Ministry of Education and Science of the Republic of Kazakhstan
Suleyman Demirel University

Nurbek Ismagulov

Development of a system for generating test questions and answers for a given text

THESIS

Presented in Partial Fulfillment for the
Degree of Master of Science in Computer Science
(degree code: 7M06102)
Department of Computer Sciences
Faculty of Engineering and Natural Sciences

Supervisor: **Assist. Prof. Darkhan Kuanyshbay**

Kaskelen, 2022

Suleyman Demirel University

Faculty of Engineering and Natural Sciences

Department of Computer Science

Dean of Faculty

Associate Professor, PhD Zhamanov A.

«20» 06 2022

Topic of the thesis:

Development of a system for generating test questions and answers for a given text

Thesis submitted as part of the requirements for the award of the MSc in "7M06102 - Computer Science"

Head of Department Assoc. Prof. Cemil Turan

Academic Supervisor Assist. Prof. Kuanyshbay Darkhan

Master's student Nurbek Ismagulov

Kaskelen, 2022

Abstract

Currently, there is a tendency to increase the volume of documents containing complex text structures. Professional text proofreaders provide their services to correct errors in the text for not a small amount of money. A very large number of people every day around the world, who are closely connected with science or education, write a large volume of articles and, accordingly, each and all of them should be written without errors and in the correct version. In order to automate this process, it was necessary to develop an algorithm based on the methods of analysis and correction of the input text. For high-quality text synthesis, it is necessary to use machine learning technologies that require deep knowledge and understanding in this area. Among the many machine learning algorithms Hunspell algorithm seems to be one of the best to solve this issue. The essence of this algorithm is to bring all the words contained in the text to the original format. Thus, this work is based on multilevel segmentation of errors of Kazakh-language text from the Internet or by manual user input. It is worth noting that due to technological progress, the main source of linguistic research is social networks, which is a critical problem due to the dubious fidelity of texts. The main purpose of this work was the formation and development of a spell-checking algorithm for the Kazakh language based on the existing Hunspell algorithm for English. As a result, the Hunspell algorithm was studied, where the methods of extracting the base of the word, as well as the ways of aggregation of the normal form were analyzed. The essence of this algorithm lies in the inherent rule for detecting errors and typos in a word. These rules are contained in the dictionary and its dependence on the algorithm is directly proportional. Thus, the more rules there are, the more errors the algorithm detects and the more accurate it is. In addition, the forms and categories of Kazakh words and their differences from English words were also considered. As a result, an algorithm was created to check the spelling of Kazakh words based on the existing Hunspell algorithm. The significance of this work seems to be that the correct spelling of words, sentences forms a literate society, which is key in the development of modern civilization.

Keywords: Kazakh language, natural languages, processing, Hunspell algorithm, text processing, text correction

Аңдатпа

Мәтіндік материалдан логикалық негізделген және өзекті сұрақтарды құру, сондай-ақ оларды тексеру білім берудің көптеген салаларында кең таралған, сонымен қатар маңызды жұмыс болып табылады.

Әрбір оқу жүйесінде бағалау маңызды. Бағалау жүйесі оқушылардың өздігінен білім алудағы кемшіліктерін анықтап, олардың оқуда дамуына көмектеседі. Сұрақтарды қолмен құрастыру және оларды тексеру көп күш жұмсайтын мәселе, өйткені ол мәтіннің ұзақ үзінділерін оқуды, талдауды, түсінуді және еске түсіруді талап етеді. Нәтижесінде автоматтандырылған бағалау жүйесінің негізгі мақсаты оқу ресурстарынан автоматты түрде сұрақтар құру болып табылады.

Жақында зерттеу әлемінде тілдік модельдеу мәселелерінің бірнеше түріне, әсіресе реттілікке болжауға негізделген әртүрлі сұрақтар генерациялау жүйелері ұсынылды. Дегенмен, мұндай жүйелердің көпшілігі пайдаланушыға ыңғайсыз; сонымен қатар сұрақтар мәтіндерден емес, жеке сөйлемдерден жасалады.

Бұл мақалада мен берілген мәтіндік материал негізінде сұрақтарды құру үшін NLP технологияларын (заманауи табиғи тілді өңдеу) қолданатын қолданбаны, автоматты мәтінге негізделген тест жасаушыны (TestGen) және осы сұрақтарға автоматты тексеру жүйесін ұсынамын. TestGen сұрақтарды жасау және тестілеу әрекеттерін пайдаланушыға ыңғайлы ету үшін көптеген жаңа мүмкіндіктерді біріктіреді.

Түйін сөздер: сұрақ қою, табиғи тілді өңдеу, білім беру, тест арқылы бағалау, тілдік модель

Аннотация

Генерация логически обоснованных и релевантных вопросов из текстового материала, а также их проверка является обычным, но жизненно важным действием для многих сфер образования.

Оценка имеет решающее значение в любой системе обучения. Система оценивания может выявить пробелы в самообучении учащихся и помочь им развиваться в процессе обучения. Ручная генерация вопросов вместе с их проверкой требует больших усилий, поскольку требует чтения, разбора, понимания и запоминания длинных отрывков текста. В результате основная цель автоматизированной системы оценивания состоит в том, чтобы автоматически генерировать вопросы из учебных ресурсов.

В последнее время в исследовательском мире были предложены различные системы генерации вопросов, основанные на нескольких типах задач языкового моделирования, особенно на прогнозировании от последовательности к последовательности. Однако большинство таких систем неудобны для пользователя; также вопросы создаются из отдельных строк, а не текстов.

В этой статье я предлагаю автоматическую генерацию тестов на основе материала для чтения (TestGen), приложение, которое использует современные технологии обработки естественного языка (NLP) для создания вопросов на основе предоставленного текстового материала и автоматической системы оценки для этих вопросов. TestGen сочетает в себе множество новых функций, чтобы сделать создание вопросов и действия по тестированию удобными для пользователя.

Ключевые слова: генерация вопросов, обработка естественного языка, образование, оценка, языковая модель

Contents

Nomenclature

LM Language model

NER named entity recognizer

NLG Natural Language Generation

NLU Natural Language Understanding

QG Question Generation

$SQuAD$ Stanford Question Answering Dataset

$UniLM$ Unified Language Model

VQG visual question generation

1. Introduction

1.1 Motivation

In the past, people have generally depended on their capacity to ask insightful questions in order to evaluate a user's level of understanding regarding a specific body of information (a comprehension, an article, etc.). Nevertheless, coming up with questions one at a time is a challenging task to undertake. What kinds of solutions are there for dealing with this issue? Automated question generating (QG) systems might be useful in this regard because they teach users how to effectively construct questions in big quantities. In the course of this study, my objective is to automate that process. The difficulty of automatically generating factual questions from specific texts is something that particularly piques my curiosity.

Natural language understanding and generation are necessary in question creation[1]. As detailed above, this is particularly applicable to a system that uses NLU to analyze raw text as input, and then uses NLG to translate the intermediate representations into English language queries [2]. NLG, according to McDonald, is the act of putting thoughts into words [3]. A QG system has several uses, including FAQ generating, intelligent teaching systems, automated reading comprehension, and virtual assistants/chatbots.

These intricate and varied interpretations of questions highlight the significance of questions and, by extension, the goal of automatically generating them. Later on in this dissertation, questions will be discussed, and the relevance of questions will be underlined.

In recent years, an increased number of end-to-end neural network designs have been constructed to perform tasks including question answering. On the other hand, conventional approaches are predicated on syntactic and semantic analysis,

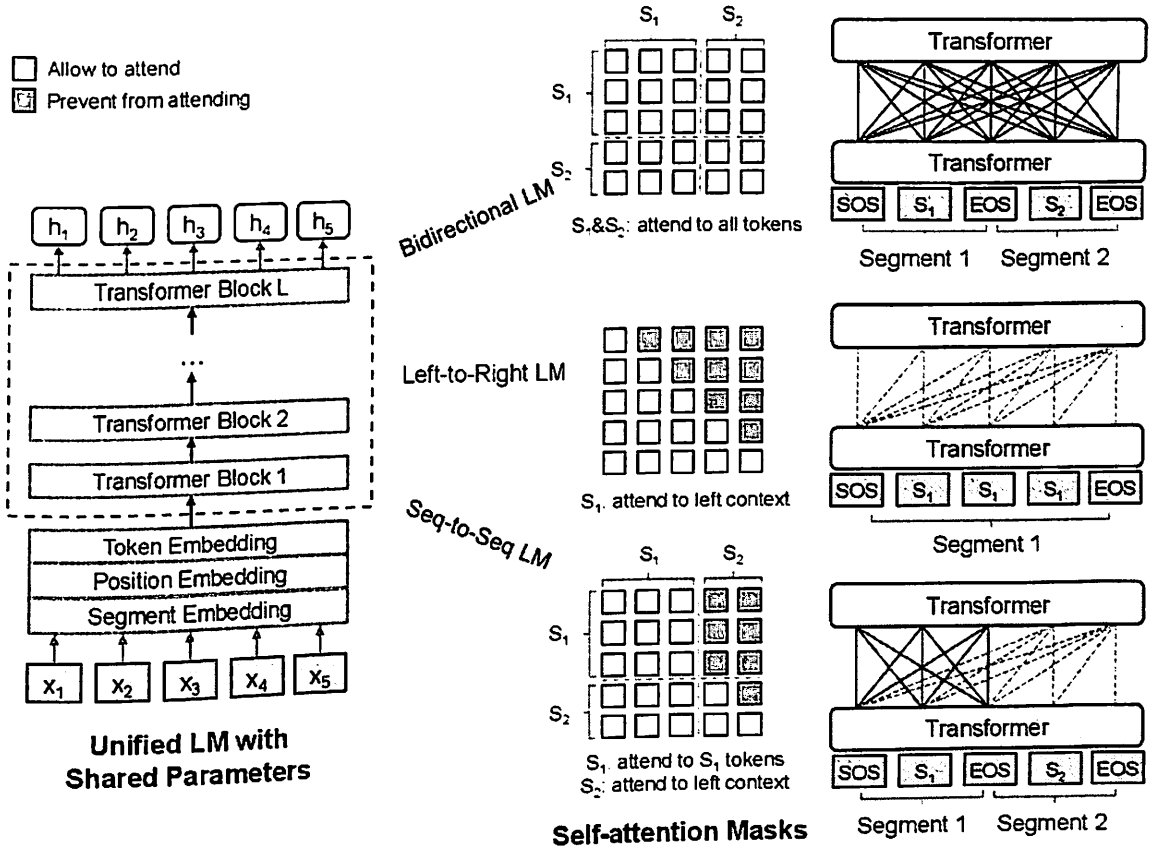

Figure 1.1: Overview of UniLM pre-training [4]

in addition to created qualities. In order to obtain more precise findings, end-to-end neural network methodologies are utilized. On the other hand, conventional methods are simpler to grasp than their modern equivalents.

The SQuAD - Stanford Question Answering Dataset - is used in this study. SQuAD takes into account queries that were asked by individuals on Wikipedia articles. The solution to each question can be found in the corresponding article on Wikipedia that has been linked. SQuAD has around one hundred thousand question-answer pairs that are dispersed among five hundred or more articles.

This study presents a question generating system that makes use of both SQuAd and UniLM in addition to both of those systems. The abbreviation for "Unified Language Model" is "UniLM". There are three main kinds of unsupervised language modeling goals for UNILM, which has been pre-trained on enormous amounts of text. Because of the preliminary training done using language models, a wide array of natural language processing applications have made enormous strides in terms of advancing technology (LM). Overview of UniLM pre-training is shown in Figure 1.1

Researchers of UniLM carried out experiments with the intention of coming up

with answer-aware questions. They sought to devise a question that, depending on the sequence of events and the amount of time that was provided for responding to it, would evoke a particular response from the participants. For purposes of evaluation, the SQuAD dataset is utilized. In the following chapter, a more in-depth discussion of the SQuAD dataset is going to be presented. In order to preserve the development set, the testing and training sets from the initial data collection were split apart. After the data has been divided, the trials are conducted using the development-test split that has been turned upside down. [5].

A sequence-to-sequence problem is used to describe the task of question development. First, the paragraph and the response are combined in the first section, followed by a new question that has been generated. With batch size set to 32 and masking probability at 0, the learning rate is set to 2e-5 as the developers fine-tune UNILM on the training set for 10 epochs. 0.1 is the smoothing rate for the labeling. Training hasn't changed the remaining hyper-parameters in any way. By selecting a passage segment that contains the answer, the input is reduced to 464 tokens throughout the decoding process. Given an input sequence $x = x1...x|x|$, for each token, UNILM generates a contextualized vector representation.

Figure 1.1 shows how pre-training improves the shared Transformer [6] network's performance on a variety of unsupervised language modeling tasks, including single directional language model (LM), bi-directional LM, and seq-to-seq LM. To control who has access to the expected context of a word token, creators use a variety of self-attention masks. In other words, the token's contextualized representation is computed based on how much information it should focus on via masking. It is possible to fine-tune UNILM using task-specific data once it has been pretrained. All LM targets have the same set of model parameters (i.e., bidirectional LM, unidirectional LM, and sequence-to-sequence LM). The authors devised a variety of self-attention masks in order to manage the access to context for each word. An analogy may be drawn between a right-left LM and the left-right one.

With the help of cutting-edge Natural Language Processing (NLP) technology, this article aims to construct a QA system on SQuAD that generates and evaluates questions depending on input text content. Here is a Figure 1.2 showing how the question generating mechanism works. It's a way of expressing the logic of the

system.

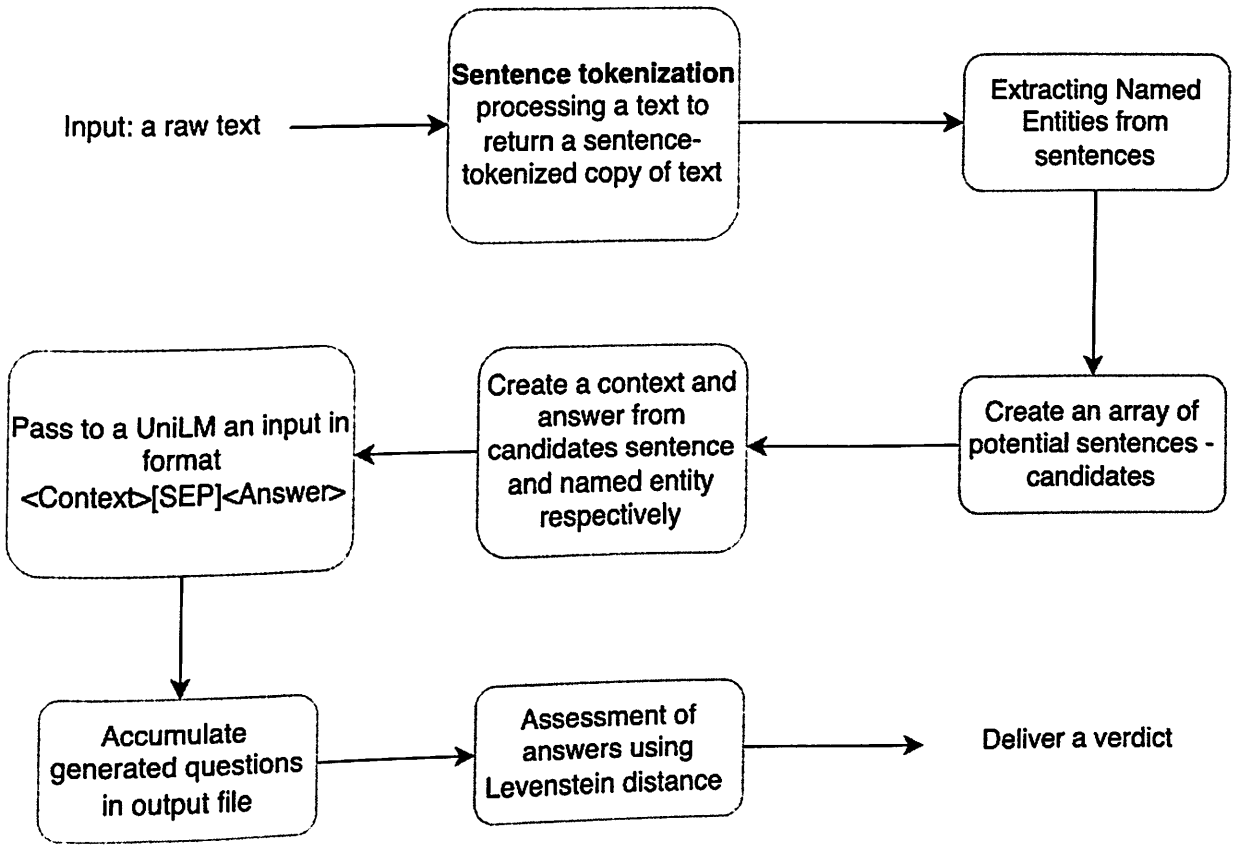

Figure 1.2: Structure of question generator and assessment

To begin, the user inputs the text, which is then processed to generate the tokens for the phrase. Tokenization of sentences will be accomplished by utilizing nltk libraries for tokenization. This stage is essential since only a sentence can be processed in order to form a question, and text cannot be handled in this manner. The SpaCy library will be utilized in order to obtain the sentence's subject because it possesses the method to extract the Named Entity (NE). At some point in the future, the potential solutions to the query that will be formed will be named entities.

The second important step is to create a set of candidate questions consisting of a sentence and a response. These sets are termed candidates. A string with the format 'Context>[SEP]' and 'Answer>' is what UniLM expects as its input. As was said before, a Context is a phrase that was gleaned through the process of sentence tokenization, and an Answer is a named entity.

In conclusion, a model generates questions after processing an input consisting of an array of sentences and replies taken from an input text. In this particular

scenario, the questions are saved in an output file, and later on, as an added functionality, they will be used in the evaluation to compare the user's answer against the given answer by making use of the Levenshtein distance.

The Levenshtein distance is a method that determines how closely related two words are when they are contrasted with one another. The closer the relationship between the two things is, the shorter the distance between them, as measured by the number that is returned. When we look up "helo" in the dictionary alongside other words, we find that "hello" is the most likely candidate for the correct answer. According to what was stated in the paragraph before this one, the Levenshtein distance was applied in order to evaluate the responses.

Following are the examples of a paragraph and questions that would be generated from a question generation system:

Paragraph 1: "The Second World War - also referred as the World War II - a worldwide conflict that took place in period of 1939 to 1945 and was referred to as the Second World War. The Allies and the Axis were constituted by the overwhelming majority of the world's nations, including all of the world's major powers."

- Question 1: What does WWII stand for?
- Question 2: How long did World War II last?
- Question 3: What were the two opposing military alliances?

Paragraph 2: "The invasion of Kievan Rus' by the Mongols in the 13th century largely obliterated the city of Kyiv, which had been founded in 1240. Near the end of the century, the vast majority of Bulgar tribes dispersed and were replaced by Khazars, who eventually acquired control of a significant portion of the region."

- Question 1: What was destroyed by the Mongols?
- Question 2: By whom the land was inhabited, during the Iron Age?
- Question 3: Who migrated in different directions?

Paragraph 3: "People become infected with COVID-19 when they breathe in polluted air that contains droplets and minute airborne particles that are carrying the virus. During the early stages of the epidemic in Wuhan, the infection and

the sickness were often called to as "coronavirus" as well as "Wuhan coronavirus," with the disease also being referred to as "Wuhan pneumonia" at times. People can continue to spread the virus for approximately 20 days after they stop showing symptoms, during which time they are still considered contagious."

- Question 1: How COVID-19 transmits?
- Question 2: For how many days people remain contagious?
- Question 3: Where the initial outbreak took place?

1.2 Aims and Objectives

This work aims to develop a question generation system that, given a piece of input text, is capable of producing a number of factual questions based on material obtained from an external source (which might be any source of factual information). After a user has picked and updated these questions, they can then use them in practice trainings or as part of a test to determine whether or not students have reviewed and remembered the material contained in the content. My efforts to generate questions are primarily concentrated on reading nonfiction literature because they present facts rather than opinions. It would be interesting and illuminating to create questions relating to personal writings and anecdotes, but these subjects will be covered in the future. Instead of modeling more specific lexical and syntactic elements of investigations and the demonstration of actual facts, I built a general-purpose system for producing factual inquiries that leaves out information that is specific to the subject being investigated. The following are the research's primary objectives:

- Use an UniLM model for question generation;
- Prepare input using Named Entity Recognition (NER);
- Use Levenshtein distance for test assessment;
- Build a web server using Django

1.3 Thesis Outline

The "Introduction" chapter is the first one that is presented in the book. This is the one that you are reading right now. It gives a feeling of how the work was performed while also providing an overview of the goals and objectives of the project. In the chapter titled 2. I discuss various publications, projects, and research that I believe to be interesting. Related publications do address approaches for the creation of questions; however, rather than examining explicitly language model based techniques, it will delve more deeply into the many sorts of generation. In addition, I will do an analysis for the dataset, during which I will cover multiple language models. In the following subsection, we will delve even further into the structure of the dataset that was developed by academics at Stanford University. The answer to the predicament is detailed in the chapter referred to as 3. To put it another way, how exactly the question is going to be formed, and what sort of technological foundations it will have. In addition to that, the criteria for a successful compilation will be stated. It will be shown how to implement everything in Chapter 4 including the web server, back end, and front end processes. A successful work effort necessitates the utilization of contemporary practices that are designed to produce a stable platform in both the front end and the back end of the system. The front-end does play an essential part in the interaction with the application, which results in the creation of a convenient testing tool. The outcomes of the implementation are presented in Chapter 5 as well as future plans for the work. The results will demonstrate the capabilities that a user can expect to obtain from the final product, which is the functional constituent of the test and the question generator. In the last part of the study, which is called the Conclusion chapter, we summarize all that has been discussed thus far in the work by evaluating and analyzing each stage.

2. Literature review

The review of the relevant literature for the question creation exercise consists of four main sections.

There have been some preliminary attempts made toward the generation of questions in this area. In the first chapter, we will go back to the early studies that were done in the 1970s. At that time, researchers were only able to construct wh-questions from single sentences and other studies.

The second part of this section is made up of related works that have been organized according to the method of investigation used.There are three different approaches that can be taken when generating questions: the objective approach, the subjective approach, and the visual approach.

The SQuAD dataset will be discussed in the third section of this article. There will be a description of other specifics, such as annotations, analyses of datasets, samples, and categories of answers. In addition, the factors indicated behind the choice of this dataset would be accounted for in a number of different language models.

In this final portion, the language models GPT-3, BERT, and UniLM will be the primary topics of discussion.

2.1 First attempts in the field of question generation

The development of automated questions and answers is challenging. Since Wolfe et al. [7] revealed their AUTOQUEST system in 1976, generating Wh-questions from single sentences has been a challenge for researchers. After this, a large number of citation-based models were utilized, including those based on pattern matching and linear regression. This kind of system is extremely reliant on rules

or question templates, necessitates significant language comprehension, and is not sufficiently exhaustive [8]. The most recent advancements in neural machine translation have made it possible for deep neural networks to learn implicit rules from data [9]. The use of sequence-to-sequence learning as a method for automating the production of interview questions has gained popularity in recent years.

2.2 Related works by question generation techniques

There are two major classes of questions that can be split up into the objective and subjective categories, respectively. This objective question does not present students with a choice between two alternatives; rather, it asks them to suggest either a single phrase or a string of words that can be used to solve the issue that is being discussed.

The majority of the questions on school exams are from the multiple-choice, true-false, matching, and fill in the blank categories. On the other hand, the subjective nature of a student's question necessitates an explanation that enables the student to develop and compose their own response. Both short-answer questions and long-answer questions are examples of the subjective forms of queries that are asked the most frequently. On the other hand, a good classroom test should have an equal number of subjective and objective questions.

On the other hand, we will discuss visual question creation in this section, which is a form of question generation. Because technology is getting better, more and more people are using computer vision to make questions.

2.2.1 Objective question generation

According to the findings of the literature review, the majority of researchers focused their attention on developing objective types of inquiries. In their experiments, the researchers asked participants only questions that were either multiple-choice or closed-ended. There isn't much research done on topics with no clear answers.

Pino and Eskenazi (2009) [10] supplied the clue inside an open-cloze question. They observed that the initial couple letters of a word that was missing provided a

hint about the term that was missing. Their objective was to adjust the amount of letters in the suggestion in order to pose questions of varying degrees of difficulty in order to make it easier for kids to expand their vocabulary.

Das and Majumder (2017) [11] proposed a way for generating open-ended questions to assess learners' knowledge. These questions were made by an automated system that makes only fill-in-the-blank options for factual open-cloze questions. First, the algorithm pulls a group of useful phrases out of the input corpus. Focusing on part-of-speech tags as well as other requirements, the phrases are considered to be informative. Then, questions are made by leaving out the answer keys, which are chosen by looking for phrases inside these sentences that are specific to the domain. An open-cloze question's unbound choice set commonly confuses test-takers. On the other hand, open-cloze questions need students to have more information in order to be more effective than cloze questions. Finally, in order to reduce the number of possible answers and make grading simpler, Das and Majumder have provided answer suggestions for candidates.

Coniam (1997)[12] proposed an approach that is one of the oldest strategies for the generation of cloze test items. It was through the use of word frequencies that he analyzed the corpus in its many stages of development, like in order to collect the keys to test items, produce test item alternates, create cloze test items, then determine which test items were good and which were poor. In order to design test items, he began by matching the parts-of-speech and word frequency of every given test key with a word class that contained a word frequency that was comparable.

Chen et al. (2006) [13] created a program that is partially automated for generating grammatical test items by utilizing NLP methods. In this work, Natural Language Processing tools are used to make grammar test questions. Sentences acquired from the web are turned into grammatical exams based on carefully established patterns. Encoding test writing knowledge as test patterns, collecting real sentences from the Web, and applying generation processes to translate sentences into objects are all part of the strategy. Sentences are translated into two sorts of TOEFL-style questions at runtime: multiple choice and mistake detection. Their method entailed handcrafting patterns to discover actual phrases and distractors on the internet, which were then transformed into grammar-based exam questions. According to their trial results, the strategy generated 77 percent

relevant inquiries.

Cloze test items from online news articles were created by Hoshino and Nakagawa in 2007 [14] using a semi-automated technique. Students had to supply the missing words from a passage that was used in a test item missing one or more of them. The grammar and vocabulary distractions produced by their method were the outcomes of their strategy. It was found that their system was responsible for producing 80 percent of the problems worthy of testing.

Using a biology textbook, Agarwal and Mannem (2011) [15] showed how to produce gap-fill questions. The number of pronouns and nouns in a phrase, abbreviation or superlatives, and sentence length were all factors in the creation of the questions. These features were taken into consideration as the questions were being drafted. Fill-in-the-blank questions are comprised of one right answer and three incorrect alternatives. By first blanking out the keys in the sentences and then detecting the distractor phrases, the algorithm finds instructive sentences to use as the basis for gap-fill questions. No external resources are required for this strategy, which uses syntactic and lexical properties. In terms of how these characteristics were weighted or blended, they didn't go into great depth.

Multiple-choice questions from a language textbook may be generated using a semi-automated method, according to Mitkov et al. (2006) [16]. Their NLP techniques included a wide range of approaches. In addition to traditional NLP techniques such as shallow parsing, automatic word extraction, sentence modification, and semantic distance calculation, the system uses corpora and ontologies as language resources. In addition to creating questions and multiple-choice distractors based on the text's core ideas, it provides the user with an easy-to-use interface to alter the test items.

An automated method for creating multiple-choice questions based on Wikipedia has been developed by Bhatia et al (2013)[17]. In work, theyprovide a way for generating multiple-choice examination questions with Wikipedia. Using existing online test questions, they provide a strategy for potential phrase selection in this study. Inferred from the present questions, the sentences are picked based on a pattern. In addition, Bhatia et al. provide a novel way for constructing distractors for named entities. To produce high named entity distractors, they get certain additional parameters out from web and search Wikipedia for entities to the same attribute values. The sports world is where they do their research.

A group of evaluators uses a set of criteria to judge the questions and the things that distract from them. Based on what the investigation found, the method is pretty accurate.

Nominated Entity Recognition and Syntactic Structure Similarity were utilized to choose MCQ production phrases by Majumder and Saha (2015) [18]. Topic modeling as well as parse tree - structured similarity were used to find informational terms for inquiries. Based on subject and the named-entity, a gazetteer list was utilized to choose keywords and distractors.

2.2.2 Subjective question generation

In the literature, there are just a few studies that concentrate on the generation of subjective questions.

It was hypothesized by Deena et al. (2020) [19] that an NLP and Bloom's taxonomy-based question generation system might dynamically generate subjective questions while lowering memory consumption. First, a named entity recognizer is used to make questions with distractor sets. Additionally, Blooms taxonomy-based natural language processing is used to construct the Subjective questions (NLP). The goal of the proposed mechanism is to come up with questions on the fly, which helps people remember more ideas.

The fundamental problem of subjective evaluation is proper scoring. An educational institution's current research focus is thus on automated subjective-answer assessment [20]. It saves time and energy in the education system by cutting down on evaluations. Evaluation of objective-type answers is simple and only needs a binary system to figure out where the choice is best. However, due to its complexity, subjective response evaluation does not produce appropriate results. Next, we'll look at several research on subjective answer evaluation and methodology.

Leacock and etc. in 2003 [21] presented a C-rater response grading system that interacts with text semantic information. Short answer questions with topic-based material are assessed using C-rater, an automated scoring method. In addition to string matching, it utilizes predicate argument structure, pronoun reference, morphological analysis, and synonyms to give credit to a short answer. Several studies have made use of the C-rater. With the manual review of human graders, their method attained 84 percent accuracy.

Bin et al. (2008) [22] used the text classification model and the KNN classifier is used to grade essays automatically. Each essay was expressed using the Vector Space Model. Essay qualities were represented by TF-IDF weights for words, phrases and arguments. It was found that the cosine similarity strategy was used to create essay scores, and it was shown to have an average accuracy of 76 percent using various feature selection approaches, such as TF-IDF, information gain, and word frequency (IG).

Using three alternative approaches - Probabilistic and Latent Semantic Analysis and Dirichlet Allocation - Kakkonen with colleges in 2008 [23] suggested a essay grading system that makes a comparison of learning materials to teacher-graded essays. Before grading, it uses learning materials and a limited number of teacher-graded essays to calibrate the scoring process.In AEA, they ran several studies with LDA, LSA and PLSA for document comparisons. As part of their evaluation of the approaches, they looked at how they may be used in practice for essay grading. K-NN grading methods were shown to be inferior to learning resources used as training data for the evaluation model, according to these results. Furthermore, they discovered that utilizing LSA produced somewhat more accurate grading than PLSA and LDA. They also discovered that the division of the learning items in the training data is critical. It is preferable to split learning information into sentences rather than paragraphs.

Using the M-BLEU short for Bilingual Evaluation Understudy algorithm, Noorbehbahani with co-authors in 2011 [24] provided a system for analyzing students' text responses. For free text responses, they modify the BLEU algorithm and name the new technique modified BLEU. A library of reference answers compiled by academics or other specialists is needed to perform an evaluation. There are a variety of citations available for each question. Using the M-BLEU approach, a score of similarity is produced then used to assess student responses; the most comparable reference response is found. When compared to existing evaluation approaches, the suggested methodology has the closest resemblance to human expert assessments like latent semantic analysis and co-occurrence of the n-gram.

For assessing the student's reaction, Dhokrat et al. (2012) [25] designed an evaluation approach. For analyzing a student's answer, they employed a centralized file system that comprised both the reference material and the model

solution for each question. When the needed selected pattern of every right response agreed with the model-answer, a positive confidence factor was awarded. The efficiency of their system was found to be 70 percent.

Islam and Hoque (2010) [26] introduced a generalized latent semantic analysis-based computerized essay grading system (GLSA). One of the most pressing issues in educational technology right now is AEG (Automated Essay Grading). A method so called Latent Semantic Analysis, LSA is used in automated essay grading. Singular Value Decomposition (SVD) is used for decompose the matrix of the word that LSA develops. The current AEG structure, which is generally based on LSA, does not meet the performance standards necessary to mimic a human grader. Instead of a word-by-by-word matrix, they developed a system based on Generalized Latent Semantic Analysis. They used a detailed depiction of the system's performance to examine it and illustrate its efficiency. Their system outperforms the current system in terms of performance.

An original method for evaluating brief responses was described by Ramachandran and colleagues in a publication that was released in the year 2015 [27]. Word ordering graphs were used to do an analysis on the students' best replies to the handicraft rubrics.

Content-based short responses were graded by Sakaguchi et al. (2015) [28] utilizing information from a variety of sources. The responses were put through a procedure that highlighted particular characteristics before being examined. After utilizing the reference-based technique, it was discovered that the answer quality and the scoring systems shared many characteristics in common with one another. During training, their model performed better despite having less data points. It is now abundantly evident that in the not-too-distant future, response assessment might make use of a kind of NLP that is predicated on deep learning.

A potential future for response assessment may be seen in the development of deep learning-based NLP. NLP's general representation of sentiment allows Abdi and colleagues (2019) [29] to be used in numerous scenarios.According to recent findings, using feature vectors that integrate a statistical knowledge component, linguistic information, and sentiment shifter rules, in addition to word embedding, may prove to be an effective method for enhancing sentiment analysis. They outperform competitors who base their learning on a more limited set of attributes because of the methodology they apply. According to the research that has been

done, the neural model is superior to other well-known methods in terms of its effectiveness.

2.2.3 Visual question generation

To produce image-based questions, the area of computer vision has recently added question creation. To train machine learning algorithms to generate several questions from a single picture, the most recent solutions have relied on labor and time-intensive human annotations. According to Zhu et al. (2016) [30], citing Zhu, seven wh-type inquiries were manually generated, including when and where. They used object-level grounding to create a semantic relationship between written descriptions and picture areas. It allows for a new form of QA utilizing visual replies in addition to the traditionally utilized textual responses. These visual QA tasks are examined in a realistic context using various 7W multiple-choice QA pairs. On the QA tasks, they also evaluate human performance and a wide range of baseline models. As a last resort, they propose using an LSTM model that incorporates positional awareness for 7W quality assurance tasks. People also considered using criteria to produce automated visual questions.

An image caption may be transformed into an effective inquiry by deleting content words (answers) and reformatting the caption phrase as a question, according to a study by Yu et al. (2015) [31]. A novel dataset of 360,001 condensed descriptions in natural language for 10,738 pictures is also presented. Automated templates for filling in the blanks are used to create the Visual Madlibs dataset, which aims to record detailed descriptions of individuals and objects in the scene, including their appearances, movements, and interactions. For two innovative description generating tasks — focused description production and multiple-choice question-and-answer for images — they perform several assessments on the Visual Madlibs dataset and demonstrate how it might be used. Deep learning algorithms have shown promising outcomes in this area of research so far.

Furthermore, Ren et al. (2015) [32] presented a rule that would transform picture captions into specific types of questions. Concerns about how models might affect question variety were raised by some. The problem concerning image-based question answering is being solved using new models and datasets (QA). The use of neural networks as well as visual semantic embeddings instead of object recognition and image segmentation to predict answers to basic picture queries

is being considered by some. By a factor of 1.8, their method surpasses the only other strategy based on results from an existing image QA dataset. They also offer a technique for generating questions based on generally available image descriptions. They were able to build a dataset that was many orders of magnitude bigger and more evenly distributed by using their technique. This new dataset also contains a collection of first results.

One picture may be used to generate a large number of visually-based inquiries, as proposed by Zhang and colleagues (2017) [33]. If you're interested in learning more about the topic of visual question generation, here's a primer: For the same visual input, there is no automated technique to generate appropriate questions of various kinds. They propose an approach that automatically generates visually based questions of a variety of types. With the use of a deep caption model, they sample the most probable question types and sequentially generate questions based on this data. They observed that their model outperformed the strongest baseline in terms of accuracy and variety by a substantial margin using two real-world datasets.

Many questions were generated for each image in the initial dataset gathered by Mostafazadeh et al. (2016) [34]. They offer a new activity called Visual Question Generation (VQG), when provided an image, the system is tasked with asking a genuine and interesting inquiry. They present three datasets that cover a range of pictures from object-centric to event-centric, with far more abstract training material . They train and evaluate multiple generating and retrieval models to deal with VQG. The evaluation findings demonstrate that, while such models ask fair questions for a wide range of images, there is still a significant gap with human performance, motivating additional research into integrating images with commonsense knowledge and pragmatics. Their suggested goal presents a fresh challenge to the community, which they believe will increase interest in investigating deeper relationships between vision and language.

Using the Densecap architecture given by Johnson et al. (2016) [35], the supplemental information required to oversee question production, region captions may be created. When just a single word is used to describe an item, Dense Captioning and Picture Captioning may be used to help identify it more broadly. FCLNs may be trained end-to-end with one round of optimization to meet both the localization and description problems concurrently by processing an image

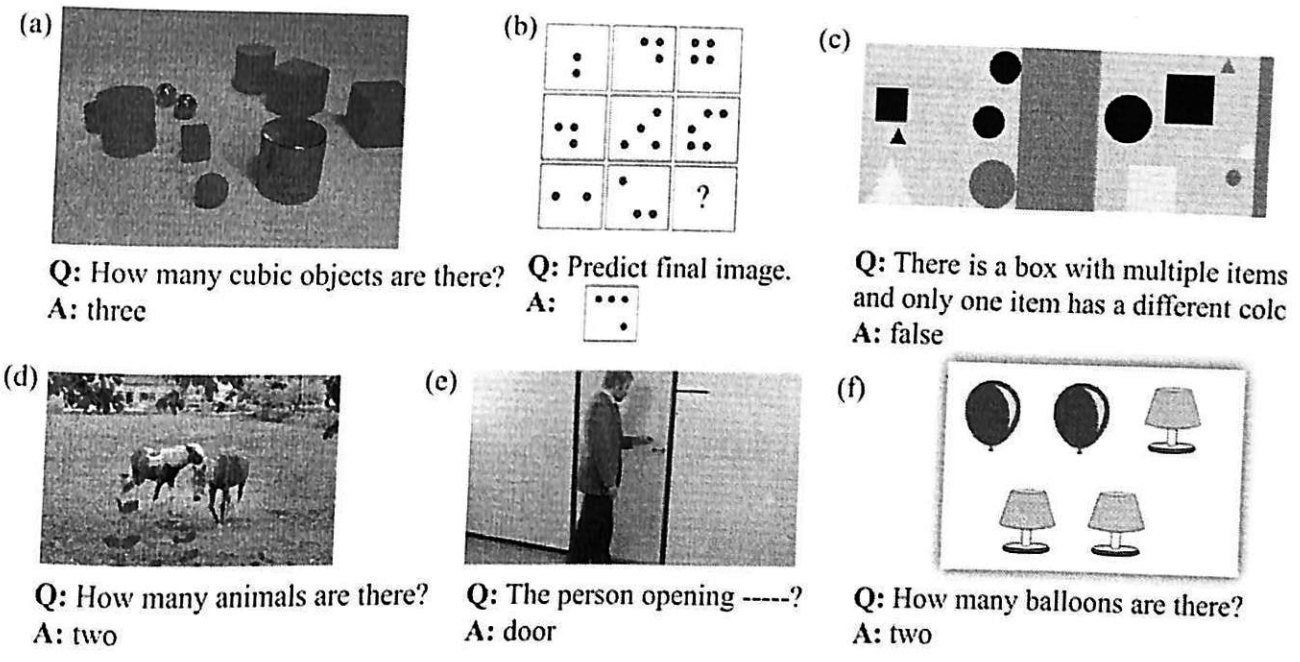

Figure 2.1: Various datasets for visual question generation and answering.[35]

inside a single, effective forward pass without the need for external region recommendations. Three different types of neural networks create the architecture's label sequences: convolutional, dense localization, and linguistic recurrent neural networks. To put their network to the test, they employed the Visual Genome dataset, that contains 94,000 pictures and 4,100,000 region-based descriptions. Compared to existing state-of-the-art approaches, they perform faster and more accurately in both the creation and retrieval contexts. A Figure 2.1 shows variations of questions regarding to image contents.

Variational auto-encoders and LSTM networks were utilized by Jain et al. (2017) [36] to construct several different queries from a single picture. Images taken from the real world were essential to the vast majority of these questions and the responses to them. They show that a single input photo can result in an extremely large number of different questions being asked by their algorithm.

2.3 Related datasets

Techniques for natural language processing (NLP) are presented with a significant obstacle in the form of question and answer pairs derived from the Stanford Question Answering Dataset (SQuAD).

In keeping with its name, SQuAD focuses on answering questions. A model's reading and comprehension skills are tested in this thesis work. As a rule, it's

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 2.2: Examples of questions and answers from the SQUAD dataset for a passage. The dataset's original developers, Rajpurkar et al. [37], are given image credit.

a really straightforward procedure. in Figure 2.2 is an example provided by the dataset's developers.

The creation of SQuAD

The top 10 000 Wikipedia entries were chosen by SQuAD's creators. Their search yielded 23 215 different paragraphs, all carefully filtered to exclude those that were too short. When it came to training and testing, they broke the dataset down into 80 percent of each article, 10 percent of each article, and 10 percent each of each article.

Annotating SQuAD

Mechanical Turk volunteers handled the most crucial aspect of constructing a dataset: annotating it. Employees with a proven track record of generating high-quality work were solely considered. When a paragraph was chosen, workers were asked to develop and answer five questions concerning its content. A text box was provided for their inquiry, and they were allowed to mark the paragraph

comments that responded to it. In order to ensure that the employees' inquiries were submitted in their own words, the SQUAD architects blocked the copy-paste option.

Dataset analysis

Understanding the qualities of a dataset is a critical component of creating a strong one. To do this, the designers investigated three areas: response categories, reasoning needed, and syntactic divergence.

Regarding the answer responses, each response was catalogued according to one of the following categories: "date," "other numeric," "person," "location," "other entity," "common noun phrase," "adjective phrase," "verb phrase," or "clause," or "other." There were 19.8 percent of replies that included dates and numbers, 32.6 percent that included nouns, and 31.8 percent that included nouns and noun-noun pairs.

The developers choose a selection of questions at random from the development set, and then manually segmented the problems into the several logical domains that were necessary for the resolution of those issues. For instance, the term "syntactic variety" indicates that the question is nearly rephrased and that the answer can be uncovered by rearranging the words in the query. Original table with all categories and percentages for each group is displayed in Figure 2.3.

The degree to which a response phrase differed syntactically from a query was one of the factors that was considered when evaluating the difficulty of a question. To put it another way, they came up with a system for counting the number of adjustments that must be made in order to translate a query into an answer. The primary research may be found at this link, however the finding that the sample contained a large variety of syntactic variance is the most relevant takeaway. Figure 2.3 shows the many types of reasoning that are required.

Benefits of SQuAD

A task such as question answering can obviously make use of any number of different datasets that are available. When compared to other datasets, SQuAD has a few key distinctions that set it apart.

In the first place, SQUAD is a very large organization. The other datasets that measure reading comprehension are insufficient to enable complicated models

Reasoning	Description	Example	Percentage
Lexical variation (synonymy)	Major correspondences between the question and the answer sentence are synonyms.	Q: What is the Rankine cycle sometimes **called**? Sentence: The Rankine cycle is sometimes **referred** to as a practical Carnot cycle.	33.3%
Lexical variation (world knowledge)	Major correspondences between the question and the answer sentence require world knowledge to resolve.	Q: Which **governing bodies** have veto power? Sen.: **The European Parliament and the Council of the European Union** have powers of amendment and veto during the legislative process.	9.1%
Syntactic variation	After the question is paraphrased into declarative form, its syntactic dependency structure does not match that of the answer sentence even after local modifications.	Q: What Shakespeare scholar **is currently on the faculty**? Sen.: **Current faculty include** the anthropologist Marshall Sahlins, ..., Shakespeare scholar David Bevington.	64.1%
Multiple sentence reasoning	There is anaphora, or higher-level fusion of multiple sentences is required.	Q: What collection does **the V&A Theatre & Performance galleries** hold? Sen.: **The V&A Theatre & Performance galleries** opened in March 2009. ... **They** hold the UK's biggest national collection of material about live performance.	13.6%
Ambiguous	We don't agree with the crowdworkers' answer, or the question does not have a unique answer.	Q: What is the main goal of criminal punishment? Sen.: **Achieving crime control via incapacitation and deterrence** is a major goal of criminal punishment.	6.1%

Figure 2.3: Each kind of reasoning needed to solve a question is represented by an example from the development set. The dataset's original developers, Rajpurkar et al. [37], are given image credit.

that require a significant amount of data. The MCTest only has a total of 2,640 questions, while the Deep Read only has a total of 600. The SQUAD database, which contains approximately 100,000 questions, is the most prominent of these datasets.

On the other side, SQuAD is a system that is quite sophisticated. Other answer extraction-based QA datasets contain a huge number of documents that contain the response to a particular query that was provided. However, you are able to access a route while using SQuAD, which makes the process more challenging because you cannot afford to make any mistakes.

Last but not least, reasoning is necessary for SQuAD. The kind of dataset that requires you to supply the missing word in a passage is an illustration of a cloze dataset. There is a possibility that these enormous datasets have a problem that is analogous to SQuAD. It is beneficial to use SQuAD to evaluate model knowledge and abilities since its responses are more complicated, and as a result, require more reasoning. This makes SQuAD a more powerful evaluation tool.

It is apparent that SQuAD is one of the most famous datasets for answering questions as a result of the over 2,000 times it has been cited. This is due to the fact that SQuAD is well-designed. Since SQuAD is an excellent dataset that can

be used to evaluate not just how well NLP models comprehend language but also how well they function in general, anyone who is interested in doing so can make use of it.

2.4 Language models

The process of gleaning information from a myriad of sources has seen significant development over the course of the past decade. Since the term Natural Language Processing has replaced Text Mining as the name of this field of study, there has been a significant shift in the approach that is used. One of the key drivers of this transition was the emergence of language models as a foundation for numerous applications that were aimed at extracting important insights from raw text.

Probability distributions over words and word sequences are all that are required to create a language model. When a language model is put to use in the actual world, it may be able to predict the chance that a particular string of words is "valid." In this particular instance, the word "validity" does not refer to the correct use of grammar. It gives the impression that it is comparable to how people speak (or, more specifically, write), which is how the language model acquires its knowledge. This is a very important point to keep in mind: a language model, like other machine learning models and especially deep neural networks, is not magical. Rather, it is "simply" a tool that allows for the incorporation of a large amount of information in a condensed fashion that can be reused in situations where there is no training data.

In a deep learning model, each piece of incoming data is weighted differently by a transformer. [6]. Applied to NLP and computer vision, it is commonly utilized. RNNs, for example, are a type of data transformation that is designed to handle sequential data input, such as natural language. This type of input is required for operations such as translation and text summarization. Transformers, on the other hand, may not necessarily process the data in the same sequence as RNNs do when it comes to processing. On the other hand, attention processes offer context for any point in the flow of information that is being considered. If the data is in the form of a natural language sentence, the transforms do not need to concern themselves with the beginning of the phrase until after the phrase has been completely processed. Instead, it creates a context that clarifies the mean-

ing of each individual word in the phrase. Because of this characteristic, which allows for greater parallelism than RNNs, shorter training times are achievable. RNNs cannot achieve this level of parallelism. The transformer concept is used in both the GPT models developed by OpenAI and the BERT models developed by Google. In addition to this, they make use of a system that they call "Attention" in order to determine which inputs deserve greater acknowledgment than others at any particular time.

2.4.1 GPT-3

The GPT-3 is an autoregressive language model that, via the use of deep learning, generates writing that has sentience.

OpenAI's GPT-n series' third generation language forecasting model was cre-
ated in San Francisco by the company.[38] One hundred seventy-five billion pa-
rameters were learned on the largest training corpus, Common Crawl, for the
complete version of GPT-3. This is made feasible in part by a language model's
semi-supervised training technique, which allows a text to be used as a training
example with certain words removed. The extraordinary capacity of GPT-3 stems
from its ability to read almost all content that has emerged on the internet in re-
cent years, and to represent the majority of the complexities inherent in natural
language. The natural language processing (NLP) trend toward pre-trained lan-
guage representations is shown by GPT-3, which came out in May 2020 but was
still in beta testing as of July 2020.[39]

There are both benefits and downsides to using GPT-3's high-quality writing,
which may be difficult to identify whether or not it was produced by a human. 31
OpenAI scientists and developers contributed to the first publication on GPT-3,
published on May 28, 2020. As a result of their findings, they recommended more
research into the potential dangers of GPT-3. "One of the most intriguing and
crucial AI systems ever built," said Australian professor David Chalmers about
GPT-3.[40]

Nobody else has access to Microsoft's "exclusively licensed" GPT-3 model,
which was disclosed on September 22, 2020 as being accessible by means of a
public API that is still available to all users.

According to an article published in April 2022 by the New York Times, GPT-3
is capable of writing original content with the same fluency as a human.[41]

2.4.2 BERT

As a pre-training method, Google developed Bidirectional Encoder Representations from Transformers also known as BERT is a transformer-based machine learning strategy. BERT was created by Jacob Devlin and his Google colleagues, and it was launched in 2018 under their leadership [42]. A year later, towards the end of the year 2020, Google made the announcement that it has started implementing BERT into its search engine for the vast majority of English-language queries. Over the course of the past year, BERT has been used to publish more than 150 different research publications.

Both the BERTBASE and the BERTLARGE come equipped with 12 encoders with self-attention heads that can look in either direction. Both of these models, which together contain more than 3,300 million words, are trained using unlabeled data obtained from BooksCorpus and English Wikipedia. Configurable self-attention heads along with encoder layers make BERT an ideal transformer language model. [6]

The first task was to model the language (BERT was trained to predict 15 percent of words from context), and the second task was to forecast what would follow next (next sentence prediction). Because it was trained to do so, BERT was able to determine, based on the first sentence, whether or not a particular future sentence was likely. As a direct result of the training method, BERT is able to acquire contextual embeddings for individual words. After the computationally intensive pretraining phase is complete, BERT may then be fine-tuned using more limited datasets for specific tasks.[42]

2.4.3 UniLM

In recent years, there has been substantial development in natural language processing, commonly known as NLP. The use of language model pretraining is one of them. This technique "teaches" machine learning algorithms how to correctly comprehend text depictions by having the algorithms predict words based on the situations in which they are found. Systems such as Google's BERT, which are bidirectional in nature (drawing on both the left and right of word context to make predictions), are not well-suited to the job of natural language generation without major modification. This is because such systems draw on both the left

and right of word context to make predictions.

That is why Microsoft Research scientists researched an alternate technique known as UniLM [4], natural language creation may be fine-tuned using this tool, also for interpretation. The model is able to successfully complete sequence-to-sequence, unidirectional, and bidirectional prediction jobs on standard benchmarks. It performs better than BERT, getting the greatest possible results on a sampling of abstract summarization, generative question answering, and language creation data sets. This shows that it is superior to BERT.

At the center of UniLM's multi-layer network are Transformer AI models that have been pre-trained on enormous volumes of text and optimized for language modeling. These models were developed specifically for language modeling. For the benefit of those who aren't aware, a transformer is a group of neurons (or functions) that are connected to one another and whose signals are exchanged and the weights of their connections are varied based on the data that is being input. To put it another way, that's how modern AI systems learn to forecast, but Transformers pay closer attention to the relationship between each and every output element and virtually every input factor. In other words, the relative significance of each is decided on the spot as the situation develops.

According to the findings of the researchers, the pretrained UniLM is analogous to BERT in the sense that it might be adjusted to accommodate a variety of jobs that come later (with extra task-specific layers if required). However, in contrast to the self-attention masks used by BERT, the independent self-attention masks that are used by UniLM allow it to be set to collect context for a variety of language models. Pre-training is standardized across all Transformer networks, allowing for the parameters to be shared among them (previous training data). Because of this, the text representations that are learned will be more applicable to a wider variety of contexts. Overfitting, which occurs when a program models training information too accurately, is mitigated as a result of this approach to every single task.

According to the authors of the study, the researchers employed the free software BookCorpus and English Wikipedia articles to teach UniLM's vocabulary. Together, these resources had a total of 28,996 words. On summarizing (CNN/DailyMail), abstractive summarization (Gigaword), question generation (SQuAD), generative question answering (CoQA), and generative question an-

swering (DSTC7) natural language output data sets. [Summarizing] refers to the process of taking a large amount of text and breaking it down into its component parts. [4]

The team plans to investigate the limitations of their existing method by using text corpora on a "web-scale." They also want to look at the prospect of extending UniLM so that it can support multilingual projects.

3. Research method

In this section, the methods of performing the question generation will be presented. In previous chapter, we have discussed several possible approaches selected in the past. Especially, the language models that have been popular recently in the field of question generation.

First and foremost, it was created based on performances of other language models, as well as presenting the factual outperformance such as applicability of bidirectional, left-to-right, seq-to-seq language models.

Additionally, multiple uses like abstractive summarization, question generation and etc. There is pre-trained model on SQuAD for question generation projects, which makes the UniLM the most applicable and unique to do a choice on.

Therefore, the UniLM has been chosen to be the model to base on in creating a system of automatic test generation and assessment.

3.1 Question generation

Input preparation, pre-trained fine-tuned model launch, output saving, and output display are all steps in the question generating process. It would be logical to separate the part of the test assessment separately. Thus, test assessment part consists of test answer entering, comparison of the entered. text with the answers using Levenshtein distance accurracy method.

An input file is required in a certain format, and an output file containing produced questions is returned. As a separate service, the model is run on a Flask, a micro web framework for Python, to accommodate any future development of the application. Implementing the UniLM model is the only thing it can do. Upon receiving a piece of text, it stores this in a txt file in a certain format and then

runs a model that generates the questions.

In this section, the flask micro-service usage as a UniLM model launcher will be described, followed by the input preparation process description as well as the test assessment part.

3.1.1 Flask micro-service

The flask micro-service has been used to execute the UniLM model as stated above. Micro web framework Flask is a Python implementation of the Flask library. It is referred described as a micro-framework because no specialized tools or libraries are required. It does not include a database layer of abstraction, form validation, or other features that third-party libraries may provide. However, Flask allows extensions that may be added to Flask as if they've been built into the framework itself. Form validation and upload handling, open strong authentication, and other framework-related utility extensions are available. The Flask framework is used by two different applications: Pinterest and LinkedIn.

As a result of Python's vast variety of capabilities, flask is frequently used in machine learning techniques as well as other related fields such as natural language processing (NLP). That's being said, to make an integration between web representation of an application and language model several libraries have been used. List of libraries are following: os, subprocess, Flask from flask, request from flask, jsonfy from flask.

Flask side code implementation can be found in Appendix A.

3.2 Input preparation

Context is the response's context; answer provides the appropriate response to the question, and this format is required for input into UniLM models. SpaCy ibrary has been utilized throughout the development of the system to be able to btain the desired outcome.

Natural Language Processing, shortly NLP can be made easier using SpaCy, Python and Cython module. Based on cutting-edge research, it was built from ie ground up to be used in everyday products.

More than 60 languages may be tokenized and trained on top of SpaCy's exting pre-trained pipelines. Advanced neural network models for text tagging,

NER List	
NER label	Description
PERSON	People
NORP	Nations or religious or political organizations
FAC	Highway, Building, Bridge, airport names
ORG	Companies, institutions, agencies
GPE	City, state, contry
LOC	Mountain ranges, water bodies and others
PRODUCT	Food, vehicles, objects and others
EVENT	War, sports, battle names
WORKOFART	Book, song etc. titles
LAW	Law named documents
LANGUAGE	Language names
DATE	Date, period of dates
TIME	Time not proceeding the day
PERCENT	Percentage along with sign
MONEY	Monetary, units
QUANTITY	Weight, distance and other measurements
ORDINAL	Ordinal numbers
CARDINAL	Numerals

Table 3.1: Table of NERs

parsing, named entity identification, and text classification. This solution includes multi-task learning with pre-trained transformers like BERT, as well as a production-ready training system. MIT-licensed commercial open-source software, SpaCy is free to use and distribute.

Named Entity Recognition (NER) has been used to choose a response. As a subproblem of information extraction, the goal of named-entity recognition is to identify and categorize named entities asserted in textual data into pre-define groups, for example people's names or organizations' names or locations or medical codes or time expressions or quantities or financial values or percentages, etc. Here's a table with all of the SpaCy NER labels displayed (see Table 3.1).SpaCy allows you to obtain the text's named entities, and having those named entities helps you to obtain more realistic input for the UniLM model.

To begin, we eliminate any phrases that lack those NEs. Following that, various heuristics were used, such as selecting phrases with lengths greater than 5, and avoiding selecting stopwords or prepositions as answers. The quality of the questions was somewhat enhanced. Then we randomly select the quantity

required, and for each phrase, we use a few of preceding sentences as background to provide a more complete picture. Following is the piece of code of realization:

```
sentences = nltk.tokenize.sent_tokenize(text)
sentences = [sent for sent in sentences
if len(nltk.tokenize.word_tokenize(sent)) > 5]
namedents = [en_core_web_sm.load()(sent)
for sent in sentences]
candidates = [(sent, ne) for sent, ne in
zip(sentences, namedents) if len(ne.ents)]
candidates = random.sample(candidates,
min(10, len(candidates)))
```

3.3 Using Levenstein distance for test answers assessment

There are several methods for identifying lexical similarities between two texts; the one we'll look at today is Levenshtein distance. The Levenshtein distance was employed for the test evaluation. As an algorithm for determining the difference between various sequences I have used the Levenshtein distance. Two words are said to have a "Levenshtein distance" if the amount of single-character alterations needed to transform one to another is known. Vladimir Levenshtein, a Soviet mathematician who studied this distance in 1965, gave it its name.

It is also possible to refer to Levenshtein distance as edit distance. However, it might also refer to a larger group of distance measures [43]. In many ways, it's similar to the pairwise string alignment method.

String a to string b may be converted to a Levenshtein distance by making as little modifications as possible. The Levenshtein distance was used to compare two texts on a 100-point scale using the free fuzzywuzzy library.

It was suggested that the aforementioned method be used since precise matches might have issues such as incorrect prepositions, missing articles, or typos. It's up to the user whether or not they want to include them, but because the answer was gleaned from the text, the user will be held responsible for including them. When two strings of equal length are compared, the Levenshtein distance does have a

significant impact. Levenshtein distance is a reasonably straightforward concept to comprehend. The following piece-wise formula may be used to represent the Levenshtein distance:

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}\left(\operatorname{tail}(a), b\right) \\ \operatorname{lev}\left(a, \operatorname{tail}(b)\right) \\ \operatorname{lev}\left(\operatorname{tail}(a), \operatorname{tail}(b)\right) \end{cases} & \text{otherwise,} \end{cases}$$

Where a and b are the two input strings, and |a| and |b| are the lengths of their respective strings. Except for the initial letter, the tail of a string an or b corresponds to all of the characters in the string. It specifies the character in a and b at the 0th element where it denotes a[0] and b[0].

Because it typically takes three modifications to turn one into the other, the Levenshtein distance among "kitty" as well as "sitting," for instance, is three.

kitten to sitten (using "s" instead of "k")

sitten to sittin (using "i" instead of "e")

sittin to sitting (added "g").

An "edit" is described as the addition of a character, the removal of a character, or the replacement of a character.

4. Implementation

In this chapter, the implementation of the Test-Gen application will be presented.

First of all, the environment setup will be shown. To accomplish the setup, it was necessary to follow the recommendations of uploading the UniLM model, such as using the docker under Linux system, Ubuntu 20.04 was used to be. specific.

The second section describes the web server. As a web server the Django web framework was chosen. As a matter of fact, it is python-based, which makes it perfect for the implementation. Additionally, the directory structure makes the work done more organized.

The third section will show the front-end side of the application. Overall, the review of the functionalities will be presented along with the pages screenshots.

4.1 Environment setup

UniLM code is best executed using docker on Linux. It is possible to build, run, and administer containers across servers and even in the cloud using Docker, which is a platform for software development. "Docker" may refer to either the instructions and a daemon or indeed the Dockerfile file format, depending on context.

It used to take a server purchase, Linux installation, LAMP stack configuration, and program execution to launch a web application. By putting in place a second server in case your software got popular, you ensured it would not crash due to overwhelming traffic.

Container technology may be divided into three categories:

- Builder - In the world of LXC and Docker, a builder is a tool or combination of tools that is used to generate the containers themselves.

- Engine - the software that runs on a container. Docker's docker command and dockerd daemon are at the heart of this issue.
- Orchestration - a system that is used to manage numerous containers, such as Kubernetes and OKD.

The Docker engine can be beneficial to lone engineers that want a lightweight, clean environment for testing but do not require complicated orchestration.

As a result, Docker was the ideal solution for running the project. The following command was required to execute the Docker:

```
alias=‘whoami | cut -d’.’ -f2 ‘;
docker run -it —rm —runtime=nvidia —ipc=host
—privileged -v /home/${alias}:/home/${alias}
pytorch/pytorch:1.1.0-cuda10.0-cudnn7.5-devel bash
```

The initialization of the docker can be found in Appendix B

Finally, the following command was used to install the UniLM repository as a package in the docker:

```
mkdir ~/code; cd ~/code
git clone https://github.com/microsoft/unilm.git
cd ~/code/unilm/unilm-v1/src
pip install —user —editable .
```

4.2 Web server

The primary responsibility of a web server is to provide users with access to a website's content by storing, processing, and transmitting web pages to them. Django was the framework of choice for the development of the web server. SQlite was utilized as the database engine for this project. It is a library written in the C programming language that implements a high-reliability, minimal, fast, self-contained, and complete set of features. SQLite is pre-installed on virtually all personal computers and mobile phones, as well as being packed with innumerable other apps that are utilized on a daily basis. Relational Database Management System (RDBMS), which keeps all of the user credentials together with the questions and answers. The entire process of evaluating content, preparing input, and generating web views is handled by the web server.

4.2.1 Django server

This advanced Python web application framework, known as Django, supports fast development and a minimalistic, practical aesthetic. Experienced developers built it to take care of the tedious aspects associated with web development, enabling you to concentrate on creating an app rather than inventing the wheel from scratch. Because it's open source, you don't have to pay a dime for it. Django organizes the various sections of the web application using a directory structure. It creates a project and an app folder for this purpose.

Making a suitable project and arranging it aids in keeping the project DRY (Don't Repeat Yourself) and clean. Django uses the filename you supply when creating a directory structure for a Django project. It contains all the files you need to get your web applications up and running.

The final structure of the app proposed in this thesis work is shown in Figure 4.1

Project is the name I have given to my project while typing

`django –admin startproject (project_name).`

In my case it is BACKEND. It contains configuration files of the project. Project has several files as well as the App.

1. init.py As you can see in the image below, this is an empty file. By including this file, the Python interpreter is informed that the specified folder is a package and so becomes a Python project.

2. settings.py The Django project's configuration is stored here.

All the applications and middleware applications are added to settings.py, the most important file. This is where Django stores all of its settings. Changing the value of this variable will force your software to behave in a certain way.

Sqlite3 is the database of choice by default. We may change this database depending on the web application. Several pre-installed applications and middleware are also included.

3. urls.py There are a lot of endpoints in URL, which refers for universal resource location. URLs (websites, images, and so on) are given to you as a means of locating resources on the internet. Django will be told by this file to send users to the given website or image if they input this URL.

4. wsgi.py Host your application once you've finished the journey through

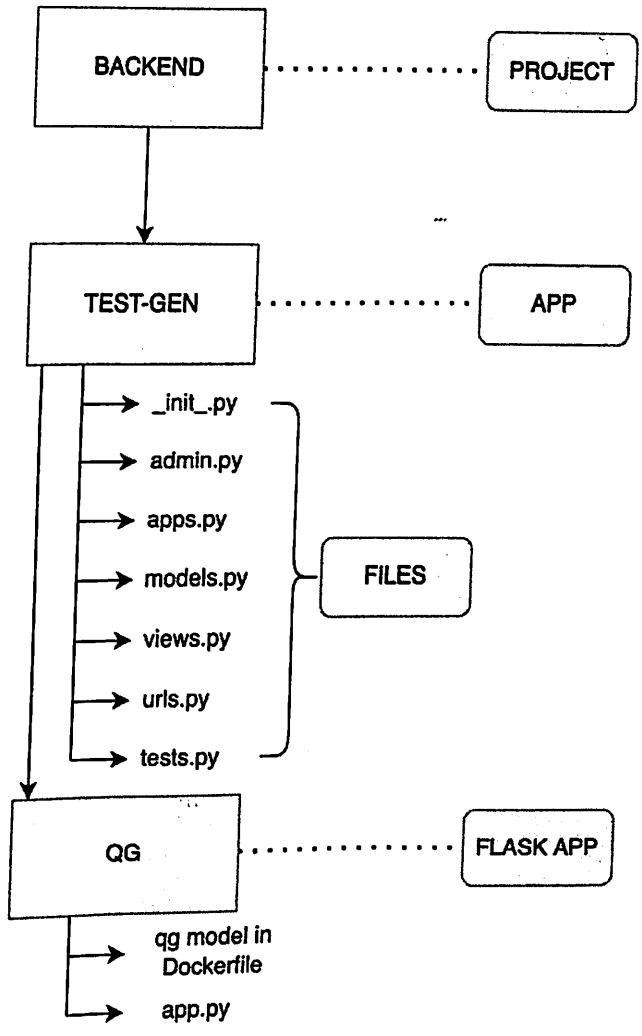

Figure 4.1: File structure of the question generation application on Django server

development to production Instead of using Django's web server, you'll be using the WSGI server. When it comes to web servers and applications, WSGI is an abbreviation which stand as Web Server Gateway Interface.

Simply import the middleware for the host you want to use, and you're done. Every server has a Django middleware that solves all integration and communication issues with Django.

5. asgi.py ASGI performs comparable tasks to WSGI, however it comes with a few more options. ASGI is an abbreviation for Asynchronous Server Gateway Interface. It has now taken the place of its predecessor, WSGI.

To make things clearer, the structure of the files is depicted in Figure 4.2

TEST-GEN is the name of the program, and it contains application files. The following are the application file descriptions with enumeration:

1. init.py This file serves the same purpose as the init.py file in the Django

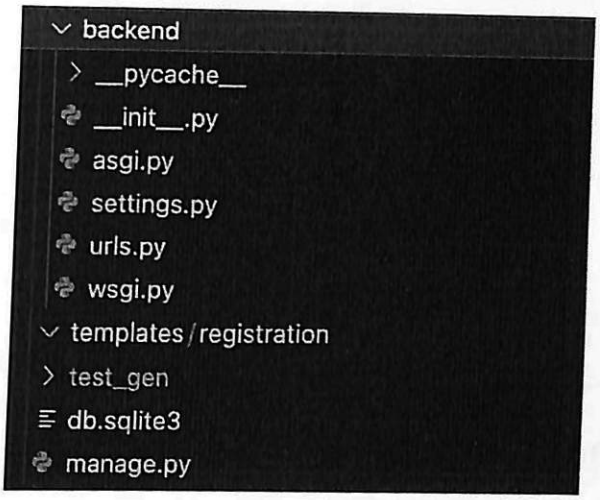

Figure 4.2: Project configuration files

project structure. It is an empty file that does not require any changes. It just indicates that the app directory is a package.

2. admin.py The Admin.py file is used to register Django models in Django administration. Django's admin panel makes use of it to show the Django model. This body part is primarily responsible for three key roles:

a. Identification of the model

d. Setting up a Superuser

c. Accessing and utilizing the web application

I defined the admin.py file in the following manner:

```
from django.contrib import admin
from .models import Question, Test
admin.site.register(Question)
admin.site.register(Test)
```

3. apps.py Apps.py is a file that allows the user to include the application setup for their app.

The apps.py file allows users to customize the characteristics of their application. However, customizing the properties is an uncommon job that a user does because the default setup is usually sufficient to operate with.

4. models.py Using the Python script Models.py, web application models may be represented in the form of classes. Most people agree that the App filesystem would be incomplete without it.

Models define the database's structure. Describes in detail the actual architecture, the linkages between data sources, and the constraints on attribute values.

Model.py is shown in following Figure 4.3

```
from django.db import models
from django.contrib.auth.models import User


class Test(models.Model):
    name = models.CharField(max_length=50)
    author = models.ForeignKey(User, on_delete=models.CASCADE)

    def __str__(self):
        return self.name


class Question(models.Model):
    question = models.CharField(max_length=200)
    answer = models.CharField(max_length=200)
    test = models.ForeignKey(Test, on_delete=models.CASCADE)

    def __str__(self):
        return self.question
```

Figure 4.3:

5. views.py Views are an important part of the Django app framework. Django web applications may be interacted with using views, which are a graphical user interface. There are classes for each of the views.

Django Rest Framework uses the Serializer concept to build a variety of views. List Views using Class-Based Filters or CustomFiltering are some examples. In this project, I have created following list of views:

`index`, `profile`, `test_view`, `all_tests`, `add_test`, `take_test`, `generate_test`, `assess_test`.

6. urls.py Urls.py functions in the same way as urls.py in the project file structure. The major goal is to connect the user's URL request to the sites it refers to. The home page, for example, will have an empty route and will display the index page. All existing pages will be documented in the 4.3 section that follows. Actually, it will not be created by Django under the app files. It is created manually.

7. tests.py Tests.py enables users to create test code for their web apps. It is used to validate the app's functionality. I did not write tests for this project because they are typically more beneficial in more sophisticated programs rather than compact ones like this one.

Flask App is named QG in this file structure. It has question generation model UniLM that was installed using Dockerfile in previous 4.1 section. Also, the flask implementation of the model, previously explained in 3.1.1 subsection.

4.3 Front-end

While Django may be used to create APIs, its primary aim is to serve as a framework for building entire websites. Django's primary ability is to render HTML, and it leverages Jinja2, hereinafter referred to as Jinja, templates to enable developing flexible, reusable webpages simple. Django comes with built-in support for Jinja templating. Jinja templating is an HTML extension that lets you incorporate data with a double bracket as well as expressions with comparable syntax

{% expression %}

In this section I will demonstrate essential parts of front-end side of the web application with Django: web page base template, review of functions, functional pages representation.

4.3.1 Jinja template

I use Jinja tags to configure the HTML page. A text file is all it takes to create a Jinja template. Every textual format may be created with Jinja (HTML, CSV, XML, LaTeX and others). No special extension is required for a Jinja template. Types like html, xml, and any other extensions will do.

When a template is shown, the template's variables and/or expressions are changed with the proper values, and tags are used to manage the template's logic. The template syntax in Django and Python has a huge impact.

Compressed version of the complete web page that displays webpages by url through using modified Jinja template is shown below. I'll go into the nitty-gritty later on:

```
<!DOCTYPE html>
<html lang="en">
<head>
  {% block title %}<title>Test Gen</title>{% endblock %}
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,
  initial-scale=1">
  {% load static %}
  <link rel="stylesheet"
  href="{% static 'css/styles.css' %}">
</head>
<body>
  <div class="container-fluid">
    <div class="row">
      <div class="col-sm-2">
      {% block sidebar %}
        <ul class="sidebar-nav">
            {% if user.is_authenticated %}
                <li><a href="{% url 'profile' %}">
                My profile</a></li>
                <li><a href="{% url 'logout' %}">
                Logout</a></li>
                {% else %}
                <li><a href="{% url 'login' %}">
                Login</a></li>
            {% endif %}
          <li><a href="{% url 'index' %}">
          Home</a></li>
          <li><a href="{% url 'all_tests' %}">
          All tests</a></li>
          <li><a href="{% url 'add_test' %}">
          New Test</a></li>
        </ul>
      {% endblock %}
```

```
        </div>
        <div class="col-sm-10 ">{% block content %}
        {% endblock %}</div>
      </div>
    </div>
</body>
</html>
```

I have made use of Bootstrap classes for the CSS part. When the size of the device or viewport increases, the fluid grid architecture that is included in Bootstrap expands up to 12 columns to provide an optimal layout that is responsive and mobile-first. It comes with powerful mixins and predefined classes for easy layout options, and it also contains predefined classes for simple layout options. As an example, I made use of the col-sm-2 and col-sm-10 classes. When utilizing Bootstrap's default grid system, you are essential to use containers because they are the most fundamental layout element. In addition, when I need a container that is full width and fills the entire viewport, I use the container-fluid property.

4.3.2 Review of functions

Up until this point, a variety of the components that went into the making of the question generator TestGen have been described. To begin, the structure is fairly complicated while while being very understandable. It was chosen to integrate the navigation bar in order to simplify the complexity and make it easier to understand. The navigation bar is the most important component of any website that covers a lot of ground. For instance, if Amazon were to eliminate their menu, the total number of users would immediately decrease. Basically because there would be complete chaos within the application if there wasn't any navigation. In addition, the order of the menu items indicates the priority of the actions. The connections that are most important are shown first. During the process of scouting potential locations for the navigation bar, it was discovered that the position in the upper left corner is the one that draws the most attention. This is an intriguing discovery. Having considered all of the findings, I have come to the conclusion that the site would benefit from having navigation in the upper left corner, with many page links.

The next step for users is to complete a number of different tasks. Text entry, viewing created test questions, taking the exam, getting results, and viewing all previously generated tests are some of the elements that are included on the list of functions that are made available to users. The user is obligated to fill up the form with the raw material and then click the submit button after they are finished. As a direct result of this, the list of questions that were developed will be displayed. At the very bottom of the list of questions, there will be a box labeled "take test." Clicking on that option will allow you to evaluate the questions that have been proposed. Following the click, the page with the quiz will open. On this page, representations of all of the questions that have blank fields can be found. It should go without saying that the blank fields are where you should put your answers. By the way, filling out the answer fields is completely optional. On a separate page, the quizzes that were generated in the past gathered and were exhibited.

A navigation bar has been added to the left side of the screen in order to improve the overall experience for users, as was mentioned earlier. It will present the users with the many activities that are available to them throughout the web page. The following is a list of navigation options: My profile, Logout, Home, All Tests, New Test.

4.3.3 Text material entering

In Figure 4.4, "Add a test" page is shown. This page features navigation on the left side of the top, a page title, descriptions, two fields, and a button. There is a field labeled "Test name" immediately following the page's title. It is anticipated that the title of the text will be typed into this particular box. The sample title "Ukraine" has been typed. This is a figurative example. This indicates that the subject matter of the test will be related to Ukraine. The following field, which is for adding the actual text material, is labeled "Source material raw text," and it is immediately following the previous field. After that, there is a button labeled "Submit." Once all of the fields have been completed, the next step is to click the button. The procedure of generating the test takes some time, which ultimately results in the generation of the test page. The results will be illustrated in the following section.

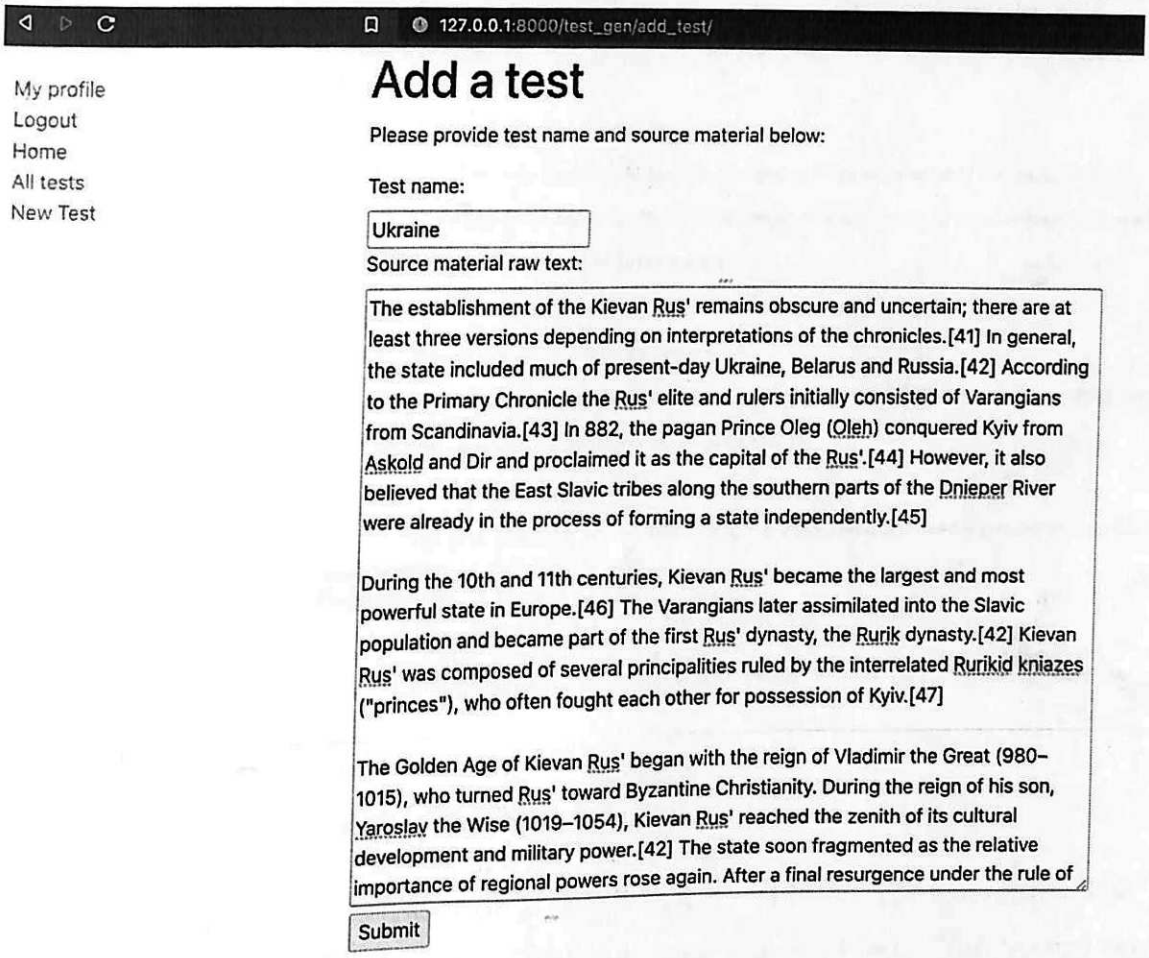

Figure 4.4: The process of adding test title and material

4.3.4 Generated questions

Figure 4.5 illustrates the outcome of the add a test page, that is the processed text material.This page, in terms of the visual aspect, is very similar to the previous add test page in that it includes the navbar and a headline that corresponds to the page called "Test page," but there are some changes as other elements on this page. The list of questions that were generated can be found on the test page. The list with the heading "My tests," along with the questions presented in bullet point format. There is a button labeled "Take the test" located at the very bottom of the questions.

In reference to the back processes, the text is placed through the stages of development, which include sentence tokenization, candidates retrieval - context and answer pair extraction, input preparation, model launch, and the generation of ten questions by shuffling the order in which they are asked. As a result, the results are entered into the database so that they can be used in further activities as outlined in the assessment section.

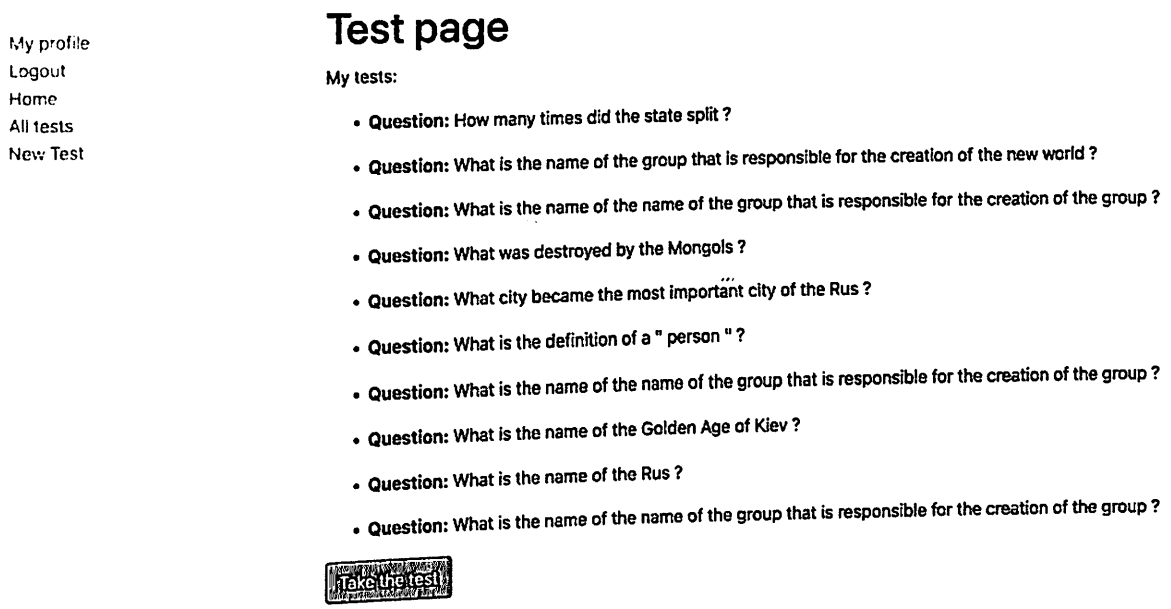

Figure 4.5: The process of adding test title and material

4.3.5 Assessment page

The assessment page is shown in Figure 4.6. This page provides the greatest amount of coverage for the data that the user has entered. The visual component follows the same organizational scheme as the preceding pages. A user-friendly website should have a framework that is consistent throughout. As a result, the website does not appear to be too overpowering. It is also important, in my view, for there to be no tension associated with the new look of the page when taking the quiz. Instead, the individual becomes accustomed to the colors and organization, which allows them to focus their attention more intently on the assessment.

In this particular illustration, the figure that serves as an example, the title of the page, determines that the name of the test will serve as the title of the test. That is, presenting the material in a large bold type immediately directs the person taking the test in the direction of the subject, which in turn directs their memory to the knowledge that is associated with the subject. As one can plainly observe, the queries are now posed in the form of genuine open questions. The user can participate in the test by reading the questions and filling out the fields that are suggested following the questions. If the user does not know the answers or does not want to enter them, they are not required to be filled out. As a result, this makes it easy to easily revise the information by acquiring the answers to the questions that were presented. The page showing the test results can be accessed once the answers have been submitted.

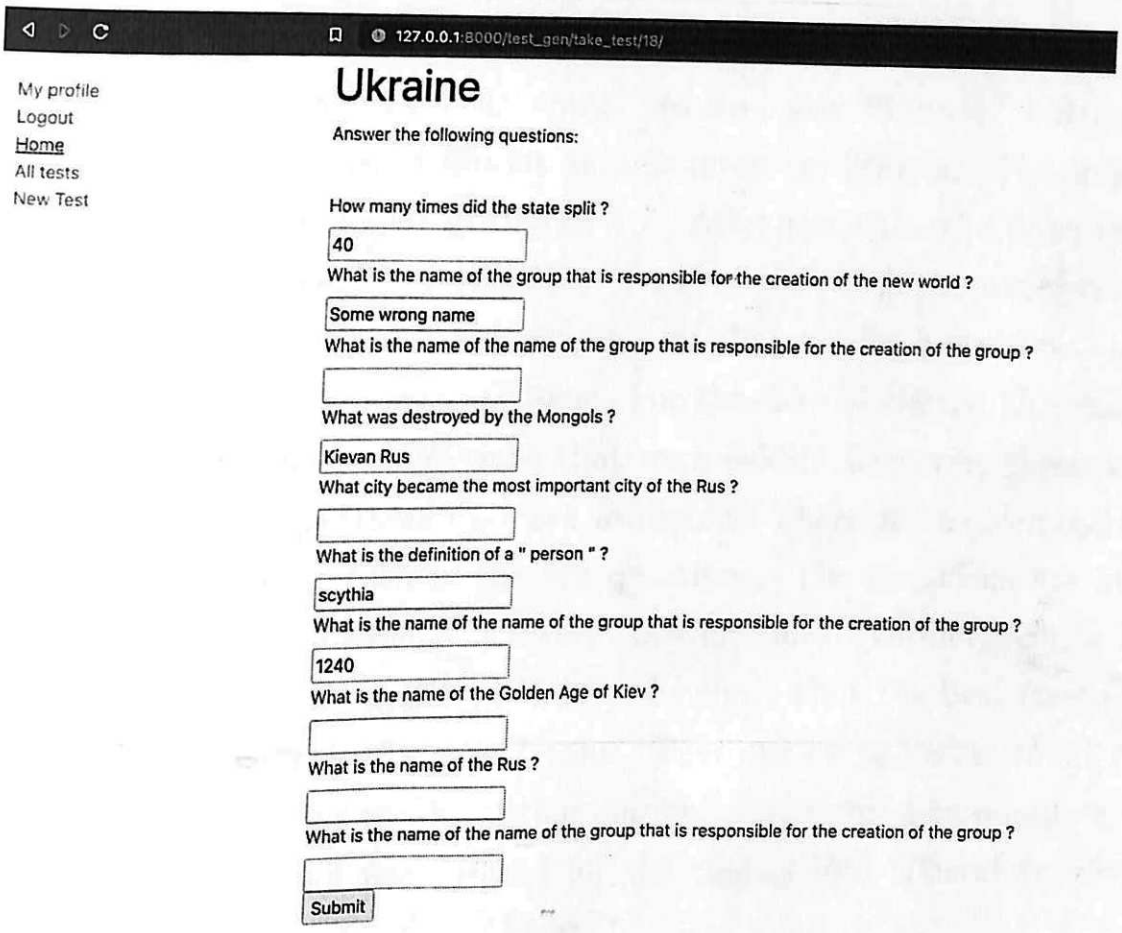

Figure 4.6: The process of adding test title and material

4.3.6 Test results page

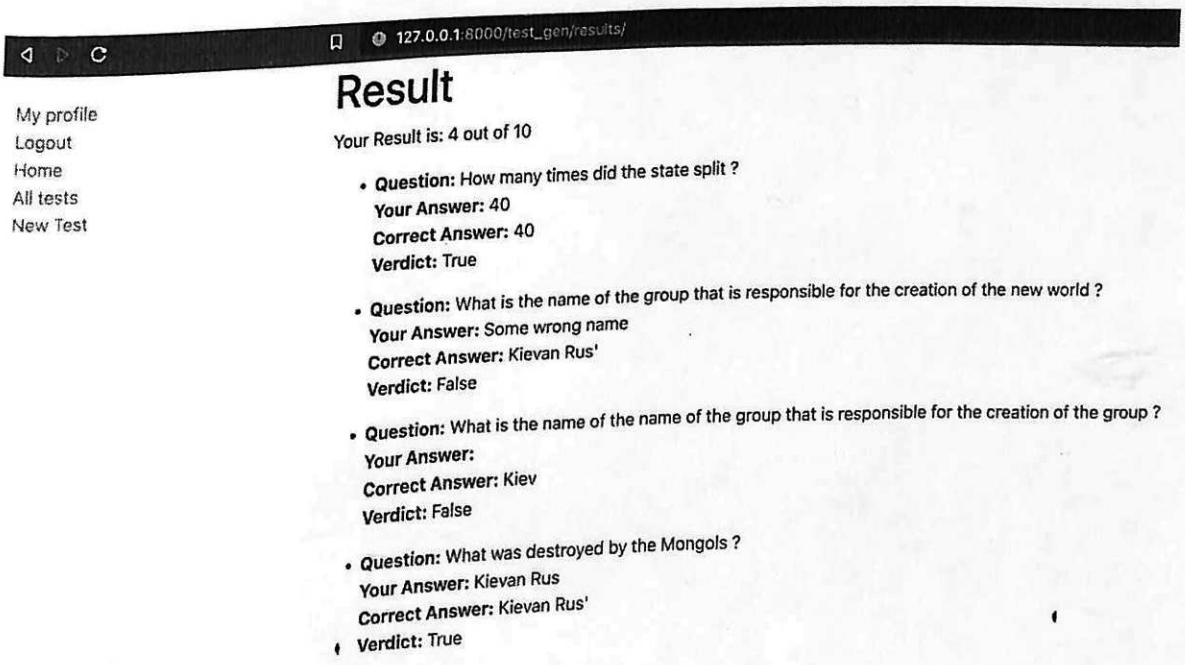

Figure 4.7: The process of adding test title and material

The "Result" page is accessible from the "Antecedent test assessment" page once the necessary tasks have been completed and the "Submit" button has been clicked. This page has a format that is more traditional. The image for the results page can be found in Figure 4.7. After the title, the description of the content is written as "Your Result is: 4 out öf 10." This particular case example is used to illustrate how the description of the content is written. This brings up the last and most important issue. For the sake of clarity, this diagram only includes four of the ten questions that were asked. However, there are also six of the remaining questions that are available. There is an examination of the answers provided for each of the ten questions. The questions are followed by the answers "Your Answer," "Correct Answer," and "Verdict." Since I consider each of the fields to be self-explanatory, I believe that the best course of action would be to explain the "Verdict" value. The user's response is compared to the right response, and the results of that comparison are used to populate this field. The Levenshtein score was utilized for the calculation. Therefore, determining whether the value is "True" or "False."

The real question, the user's response to that question, the actually correct answer, and the verdict are all displayed on the result page, therefore it is clear that this page provides a good deal of information. The Levenshtein distance is used to describe the verdict after comparing the response given by the user with the correct response.

5. Results and future work

In this chapter, the work that was done for the thesis is finished so that findings may be made, and any kind of drawbacks are considered in order to develop an outline for future work. When the results are finalized, the work that has been done becomes clearer as a consequence of the analysis and weighing of the pros and disadvantages of individual judgments made throughout the process of doing research. In subsequent works, it will be possible to both address any deficiencies that have been identified and make users aware of additional potential afforded by the system.

5.1 Results

This part will present and demonstrate the program's final results after it has been completed. The Docker container that hosts the Django web service contains a micro-service implementation of the model interpreter written in Flask. In a previous step, the structure of the beneath processes of the Django application was described, and the location where the Flask service is located was demonstrated.

Firstly, the model is the Unified Language model which was pre-trained on SQuAD dataset. Because of the high level of complexity exhibited by the model, cutting-edge technologies such as Flask and Docker have been implemented. SQuAD is widely acknowledged to be the database that performs exceptionally well. To begin, the data collection went really smoothly. Additionally, it is a well-established truth that the greater the amount of data, the higher the level of precision achieved. As a result, the fact that the selected dataset contains such a substantial amount of data makes it the most appropriate. During the course of the running process, the pre-trained model was used to decode data, make inferences, and be evaluated.

Second, since the pre-trained model requires more than 1 gigabit of RAM, it was decided to run it on docker. This decision was made because it was essential for the performance of the model. Container logic is the foundation of Docker. In relation to the memory, the SQlight3 was utilized as the database management system for this project. SQlight3 is written in the C programming language, also known as the low level language, which makes it easier for computers to read. It allows for very rapid execution of the processes that are run by the database. Django was able to function properly with both the settings and the database. This file of the configuration deals with the programs that are used by the middleware.

As a final step, although certainly not the least important one, the front-end side was also developed on Django by utilizing the components structure. The HTML was configured with the help of Jinja2, which I used. To put it another way, Jinja made it possible to give the Django project a livelier appearance. Components are small bits that make up a larger whole. It is very helpful to split a complicated undertaking into these smaller sections. Therefore, numerous pages have been realized as the home page, and each of these pages renders the fundamental functionality, which is text uploading. In addition to that, there is routing in front of a great number of pages. The following pages are produced: an initial index page that represents the text that is uploaded; a page for generated questions; a page for test taking; a page for test results after evaluation; and also, the tests that are produced are gathered and memorized. The "my tests" page is where you can find all of the tests you have memorized.

In general, the system is quite intuitive and easy to comprehend for users. The finished product is a platform that runs on the web. Users have access to the following features and functions:

- Put up readings that will be used to make questions
- Automatic evaluation and results for whichever test that is made
- Putting the questions in context
- Ability to navigate through web page
- Revise educational material, or to use in any alternative purposes
- View tests created by the user

5.2 Future work

In this section, the limitations and drawbacks of the analyzed thesis are stated, and the solutions that match to those constraints and drawbacks are presented and outlined.

Because of the limited amount of time available, a wide variety of adjustments, tests, and experiments have been delayed (i.e. experiments using actual data are typically very expensive, taking days to complete a single run.). In the future, I plan to examine certain systems in further detail, offer fresh ideas for people to test out in different ways, or simply investigate various topics out of simple interest. Working in groups, or teams as they are more popularly known, is the most efficient way to construct code that is exhaustively thorough and covers every facet of an application. This methodology is standard practice in the field of programming. Because this was an individual project, working together on it was not the solution. However, keeping in mind that working in teams is an option for the future, the question generator that was described in this article might be escalated in teams.

The usage of the UniLM model launching and the preparation of the environment have taken up the most of the attention in this thesis. Specifics such input preparation and test evaluation have been left out because they fall outside the purview of the thesis. To argue that input preparation was neglected in favor of the primary focus would be an exaggeration. In any case, it is important to point out the negatives associated with this element.

The following solutions to the aforementioned issues could be put to the test:

- Although NER was used to select the responses, random selection was applied to accomplish the task of preparing the input. Using models that extract the most essential sentences from different text sections is one approach that can be used to solve this challenge.

- The Levenshtein distance eliminates the difficulties brought on by typos and articles, although it does bring up certain difficulties with numerical values. Despite the fact that "1 million" is just half as large as "2 million," Levnshtein will award it a high similarity score. Also, to us, the numbers "10" and "ten" sound the same, but their similarity score is rather low.

These errors must to be taken into consideration when doing a thorough review of the evaluation section.

In addition, considering that the work for the thesis is predominately of a scientific nature, I have focused more on the efficiency and effectiveness of the system. There is no shadow of a doubt that design has a big influence both on the experience of the user and on the engagement of the user. Because of this, in my opinion, the additional focus that is required to construct a design that is convenient is necessary. When we talk about convenient design, we are referring to the process of browsing among applications, which paints the picture of a positive user experience.

The thesis, as a whole, has achieved all of the goals that it set out to accomplish.

6. Conclusion

In foundational subjects like history and biology, amongst others, students are required to compose a great number of essential term writing tasks. The purpose of these particular kinds of assignments is to force students to go through the content, and their weight in the overall grade is typically quite low—only about 5 percent. However, because there are hundreds of students each year, the instructors have to spend a significant amount of time checking the students' work. This problem is remedied by the proposed technique, which entails the automation of the procedure that came before it and the transformation of key-term obligations into examinations that serve the same purpose: to examine the level of comprehension the students have achieved by requiring them to review the content. In this work, automated testing generation based on reading material (TestGen) was proposed. TestGen is a program that uses cutting-edge Natural Language Processing (NLP) technology to create questions based on the text you give it and an automated system for judging how well those questions are answered.

The baselines for this research were developed in Chapter 2 which also provided a comprehensive examination of the background information regarding the associated works, dataset, and language models. The overall comprehension of the question generation subject has been significantly aided by the studies that are related to it. In particular, the research conducted by Vasvani et al. [6], Dong et al. [4], and Rajpurkar et al. [37].

The works of Vasvani et al. [6] presented the Transformers as the basis of the UniLM model. According to Vasvani et al. [6], the Transformer is the first sequence transduction model that is wholly based on attention, with multi-headed self-attention replacing the recurrent layers that are frequently employed in encoder-decoder topologies. The Transformer was developed by the group of researchers known as Vasvani. When it comes to jobs involving translation, the

Transformer has the potential to be trained significantly more quickly than models containing convolutional layers.

The work of the UniLM developers [4] served as the foundation for the system engine. A pre-training model that is optimized for a variety of LM goals and contains parameters that are shared was proposed by Dong et al. [4]. UNILM performs much better than the best models across all five NLG datasets.

Rajpurkar et al. [37] was responsible for the creation of the dataset that was utilized. SQuAD stands for the Stanford Question Answering Dataset, which was given by Rajpurkar and colleagues [37]. This large dataset is comprised of questions and responses regarding Wikipedia articles that were provided by members of the general public. They found that their logistic regression model did not perform as well as the human F1, which had an accuracy of 86.8 percent. This implies that there is a significant amount of room for improvement.

The research approach that was used for this thesis was covered in Chapter 3. This chapter included the model configuration, input preparation, flask service that was used to support the language model, and the test assessment methodologies. The UniLM was selected as a language model for a number of different reasons. To begin, in terms of its function as a Transformer, it is noticeably more effective and has demonstrated the highest rates of accuracy [4]. Second, it has already had some preliminary training with the SQuAD dataset. Lastly, the utilization of it is quite easy and uncomplicated. The Levenstein distance will serve as the evaluation method for the responses that are given. When contrasted with exact matches, which may contain errors such as typos, prepositions, or articles, the strategy described above was recommended.

The implementation of the thesis work is covered in Chapter 4. Docker was utilized in the process of establishing the environment in this chapter. It is fairly lightweight, and the orchestration does not need to be overly complicated. It was successfully executed on Linux, namely the Ubuntu 20.04 edition. Docker was utilized as a tool throughout the uploading process of the UniLM model. The web-server side was covered in greater detail later on in this chapter. Because it is structured and built on Python, I decided to use Django as my framework of choice. In most circumstances, including this one, Django is the option that works best. Django was used as the central system to join the microservices that were provided by Docker and Flask. So, the system is built all the way through,

including the back-end, the front-end, and the database.

The results and the future work are presented in Chapter 5. As was already said, the project is done in its entirety and has a number of different functions.

The fact that the thesis work is presented in the form of a web application makes it feasible to "feel" the work. Users have the ability to input the mass of text material that has to be processed, obtain questions that are generated and save them in csv format, then take an evaluation and obtain the findings. As a direct consequence of this, it is safe to claim that it is incredibly user-friendly and simple to use.

I have no doubt in my mind that the educational system personnel will benefit from the thesis work that was completed, which consisted of the construction of a system for generating test questions and answers for a given text. A review of the materials that were read in the book is something that will be beneficial not only for the instructors but also for the students. In addition, there are applications, such as the generation of commonly asked questions and a great deal more.

A. Appendix A

```
import os, subprocess
from flask import Flask
from flask import request
from flask import jsonify
app = Flask(__name__)

def generate():
    myenv = os.environ.copy()
    myenv['PYTORCH_PRETRAINED_BERT_CACHE'] = '/bert-cased-pretrained-cache'
    subprocess.call(['python',
'/unilm/unilm-v1/src/biunilm/decode_seq2seq.py',
                     '--bert_model', 'bert-large-cased',
                     '--new_segment_ids', '--mode', 's2s',
                     '--input_file', '/data/answers_flask.txt',
                     '--model_recover_path', '/data/qg_model.bin',
                     '--max_seq_length', '512', '--max_tgt_length', '48',
                     '--batch_size', '16', '--beam_size', '1',
'--length_penalty', '0',
                     '--output_file', '/data/questions_flask.txt'], env=myenv)


@app.route('/', methods=['POST'])
def hello_world():
    data = request.get_json(force=True)
    contexts = data['context']
    answers = data['answers']
    with open('/data/answers_flask.txt', 'w') as f:
        for c, a in zip(contexts, answers):
            f.write(c + ' [SEP] ' + a + '\n')


    generate()
    questions = None
    with open('/data/questions_flask.txt', 'r') as f:
        questions = f.readlines()

    return jsonify({'questions': questions})
```

Figure A.1: Code representation of Flask usage

B. Appendix B

```
. .bashrc
apt-get update
apt-get install -y vim wget ssh

PWD_DIR=$(pwd)
cd $(mktemp -d)
git clone -q https://github.com/NVIDIA/apex.git
cd apex
git reset --hard 1603407bf49c7fc3da74fceb6a6c7b47fece2ef8
python setup.py install --user --cuda_ext --cpp_ext
cd $PWD_DIR

pip install --user tensorboardX six
numpy tqdm path.py pandas scikit-learn lmdb
pyarrow py-lz4framed methodtools py-rouge pyrouge nltk
python -c "import nltk; nltk.download('punkt')"
pip install -e
git://github.com/Maluuba/nlg-eval.git#egg=nlg-eval
```

References

[1] Vasile Rus et al. "A detailed account of the first question generation shared task evaluation challenge". In: *Dialogue and Discourse* 3.2 (2012), pp. 177–204.

[2] James Allen. *Natural language understanding*. The Benjamin/Cummings Publishing Company, 1995.

[3] Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing*. Vol. 2. CRC Press, 2010.

[4] Li Dong et al. "Unified language model pre-training for natural language understanding and generation". In: *Advances in Neural Information Processing Systems* (2019). URL: https://arxiv.org/pdf/1905.03197.pdf.

[5] Yao Zhao et al. "Paragraph-level neural question generation with maxout pointer and gated self-attention networks". In: *Conference on Empirical Methods in Natural Language Processing* (2018), pp. 3901–3910.

[6] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* (2017), pp. 5998–6008.

[7] John H Wolfe. "Automatic question generation from text-an aid to independent study". In: *ACM SIGCSE Bulletin* 8.1 (1976), pp. 104–112.

[8] Lynette Hirschman et al. "Deep read: A reading comprehension system". In: *ACL* (1999), pp. 325–332.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "A machine learning approach to answering questions for reading comprehension tests". In: *Advances in neural information processing systems* (2014), pp. 3104–3112.

[10] J. Pino and M. Eskenazi. "Measuring hint level in open close questions". In: *22nd International Florida Artificial Intelligence Research Society Conference(FLAIRS)* (2009), pp. 460–465.

[11] B. Das and M. Majumder. "Factual open cloze question generation for assessment of learner's knowledge". In: *International Journal of Educational Technology in Higher Education* (2017), pp. 1–12.

[12] C. Chen, H. Liou, and J.S. Chang. "A preliminary inquiry into using corpus word frequency data in the automatic generation of English language cloze tests". In: *Calico Journal* (1997), pp. 15–33.

[13] C. Chen, H. Liou, and J.S. Chang. "Fast–an automatic generation system for grammar tests". In: *COLING/ACL on Interactive Presentation Sessions* (2006), pp. 1–4.

[14] A. Hoshino and H. Nakagawa. "Assisting cloze test making with a web application". In: *Society for Information Technology Teacher Education International Conference* (2007), pp. 2807–2814.

[15] M. Agarwal and P. Mannem. "Automatic gap-fill question generation from text books". In: *6th Workshop on Innovative Use of NLP for Building Educational Applications* (2011), pp. 56–64.

[16] R. Mitkov, H. LE An, and N. Karamanis. "A computer-aided environment for generating multiple-choice test items". In: *Natural Language Engineering* (2006), pp. 177–194.

[17] A.S. Bhatia, M. Kirti, and S.K. Saha. "Automatic generation of multiple choice questions using wikipedia". In: *Pattern Recognition and Machine Intelligence* (2013), pp. 733–738.

[18] A.S. Bhatia, M. Kirti, and S.K. Saha. "A system for generating multiple choice questions: With a novel approach for sentence selection". In: *2nd Workshop on Natural Language Processing Techniques for Educational Applications* (2015), pp. 64–72.

[19] G. Deena et al. "Developing the assessment questions automatically to determine the cognitive level of the E-learner using NLP techniques". In: *International Journal of Service Science, Management, Engineering, and Technology (IJSSMET)* (2020), pp. 95–110.

[20] S. Burrows, I. Gurevych, and B. Stein. "The eras and trends of automatic short answer grading". In: *International Journal of Artificial Intelligence in Education* (2015), pp. 60–117.

[21] C. Leacock and M. Chodorow. "C-rater: Automated scoring of short-answer questions". In: *Computers and the Hümanities* (2003), pp. 389–405.

[22] L. Bin et al. "Automated essay scoring using the KNN algorithm". In: *International Conference on Computer Science and Software Engineering* 1 (2008), pp. 735–738.

[23] T. Kakkonen et al. "Comparison of dimension reduction methods for automated essay grading". In: *Journal of Educational Technology Society* 11 (2008), pp. 275–288.

[24] F. Noorbehbahani and A.A. Kardan. "The automatic assessment of free text answers using a modified BLEU algorithm". In: *Computers Education* (2011), pp. 337–345.

[25] A. Dhokrat, H. Gite, and C.N. Mahender. "Assessment of answers: Online subjective examination". In: *Workshop on Question Answering for Complex Domains* (2012), pp. 47–56.

[26] M.M. Islam and A.L. Hoque. "Automated essay scoring using generalized latent semantic analysis". In: *13th International Conference on Computer and Information Technology (ICCIT)* (2010), pp. 358–363.

[27] L. Ramachandran, J. Cheng, and P. Foltz. "Identifying patterns for short answer scoring using graph-based lexico-semantic text matching". In: *Tenth Workshop on Innovative Use of NLP for Building Educational Applications* (2015), pp. 97–106.

[28] K. Sakaguchi, M. Heilman, and N. Madnani. "Effective feature integration for automated short answer scoring". In: *2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2015), pp. 1049–1054.

[29] A. Abdi et al. "Deep learning-based sentiment classification of evaluative text based on multi-feature fusion". In: *Information Processing Management* (2019), pp. 1245–1259.

[30] Y. Zhu et al. "Visual7w: Grounded question answering in images". In: *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4995–5004.

[31] L. Yu et al. "Visual madlibs: Fill in the blank description generation and question answering". In: *IEEE International Conference on Computer Vision* (2015), pp. 2461–2469.

[32] M. Ren, R. Kiros, and R. Zemel. "xploring models and data for image question answering". In: *Neural Information Processing Systems* (2015), pp. 2953–2961.

[33] S. Zhang et al. "Automatic generation of grounded visual questions". In: *26th International Joint Conference on Artificial Intelligence* (2017), pp. 4235–4243.

[34] N. Mostafazadeh et al. "Generating Natural Questions About an Image". In: *54th Annual Meeting of the Association for Computational Linguistics* (2016), pp. 1802–1813.

[35] J. Johnson, A. Karpathy, and L. Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning". In: *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 4565–4574.

[36] U. Jain, Z. Zhang, and A.G. Schwing. "Creativity: Generating diverse questions using variational autoencoders". In: *IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 6485–6494.

[37] Pranav Rajpurkar et al. "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *The 2016 Conference on Empirical Methods in Natural Language Processing* (2016), pp. 2383–2392.

[38] Sam Shead. *Why everyone is talking about the A.I. text generator released by an Elon Musk-backed lab.* URL: https://www.cnbc.com/2020/07/23/openai-gpt3-explainer.html. (accessed: 23.07.2020).

[39] Frederik Bussler. *Will The Latest AI Kill Coding?* URL: https://towardsdatacom/will-gpt-3-kill-coding-630e4518c04d. (accessed: 21.07.2020).

[40] David Chalmers. *GPT-3 and General Intelligence.* URL: https://dailynous.com/2020/07/30/philosophers-gpt-3/#chalmers. (accessed: 04.08.2020).

[41] Steven Johnson. *A.I. Is Mastering Language. Should We Trust What It Says?* URL: https://www.nytimes.com/2022/04/15/magazine/ai-language.html. (accessed: 15.04.2022).

[42] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (2019). DOI: 10.48550/arXiv.1810.04805. URL: https://doi.org/10.48550/arXiv.1810.04805.

[43] Gonzalo Navarro. "A guided tour to approximate string matching". In: *ACM Computing Surveys* (2001). DOI: 10.1145/375360.375365. URL: https://repositorio.uchile.cl/bitstream/handle/2250/126168/Navarro_Gonzalo_Guided_tour.pdf.