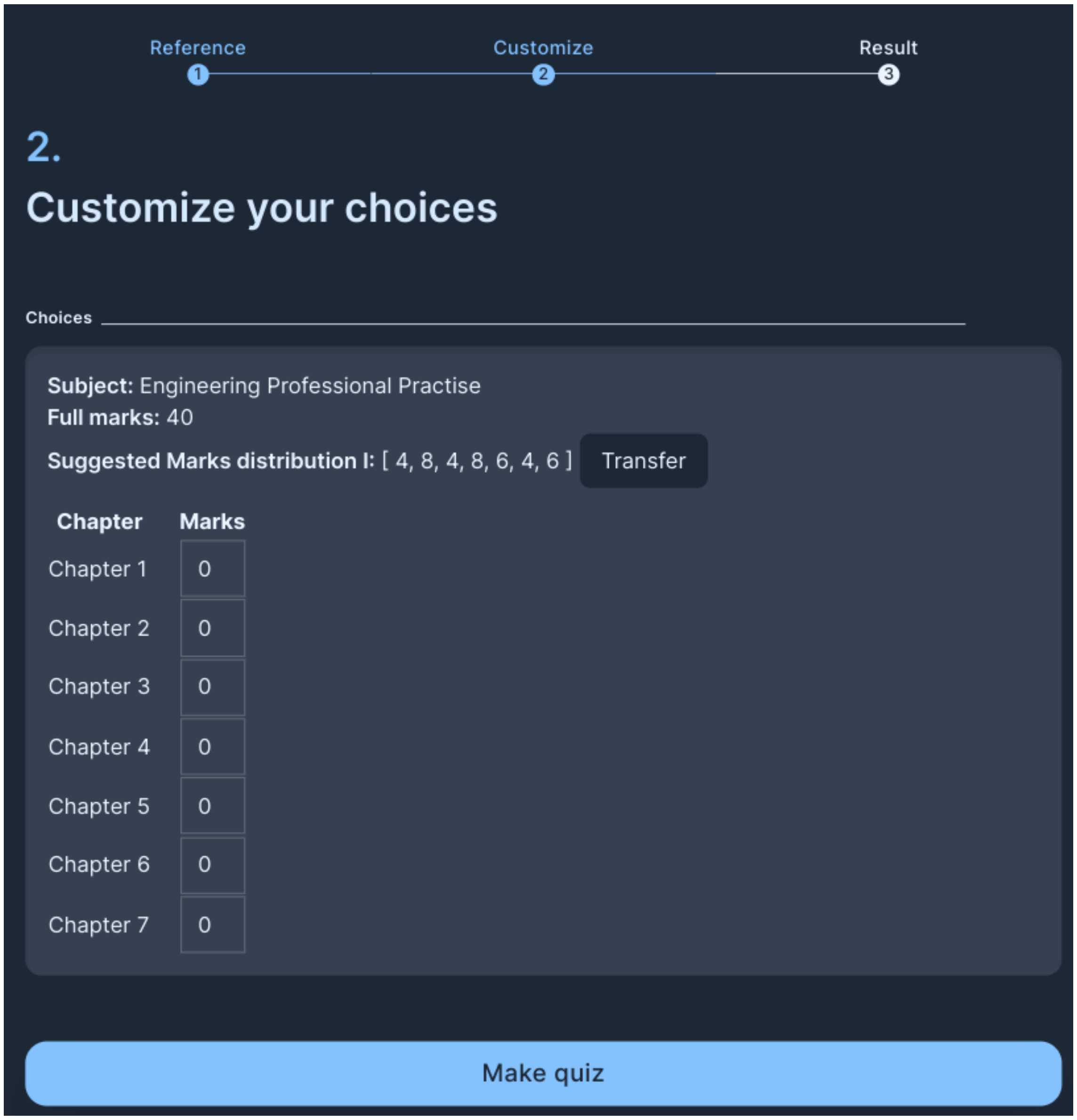


Figure 5.2: Result: Text Selection



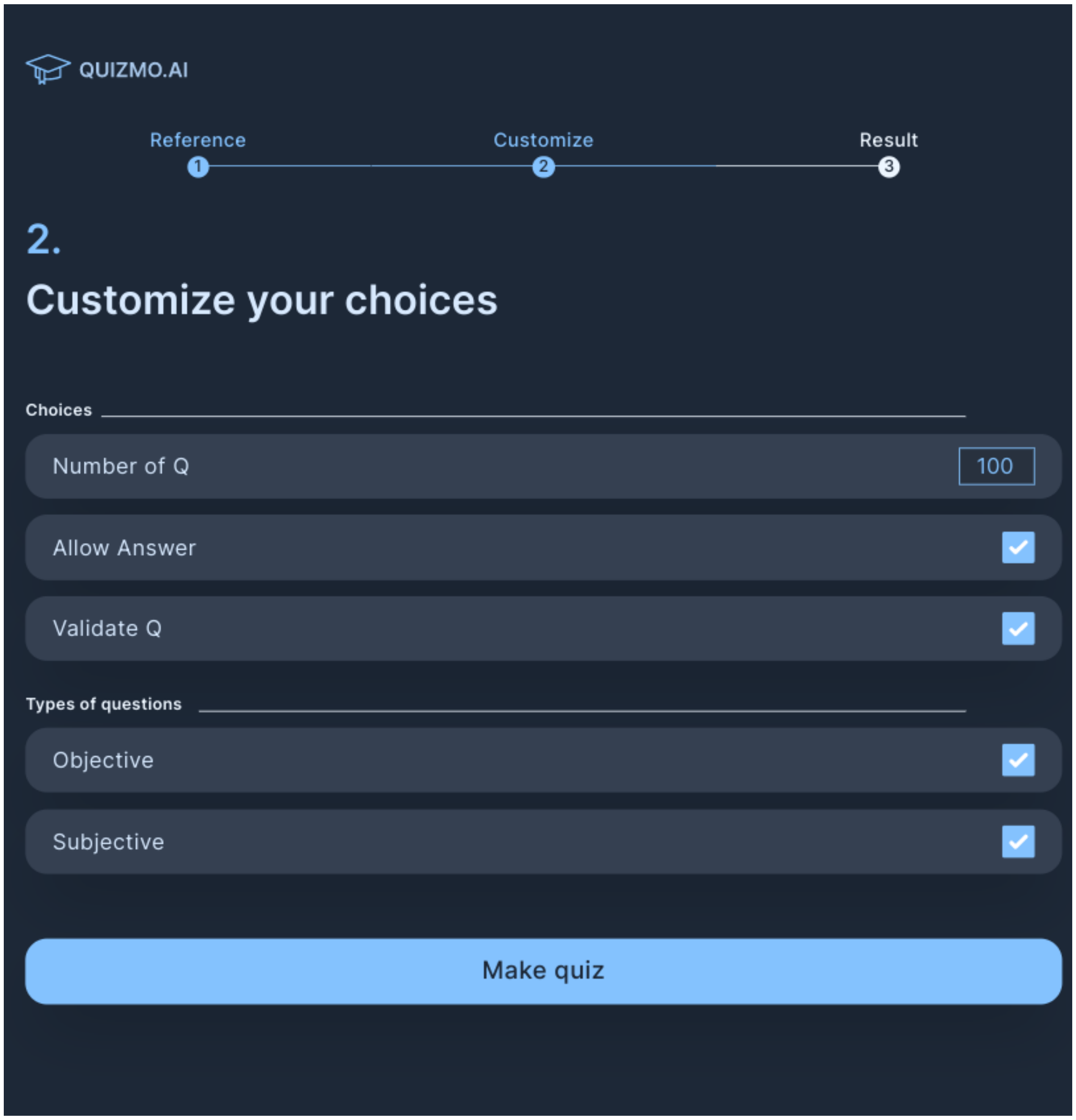
2.
Customize your choices

Choices

Subject: Engineering Professional Practise
Full marks: 40
Suggested Marks distribution I: [4, 8, 4, 8, 6, 4, 6]

Chapter	Marks
Chapter 1	0
Chapter 2	0
Chapter 3	0
Chapter 4	0
Chapter 5	0
Chapter 6	0
Chapter 7	0

Figure 5.3: Result: Mark Distribution customization



QUIZMO.AI

2.

Customize your choices

Choices

Number of Q 100

Allow Answer ✓

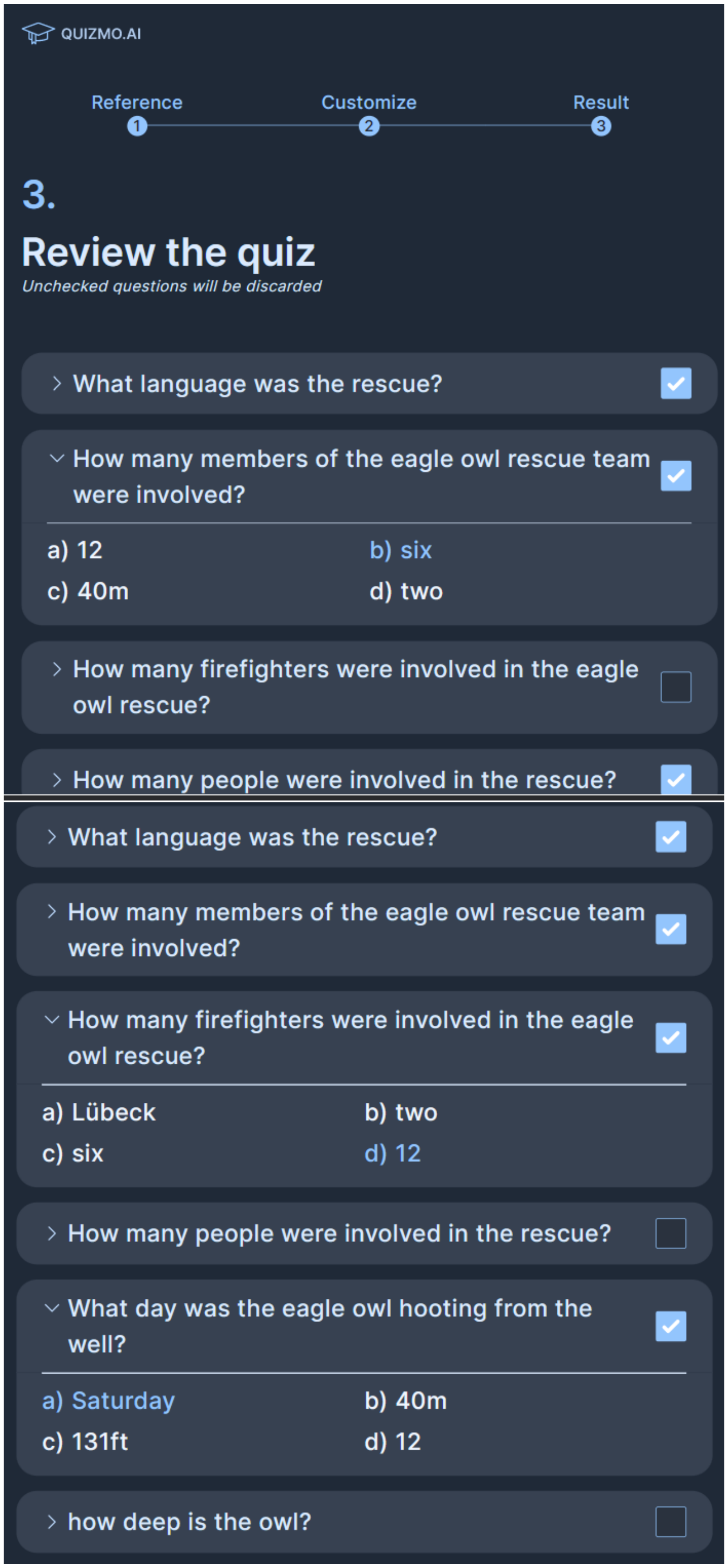
Validate Q ✓

Types of questions

Objective ✓

Subjective ✓

Figure 5.4: Result: Question Customization


QUIZMO.AI

Reference 1 — Customize 2 — Result 3

3.

Review the quiz

Unchecked questions will be discarded

› What language was the rescue? ✓

⌄ How many members of the eagle owl rescue team were involved? ✓

a) 12 [✓ b) six]
c) 40m d) two

› How many firefighters were involved in the eagle owl rescue? ☐

› How many people were involved in the rescue? ✓

› What language was the rescue? ✓

› How many members of the eagle owl rescue team were involved? ✓

⌄ How many firefighters were involved in the eagle owl rescue? ✓

a) Lübeck b) two
c) six [✓ d) 12]

› How many people were involved in the rescue? ☐

⌄ What day was the eagle owl hooting from the well? ✓

[✓ a) Saturday] b) 40m
c) 131ft d) 12

› how deep is the owl? ☐

Figure 5.5: Result: Question Review

6. Conclusions

This project on question generation from text using T5 model is a promising application of natural language processing techniques. By leveraging the power of the T5 model, the project was successful to generate high-quality questions from input text. Although the project focused on academic applications of automatic question generation, further extension to this project can be useful for a variety of applications, such as content creation, and information retrieval.

Throughout the project, several challenges and limitations were encountered. The biggest limitation being length limitations. The T5 model is limited in the length of input that it can take at a time(512 words) and lack of parallel processing options meant that it was difficult to generate questions for long input texts. This also meant that the time to train the model significantly increased. Since training a T5 model is computationally expensive and time-consuming, it limited our ability to experiment with different training data other than SQuAD such as TrivialQA, NewsQA, SQuADv2 and so on. This means that, our claim of SQuAD being the best data set was a result of our preliminary study and not a verified result. Also, the lack of reliable automatic evaluation tools meant human evaluation was required, which was expensive and time-consuming.

In conclusion, this project highlights the potential of using state-of-the-art language models like T5 for generating questions and opens up several avenues for future research. This also provides a gateway for furthur academic research such as exploring the use of multimodal inputs such that figures and numerical data are handled as input. With further advancements in natural language processing techniques, question generation can become an even more valuable tool for various applications in the future.

7. Limitations and Future enhancement

One limitation of our question generation project, or any question generation project in general using T5 base or any other natural language processing model is that it is not be able to generate questions that are outside the scope of the training data. This means that if the training data is limited to a specific domain or topic, the generated questions may not be as diverse or useful for other domains or topics. For example, we can only generate quality questions for a handful of subjects although we can generate quality questions in a variety of fields.

Another limitation of our project is the semantic relationship between questions and answers. Since it is trained on a question-answer data set, the quality of questions are good. However, the answers (mostly, when they need to be summarized) do not necessarily match the questions semantically. Also, since the model isn't trained for summarization, the answers may not necessarily make sense.

Future work for a question generation project using T5 base could increasing the number of subjects the model can generate question sets. By extension, this model can also be trained to generate sample mock question set for Engineering License examination. Another area of research could be to develop more advanced evaluation metrics to measure the quality of generated questions. This could involve using human evaluation or other metrics beyond the traditional BLEU score, such as the relevance and coherence of the generated questions.

References

[1] Ming Liu and Rafael A Calvo. Using wikipedia and conceptual graph structures to generate questions for academic writing support. *IEEE Transactions on Learning Technologies*, 5(3):251–263, 2012.

[2] Yan Zhao, Hongyu Ren, Meng Sun, Xiangyang Liu, and Yan Hu. Adaptive question generation for intelligent education. *IEEE Access*, 6:43420–43430, 2018.

[3] Yiran Li, Chuan Shi, Yue Zhang, and Ying Yan. Automatic generation of domain-specific multiple-choice questions based on textbooks and knowledge graph. *IEEE Access*, 8:214134–214146, 2020.

[4] Tianxing Dai, Nan Yang, Runxin Cui, Le Sun, and Xia Hu. Transformer-based question generation with self-supervised learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5293–5303, Online, November 2020. Association for Computational Linguistics.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, pages 5998–6008, 2017.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[7] Luis Enrico Lopez, Diane Kathryn Cruz, Jan Christian Blaise Cruz, and Charibeth Cheng. Simplifying paragraph-level question generation via transformer language models. *arXiv preprint arXiv:2102.06727*, 2021.

Appendices

7.1 Sample SQuAD dataset

Context: The War of the Spanish Succession (1701–1714) was a European conflict over who would succeed Charles II as King of Spain. The war involved nearly all of the major powers of Europe, with two main alliances opposing each other: the Grand Alliance, led by Austria, Britain, and the Dutch Republic; and the Bourbon Alliance, led by France, Spain, and Bavaria. It was marked by a series of military engagements, mostly in Italy, the Low Countries, and Germany, where armies were raised, financed and led by a complex and intricate network of coalition governments and the financing systems that supported them.
Question: What were the two main alliances in the War of the Spanish Succession?
Answer: The two main alliances in the War of the Spanish Succession were the Grand Alliance, led by Austria, Britain, and the Dutch Republic, and the Bourbon Alliance, led by France, Spain, and Bavaria.