



Dissertation on

“Question Answering System”

Submitted in partial fulfilment of the requirements for the award of degree of

**Bachelor of Technology
in
Computer Science & Engineering**

Submitted by:

Pragya Agrawal	01FB16ECS258
Priyanka A	01FB16ECS278
Ranjitha Nayak	01FB16ECS298

Under the guidance of

Internal Guide

Dr. Mamatha H.R.

Professor,
PES University

January – May 2020

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India



PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

Question Answering System

is a bonafide work carried out by

Pragya Agrawal

01FB16ECS258

Priyanka A

01FB16ECS278

Ranjitha Nayak

01FB16ECS298

In partial fulfilment for the completion of eighth-semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2020 – May. 2020. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th-semester academic requirements in respect of project work.

Signature
Dr. Mamatha H.R.
Professor

Signature
Dr. Shylaja S S
Chairperson

Signature
Dr. B K Keshavan
Dean of Faculty

External Viva

Name of the Examiner

Signature with Date

1. _____

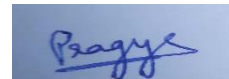
2. _____

DECLARATION

We hereby declare that the project entitled **Question Answering System** has been carried out by us under the guidance of Prof. Dr. Mamatha H.R. and submitted in partial fulfilment of the course requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2020. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

01FB16ECS258

Pragya Agrawal



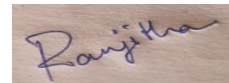
01FB16ECS278

Priyanka A



01FB16ECS298

Ranjitha Nayak



ACKNOWLEDGEMENT

First and foremost, we would like to express our utmost gratitude to our guide, Dr. Mamatha H.R. for giving us an opportunity to work under her and for giving us invaluable guidance throughout the project. She provided us with many insightful ideas and encouragement which played a vital role in the shaping of the project.

We thank our project coordinators Prof. Preet Kanwal and Prof. Sangeeta V for managing and coordinating with us during the course of the project.

We are deeply grateful to Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the opportunities that the department has provided us.

We would like to extend our heartfelt gratitude to Dr. B.K. Keshavan, Dean of Faculty, PES University, for all his aid.

The completion of this project would not have been accomplished without the support of Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro-Chancellor, PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University. We thank you all.

We are extremely grateful to our parents and family for all their love and sacrifices for us. We cannot thank them enough.

ABSTRACT

Machine understanding perception and question noting is a fundamental errand in common language handling. As of late, Pre-prepared Contextual Embeddings (PCE) model, Bidirectional Encoder Representations from Transformers (BERT) has pulled in loads of consideration because of its incredible execution in a wide scope of NLP undertakings. In this venture, the BERT model is fine-tuned with extra undertaking question-answer specific layers to improve its exhibition on Stanford Question Answering Dataset (SQuAD 2.0). A closed domain question answering system is developed, which is 'computer security' domain specific for the use of interactive learning which makes learning very exciting. The system has also been extended by adding document retrieval and cache for faster access. With these developments in the current BERT model, the best accuracy achieved is 90%.

TABLE OF CONTENTS

Chapter No.	Title	Page No.
1. INTRODUCTION		1
2. PROBLEM DEFINITION		4
3. LITERATURE SURVEY		5
3.1	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding	5
3.1.1	Author and publication details	5
3.1.2	Summary	5
3.1.3	Research Gaps	6
3.2	BERT for Question Answering on SQuAD 2.0	6
3.2.1	Author and publication details	6
3.2.2	Summary	6
3.2.3	Research Gaps	7
3.3	Real Life Application of a Question Answering System Using BERT Language Model	7
3.3.1	Author and publication details	7
3.3.2	Summary	7
3.3.3	Research gaps	8
3.4	Simple Applications of BERT for Ad Hoc Document Retrieval	8
3.4.1	Author and publication details	8
3.4.2	Summary	8
3.4.3	Research gaps	9
3.5	Efficient and Robust Question Answering from Minimal Context over Documents	9
3.5.1	Author and publication details	9
3.5.2	Summary	9
3.5.3	Research gaps	10
4. PROJECT REQUIREMENTS SPECIFICATIONS		11
4.1	Modules	11
4.1.1	Module 1: User Interface	11
4.1.2	Module 2: Cache	11
4.1.3	Module 3: Information Retrieval	12
4.1.4	Module 4: Question Answer System	12
4.2	Constraints	12
4.2.1	Design Constraints	12
4.2.2	System Constraints	13
4.3	Product Perspective	13
4.3.1	User Characteristics	13

4.3.2 General Constraints, Assumptions and Dependencies	13
4.3.2.1 Constraints	13
4.3.2.2 Assumptions	13
4.3.2.3 Software Dependencies:	14
4.3.2.4 Hardware Dependencies:	14
4.3.3 Risks	14
5. SYSTEM REQUIREMENTS SPECIFICATION	16
5.1 Functional Requirements	16
5.1.1 Picking the right document	16
5.1.2 Finding the answer in the selected context	16
5.1.3 Picking the document and answer from the cached data	16
5.2 Non-Functional Requirements	16
5.2.1 Performance	16
5.2.2 Availability	17
5.2.3 Capability	17
5.2.4 Usability	17
5.2.5 Documentation	17
5.3 Hardware Requirements	17
5.4 Software Requirements	18
6. SYSTEM DESIGN	19
7. DETAILED DESIGN	22
7.1 Component Diagram	22
7.1.1 Components:	23
7.1.2 Interfaces:	24
7.2 User Interfaces	24
7.2.1 Google Colab	24
8. IMPLEMENTATION AND PSEUDOCODE	26
8.1 Collecting and preparing data	26
8.1.1 Context	26
8.1.2 Content	26
8.1.3 Domain Specific Data Preparation	27
8.1.3.1 Implementation	27
8.1.3.2 Pseudocode	27
8.2 Training the model	28
8.2.1 Pre-Training	28
8.2.2 Fine-Tuning	28
8.3 Picking the Relevant Document	31
8.3.1 Handling the length constraint	31

8.3.1.1 Implementation	31
8.3.1.2 Pseudocode	31
8.3.2 Technique to find the most relevant document	31
8.3.2.1 Implementation	31
8.3.2.2 Pseudocode	32
8.4 Finding the right answer	32
8.4.1 Implementation	32
8.4.2 Pseudocode	33
9. TESTING	34
9.1 Strategies	34
9.2 Test Cases	35
10. RESULTS AND DISCUSSION	36
10.1 Model Evaluation	36
10.2 Document Retrieval for query	36
10.3 Answer Prediction for given query from the fetched document	36
10.4 Increase in response time by caching	37
11. SNAPSHOTS	38
11.1 Homepage of the application	38
11.2 Query Submission	38
11.3 Document Retrieved	39
11.4 Display Answer	39
11.5 Previously Asked questions	40
12. CONCLUSIONS	41
13. FURTHER ENHANCEMENTS	42
REFERENCES/BIBLIOGRAPHY	43
DEFINITIONS, ACRONYMS AND ABBREVIATIONS	46

LIST OF TABLES

Table No.	Title	Page No.
8.1	Comparison of the two models	30

LIST OF FIGURES

Figure No.	Title	Page No.
1.1	Types of QA systems	3
4.1	User Interface for the system	11
6.1	System Design with detailed modules	20
7.1	Component Diagram	22
7.2	Google Colab	25
8.1	Pseudocode for domain specific data	27
8.2	Training the model	28
8.3	Training of the BERT-Large, Uncased model.	29
8.4	Training of the BERT-Base, Uncased model.	29
8.5	Model Checkpoints created in GCS bucket after training	30
8.6	Pseudocode Handling lengthier context	31
8.7	Pseudocode Document retrieval	32
8.8	Pseudocode Answer Prediction	33
9.1	Testing Strategies	34

11.1	Homepage	38
11.2	Query Submission	38
11.3	Document Retrieved	39
11.4	Answer Returned	39
11.5	Cached Results	40

CHAPTER-1

INTRODUCTION

As technology increases there is a lot of human computer interaction and exponential rise in data in day to day life. The field of natural language processing made it possible with its methodologies to use the data to generate a variety of models that makes this interaction easy. One such model is Question answering Model which requires to understand the query given and respond with the concise answer.

QA frameworks expect to recover highlight point answers instead of flooding with reports or in any event, coordinating sections as the majority of the data recovery frameworks do. For E.g. "what does the ACL stand for?". The specific answer expected by the client for this inquiry is (Access Control List) The major testing issues in the Question noting framework is to give exact answers from colossal information accessible on the web. It likewise plans to perceive the cross phonetic inquiries which permit the clients to pose to the inquiries and acquire the appropriate responses in their local language. The preparing of time based data to answer fleeting inquiries despite everything stays as a test. In this paper, various kinds of QA frameworks are referred to and taken note of.

In any case, with ongoing advancements in profound learning, neural system models have demonstrated guarantee for QA. Despite the fact that these frameworks for the most part include a smaller learning pipeline, they require a huge measure of preparing. GRU and LSTM units permit repetitive neural systems (RNNs) to deal with the longer messages required for QA. Such systems give the current best in class execution for profound learning-based QA.

Question answering system expects to give answers for questions communicated in natural language consequently. Question Answering frameworks intend to recover anticipated answers to the questions as opposed to a positioned rundown of records as most Information Retrieval Systems do. Question answering frameworks show a vital progression in data retrieval innovations, particularly in its capacity to get to information assets in a characteristic manner by questioning and recovering right answers in succinct words.

The main aim of this system is to answer the questions asked in natural language. The most common question answering system is selecting an answer from the context. Applications of these systems can be vastly seen in customer service bots, chatbots, search engines, domain specific bots for learning etc.

Question Answering Systems offer a computerized approach to procuring answers for inquiries communicated in characteristic language. A ton of QA surveys have classified Question Answering frameworks dependent on various criteria such as inquiries asked by clients, highlights of information bases utilized, nature of generated answers, question noting approaches and methods. To fully understand QA frameworks, how it has developed into its current QA needs, and the need to scale up to meet future desires, a more extensive study of QA systems becomes basic.

There are different types of QA systems based on the question type like fact, list, definition, How, Why, hypothetical, semantically constrained, and cross-lingual questions. In general there are two kinds of question answering systems: closed domain and open domain QA. Closed domain QA is very context specific. It is simpler to build a closed domain QA when compared to open domain QA because of its restricted context. QA systems are better than search engines because they eliminate a manual effort of humans to look for an answer in the pool of documents returned by the search engines. These systems give the feel of talking to another human. The focus here is on restricted domain QA.



Figure 1.1 Types of QA System

Source: Adapted from [11]

At the point when the client offers a question to a framework sitting on a gigantic database of unstructured information, the principal request of business is to diminish that heap to maybe a bunch of records where the answer is probably going to be found. This implies utilizing quick yet non-precise choice techniques regularly for document retrieval. Questions are tokenized and sent to a document retrieval motor.

Some of the QA systems are ‘ask.com’ which is based on natural language processing, ‘Askpoddle.com’ which depends on humans to create answers etc.

CHAPTER-2

PROBLEM DEFINITION

Build a closed domain question answering system using a pre-trained BERT model and IR methodologies which is capable of answering the asked questions. The model requires the context to be provided along with the question. The challenge is to send a response to the user as quickly as possible with the design method followed.

The context for the built QA model is ‘computer security’. Any given question related to the context will be answered by the model if the answer is a part of the context. The various phases included are: Query Pre-processing, Document Retrieval and Answer Prediction. The model learns to find the answer in the given context and IR phase will decide the context to be passed to the model when given a query. As long as the correct context is passed, the model returns the correct answer.

Final deliverable of the project is to design a user interface where the user can submit the question. The question runs through various phases and finally returns the answer and the context from which the answer was picked. Users can also rate their satisfaction on the returned answer.

CHAPTER-3

LITERATURE SURVEY

The vast majority of the research work is based on BERT model and Information retrieval procedures. This stage gave us a great deal of thoughts on the most proficient methods, to approach the venture with as far a structure and usage as possible. The QA model is built utilizing a pre-prepared BERT model and TF-IDF technique.

3.1 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

3.1.1 Author and publication details

Authors: Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova

Publisher: Association for Computational Linguistics

Date: 24 May 2019

Place: Minneapolis, Minnesota

Venue: NAACL

Link: <https://arxiv.org/pdf/1810.04805.pdf>

3.1.2 Summary

- The article discusses the BERT (Bidirectional Encoder Representations from Transformers) model. BERT is intended to pre-train profound bidirectional representation from the unlabelled data by together checking on both left and right settings in all layers. It fills in as a base for the different state of art models with simply including one output layer and adjusting it to make it task-explicit.
- For fine tuning, the BERT model is first introduced with the pre-prepared parameters, and the entirety of the parameters are tweaked utilizing named information from the

downstream tasks. Fine tuning in less complex words is prepared with task-explicit, marked information.

- The downstream undertakings discussed in this paper are:
 - Conceal LM model predicts the missing word in the sentence.
 - Next sentence expectation.
 - Question answer model utilizing SQUAD dataset.

3.1.3 Research Gaps

- Sentence Reconstruction in the BERT includes haphazardly masking tokens in a sentence and remaking the first sentence from the conceal one. Masked tokens are viewed as free of each other which is anything but a legitimate presumption.
- Length of the setting of context is restricted. The most extreme length of context can be picked in BERT however there is a restriction by the accessible memory on GPU.

3.2 BERT for Question Answering on SQuAD 2.0

3.2.1 Author and publication details

Author: Yuwen Zhang, Zhaozhuo Xu

Year of Publication: 2019

Link: <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15848021.pdf>

3.2.2 Summary

- The BERT model is picked and an attempt to fine-tune it with extra task-specific layers is made to improve its performance on Stanford Question Answering Dataset (SQuAD 2.0).
- A few architectures are planned and their presentations are contrasted with the BERT based model in incredible subtleties. Up until now, best-proposed single model assembled an LSTM Encoder, an LSTM decoder and an interstate system on the BERT base uncased

model and accomplished an F1 score of 77.96 on the dev set. By applying a gathering system with chosen models, the last form model as of now positions twelfth on the Stanford CS224N SQuAD 2.0 test leader board with a F1 score 77.827 (name: Pisces_BERT).

3.2.3 Research Gaps

- There is still a gap in has-answer and no-answer accuracy which should be tried to compensate with ensemble on no-answer predictions.

3.3 Real Life Application of a Question Answering System Using BERT Language Model

3.3.1 Author and publication details

Authors: Francesca Alloatti, Luigi Di Caro, Gianpiero Sportelli

Publisher: Association for Computational Linguistics

Date: 13 September 2019

Place: Stockholm, Sweden

Venue: SIGDIAL

Link: <https://www.aclweb.org/anthology/W19-5930.pdf>

3.3.2 Summary

- The paper discusses a practical application of BERT, where the researchers created a question answering system for Italian language that gives information on a few specific subjects like digital billing and e-invoicing.
- A few question answer pairs that were collected were expanded and then fed as training data to the BERT model. The system is readily available for a demo on Telegram.
- An accuracy score of 84% was achieved using the external intent detection API, which is slightly lower to the previously achieved 86%.

3.3.3 Research gaps

- The results obtained cannot be compared to any other studies as the topic is very domain specific and there have not been any such experimentations in Italian language.
- The whole process is network dependent and can be time consuming under constrained networks.

3.4 Simple Applications of BERT for Ad Hoc Document Retrieval

3.4.1 Author and publication details

Authors: Wei Yang, Haotian Zhang, Jimmy Lin

Date: 26 March 2019

Place: Ontario, Canada

Venue: David R. Cheriton School of Computer Science, University of Waterloo

Link: <https://arxiv.org/pdf/1903.10972v1.pdf>

3.4.2 Summary

- The paper discusses the following ongoing accomplishments in applying BERT to address replying.
- Investigate basic applications to specially appointed document retrieval. This required standing up to the test presented by reports that are commonly longer than the length of information BERT was intended to deal with.
- The paper addresses this issue by applying deduction on sentences independently, and afterward accumulating sentence scores to create archive scores.

3.4.3 Research gaps

- Essentially avoided the issue of not having sentence-level importance decisions, despite the fact that there are some undeniable removed supervision methods to "venture" significance names down to the sentence level that ought to be investigated.

3.5 Efficient and Robust Question Answering from Minimal Context over Documents

3.5.1 Author and publication details

Authors: Sewon Min, Victor Zhong, Richard Socher, Caiming Xiong

Publisher: Association for Computational Linguistics

Date: July 15 - 20, 2018

Place: Melbourne, Australia

Venue: David R. Cheriton School of Computer Science, University of Waterloo

Link: [Efficient and Robust Question Answering from Minimal Context over Documents](#)

3.5.2 Summary

- The paper addresses the problem of scaling in question answering over documents to large corpora. The reason stated for this is the complex interaction between the documents and the question.
- The authors also realized that the existing models are sensitive to adversarial inputs and that minimal context is sufficient to answer a given question
- They also propose a sentence selector, which picks the minimum number of sentences required to feed to the QA models.
- Thus, the models were made scalable to large corpora and robust to adversarial questions.

3.5.3 Research gaps

- Although the framework is flexible and does not require end-to-end training, the combination of the framework and any existing QA model does not improve the speed. The SQuAD (with S-Reader) gave an F1 score of 79.9% and EM of 71.0% with no increase in training and inference speeds.

CHAPTER-4

PROJECT REQUIREMENTS SPECIFICATIONS

4.1 Modules

4.1.1 Module 1: User Interface

This module is used for taking the input from the user and also to present the output to the user. The interface requires the user to input the question which should be related to the Computer Security domain. This input is then sent from the user interface to the information retrieval module. The output received is shown as an answer to the user.

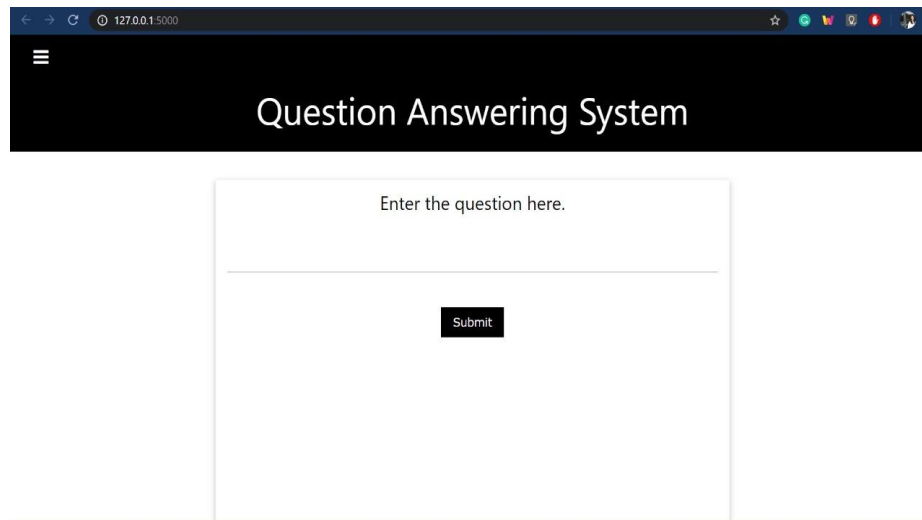


Fig 4.1 User Interface for the system

4.1.2 Module 2: Cache

This module is added to make the processing of the answer faster for the user and avoid re-computation of the answer by the model. This module is added between the document retrieval module and the user interface, taking the question as an input from the UI, and checking if the answer for it is saved on the cache. The cache saves the question-answer-context when a new question is asked and the model returns an answer to both the UI and the cache. When a question is asked, it first gets checked in the cache, of

size 140TB, and if the answer for this question exists, it gets sent to the UI directly. If the answer does not exist, it goes through the next 2 modules as usual to compute the answer.

4.1.3 Module 3: Information Retrieval

This module is used when the question is asked for the first time and is not found in the cache already. This module uses a pool of documents and retrieves the document which is closely related to the posed question. The pool of documents belongs to the ‘Computer Security’ topic. The documents each contain a particular context which may contain the answer to the posed question. The question which the user interface module sends is matched with the context present in these documents. Multiple information retrieval algorithms were tried out and the one used here is LSI. The document which matches the most is taken as the most relevant one. This document’s context is then sent to the next module as an input for searching for the exact answer.

4.1.4 Module 4: Question Answer System

This module takes the posed question from module 1 and the context chosen from module 3 as an input and aims to find the exact answer to the given question. This is done by using the BERT model by Google which is pre-trained to avoid the overhead of extra GPU and the extra time. The pre-trained model is then fine-tuned with one new layer on top to perform the task of question answering system. This fine-tuned question answering model is then trained with the domain specific data. This model then is fed with the context and question and results with the answer as an output. This output is then sent to module 1 for viewing, and module 2 to save for future faster retrievals.

4.2 Constraints

4.2.1 Design Constraints

- Training on domain specific corpus yields better performance than fine tuning BERT.
- BERT has a limitation on the sentence length.
- The higher the maximum length of the sentence, the more GPU memory or CPU time will

be required.

- Giving context along with the question limits the generic behaviour of the system.

4.2.2 System Constraints

- Requirement of high GPU memory affects the easy run on the local system and requires dependency on platforms like Google Colab and Google Cloud Platform (GCP).
- An interaction mechanism between the UI on the local system and GCP is required.

4.3 Product Perspective

The product is developed as part of the Final Year Project for the period January 2020 - May 2020.

4.3.1 User Characteristics

- End users are supposed to provide the question.
- User is responsible for the correctness of the question and the context.
- The scope of the question is limited to Computer Security, so the user needs to keep this in mind and pose only related questions.

4.3.2 General Constraints, Assumptions and Dependencies

4.3.2.1 Constraints

- The context provided should be within the domain of Computer Security in which the model is fine-tuned.
- The answer should exist in the context for the question.

4.3.2.2 Assumptions

- Questions asked will be relevant to the posed context.

- Answer is present in the context given and no null string is sent as a question.
- Syntax and semantics of the inputs are correct.
- Answer matches in the later part of the context may overshadow the earlier ones.
- The length of the sentence is limited to match with the memory available.
- Fine tuning the BERT model is done with the help of domain specific data.

4.3.2.3 Software Dependencies:

- Technologies: Pytorch, TensorFlow, Flask, Ngrok
- Tools: GPU by Google Colab, Anaconda Navigator
- Database: SQLite
- Google Drive ,Google Cloud Platform (GCP) buckets, Google Colab
- Web Development: HTML, JavaScript

4.3.2.4 Hardware Dependencies:

- Working PC
- RAM size > 8GB
- Internet connection

4.3.2 Risks

- Any context and question which is out of the domain will result in an unexpected answer.
- The answers can arbitrary span within context paragraphs rather than a limited set of multiple choices.

- The performance of the model might worsen when there is syntactic divergence between the question and the sentence containing the answer.
- Time for a fresh question can vary depending on the network speed available at the time.

CHAPTER-5

SYSTEM REQUIREMENTS SPECIFICATION

5.1 Functional Requirements

5.1.1 Picking the right document

- The Question Answering System is supposed to pick the document that matches the input question the most. Ideally, this should be the document that contains the answer to the question.
- In case the input is a question with no answer, then most linguistically and semantically similar document to the question should be returned.

5.1.2 Finding the answer in the selected context

- Once the document containing the context is selected, the Question Answering System should return the span, or segment of text as the answer to the input question.
- To the questions that have no answer, a null string should be returned.

5.1.3 Picking the document and answer from the cached data

- Once the question is entered by the user, the cache is required to check if an answer and a context exists for this question. If it exists the cache is supposed to send the answer as an output to the UI.
- This reduces the response time of the system as it skips going through other phases which are more computation intensive.

5.2 Non-Functional Requirements

5.2.1 Performance

- Response time - The model takes about 3-5 seconds to compute the answer to a fresh

question and less than a second to return answer to a previously asked question.

- Interoperability - The API service of the model can be accessed from any well-known browser with ease.

5.2.2 Availability

- The uptime of the model can be controlled by running the api.py file.
- The model works seamlessly as long as the program runs.

5.2.3 Capability

- Maximum length of the question is set at 70 characters. It can be changed in the code. The user is issued a warning if it exceeds this value.
- Maximum size of the cache - The SQLite database used has a max capacity of approximately 140TB, beyond which the oldest questions will be deleted.

5.2.4 Usability

- The application is designed for easy usage. The user will only have to enter the question in the text area provided.
- Users can also choose one of the previously asked questions for faster response.

5.2.5 Documentation

The documentation for the question answering system is provided which explains the various modules used in the project. This documentation is made available in the UI for help.

5.3 Hardware Requirements

- The application is built using Flask and is made available through a public URL created by the flask-ngrok.
- The users can access the application either on their PCs or their Mobile devices.

- The users will also need to have a secure and speedy internet connection to access the URL.

5.4 Software Requirements

- The users must have a browser installed in their device.
- The ngrok provides a secure URL to the localhost server where the application is hosted, through any NAT or firewall.

CHAPTER-6

SYSTEM DESIGN

Question Answering (QA) framework is a document retrieval framework in which an explicit reply answer is required because of a submitted inquiry, instead of a lot of references that may contain the appropriate responses. It is a man machine specialized gadget. The fundamental thought of QA frameworks in Natural Language Processing (NLP) is to give right responses to the inquiries for the students.

The question answering framework can be portrayed as a four stage process: question definition, document retrieval, answer extraction and answer caching. The initial 2 stages can be achieved by the document retrieval strategies, answer extraction is achieved by utilizing a pre-prepared BERT model, and the caching stage improves the response time of the model.

The model's ability of responding to the question relies upon the intricacy of the question and the picked document as a unique circumstance. If the right document is picked then the probability of the model answering the question is 0.85. The BERT model has a limitation on the length of the context, so as to beat this procedure it has been chosen to part the context into various paragraphs and pick the most relevant section as a context for the question.

Stanford Question Answering Dataset (SQuAD) is another perusing understanding dataset, comprising inquiries presented by crowd workers on a lot of Wikipedia articles, where the response to each address is a section of content, or range, from the comparing understanding entry. With 100,000+ inquiry answer matches on 500+ articles, SQuAD is fundamentally bigger than past perusing understanding datasets.

The diagram below shows the various modules in the design of the system. Each module is itself a complete entity, it has its own significance and design constraints. Each module is independently built and integrated. User interface is designed in such a way that any person with little knowledge on websites can use it without any problem. The document retrieval phase is tried out with different IR methodologies and based on the results obtained LSI was chosen as IR methodology. The QA model, built on pre-trained BERT because BERT is better when compared to some older CNN, RNN models.

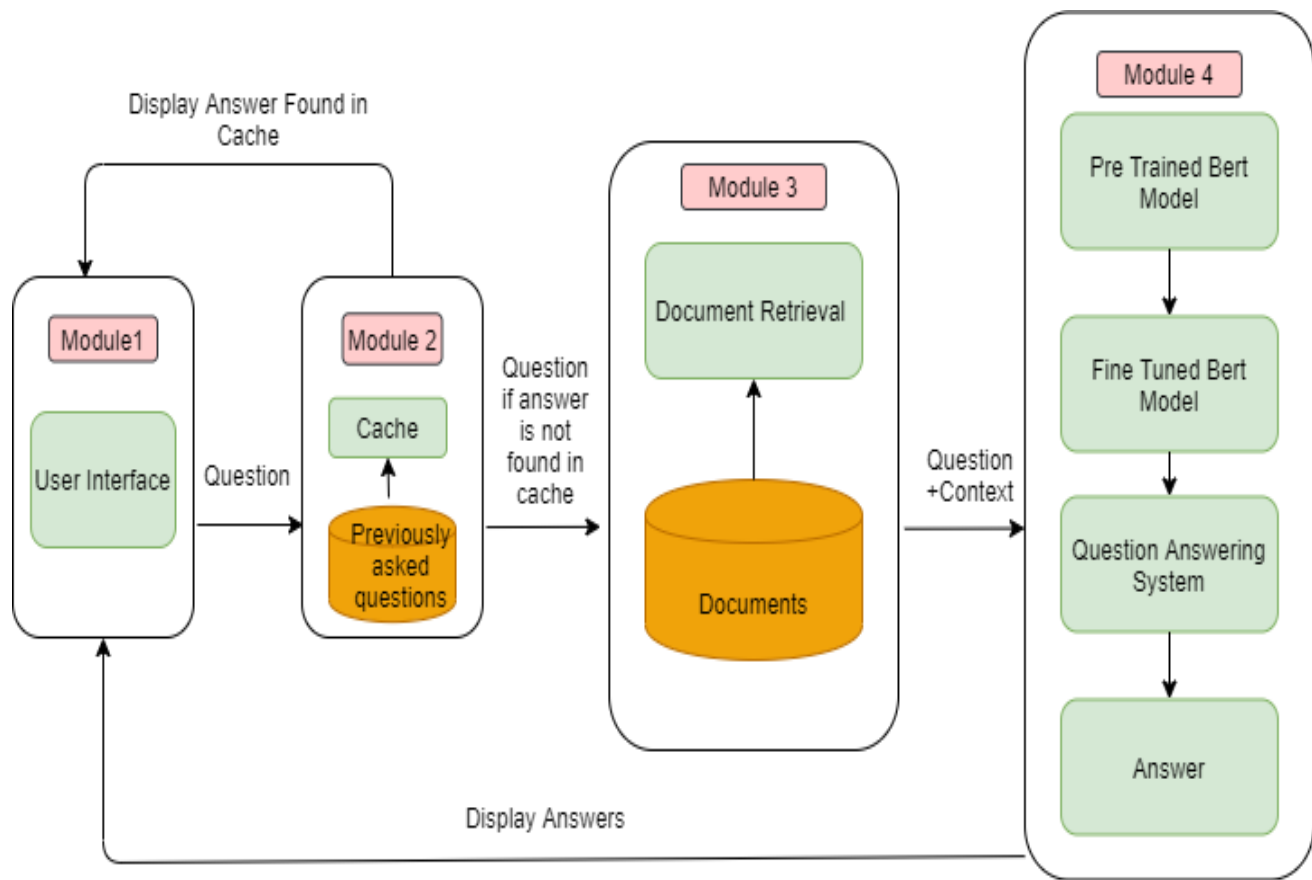


Fig 6.1 System Design with detailed modules.

The performance of the framework relies upon the precision and the response time of the appropriate response. Accuracy is limited to the heuristics utilized in the usage of the model and response time is limited by the calculation power of all the four stages. Latency stowing away is a significant structure goal that improves apparent execution of the question answer framework extensively. The thought behind this rule is the suspicion that any I/O required in a large calculation process is tedious and not CPU intensive.

A portion of the calculation processes are calculation concentrated being referred to answer framework. To diminish the time taken it is important to utilize heuristics to lessen measure of information sent to

QA model. To achieve this, keeping the time complexity in mind, ‘fetch from cache’ is a very important step since question answering forms are profoundly calculation escalated.

The database table is acquainted with a store recently responding to questions for snappier recovery with the presumption that when question is posed, it might be asked again in not so distant time. So this reduces the time taken during the document retrieval and answer extraction.

So as to place the multifaceted nature of the task in a reasonable degree, the questions identified with just ‘computer security’ domain are chosen to be utilized. There is commonly no restriction on the multifaceted nature of the question posed. The BERT model answers anything given an appropriate context. To simply make it a basic model, the area of the model has been confined to one computer science subject.

CHAPTER-7

DETAILED DESIGN

7.1 Component Diagram

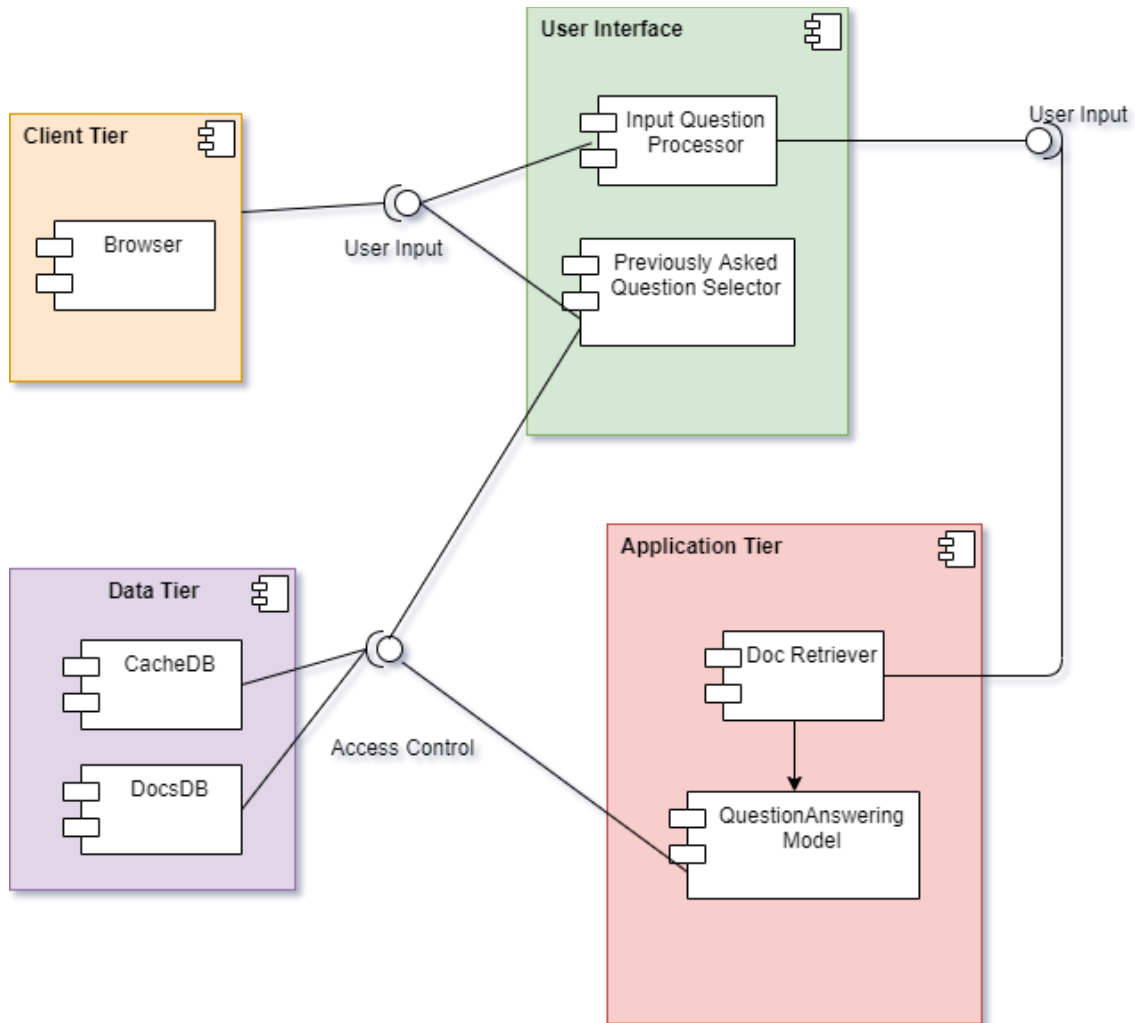


Fig 7.1 Component Diagram

7.1.1 Components:

1. Client Tier: This component contains the direct interface with the clients who enter the question for which answer has to be returned.
2. User Interface: This component comes in action directly after taking the input from the user and decides which next step will be taken by making a choice between the following two sub components:
 - a. Input Question Processor: This component is used when the question has not been asked previously. This component takes the input question and forwards it to the system which generated the answer to it after choosing the relevant document from the Document DB.
 - b. Previously Asked Question Selector: This component selects the answer and context from the Cache DB if the question is previously asked. This component then returns the answer to the UI.
- 3.Data Tier: This component contains the database for different functionalities of the system:
 - a. Cache DB: This database contains the previously asked questions' answer and context. This database is updated every time a new question is asked and is retrieved whenever an old question is asked. It helps by reducing the time to process the answer and reselect the relevant document by directly providing the answer to the user.
 - b. Documents DB: This database contains all the different documents and their contexts which are under the domain of 'computer security'. These docs are utilized in finding the relevance with the posed question and their answers if they contain any.
- 4.Application Tier
 - a. Document Retriever: It takes the question posed and the documents from Documents DB as input. This component computes the relevance between the documents and the posed question and chooses the document with higher relevance for answer finding.

- b. Question Answering Model: This model takes the most relevant document and the question as input and finds the answer for the question in this document's context, if it exists. It then stores this answer and context with the question in the Cache DB and returns the answer to the UI.

7.1.2 Interfaces:

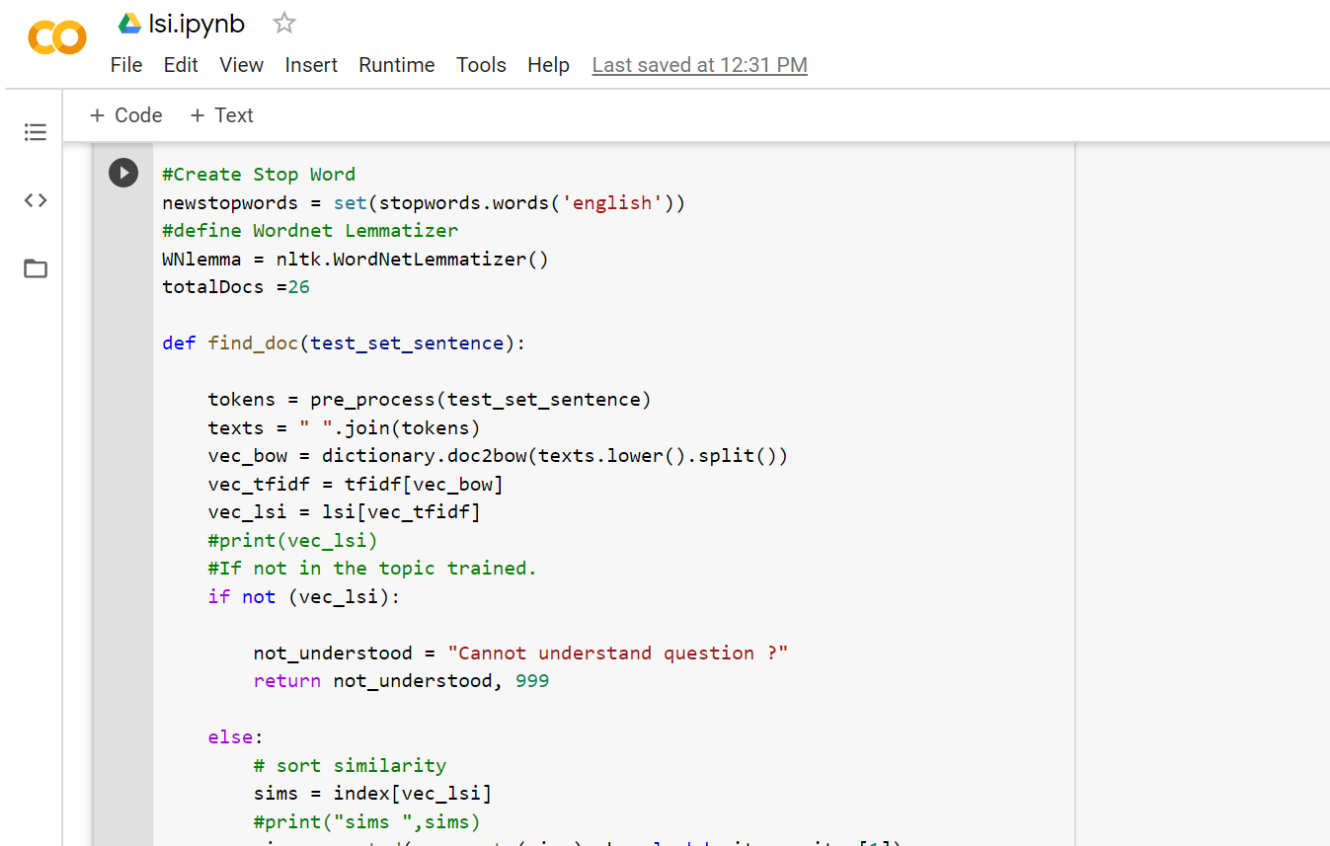
1. User Input 1: This interface is between the Client Tier and the User Interface components. It sends the user input, i.e., questions from the Client Tier, to the User Interface.
2. User Input 2: It sends the question posed from the Input Question Processor to the Document Retriever when the question posed isn't an old question.
3. Access Control: This connects the data tier with the previously asked question selector and with question answering model. This interface sends the data from the cache DB to the previously asked question selector and from the Docs DB to the question answering model component, as and when which is required.

7.2 User Interfaces

The user interfaces used for the development of the system are as follows:

7.2.1 Google Colab

The programming and the execution of the Python commands is done on this platform. This platform contains a temporary file storage and is connected to the google drive and GCP as well. All the commands are run to use the necessary GPUs as this amount of memory isn't available on the local systems.



The image shows a Google Colab notebook interface. At the top, there is a header bar with the Colab logo, the filename 'lsi.ipynb', and a star icon. Below this is a menu bar with options: File, Edit, View, Insert, Runtime, Tools, Help, and a link 'Last saved at 12:31 PM'. The main workspace is divided into two panes: a left sidebar with icons for file explorer, code editor, and output, and a right pane for output. The code editor pane contains the following Python code:

```
#Create Stop Word
newstopwords = set(stopwords.words('english'))
#define Wordnet Lemmatizer
WNlemma = nltk.WordNetLemmatizer()
totalDocs =26

def find_doc(test_set_sentence):

    tokens = pre_process(test_set_sentence)
    texts = " ".join(tokens)
    vec_bow = dictionary.doc2bow(texts.lower().split())
    vec_tfidf = tfidf[vec_bow]
    vec_lsi = lsi[vec_tfidf]
    #print(vec_lsi)
    #If not in the topic trained.
    if not (vec_lsi):

        not_understood = "Cannot understand question ?"
        return not_understood, 999

    else:
        # sort similarity
        sims = index[vec_lsi]
        #print("sims ",sims)
        sims = sorted(enumerate(sims) -> key=lambda item: item[1])
```

Fig 7.2 Google Colab

CHAPTER-8

IMPLEMENTATION AND PSEUDOCODE

8.1 Collecting and preparing data

8.1.1 Context

The dataset used for this question answering system is called SQuAD 2.0. The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset that was developed at Stanford. The dataset comprises inquiries presented by crowd workers on a lot of Wikipedia articles. The response to each address is a section of content, or range, from the respective paragraph. There are more than 100,000 question-answer pairs on more than 500 articles. Some key features of this dataset are,

- It is a closed dataset, that is, the answer to the given question is always part of the provided context.
- Answers are always a continuous span of context.
- So the problem of finding an answer is simplified as determining the start and end indexes of the context.
- Approximately 75% of answers are of length less than 5 words.

8.1.2 Content

There are two files to get started with the dataset and evaluate and train the models:

- train-v1.1.json - 41.13 MB
- dev-v1.1.json - 4.2 MB

Sample Context from SQuAD 2.0 :

“The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who,

under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. Through generations of assimilation and mixing with the native Frankish and Roman-Gaulish populations, their descendants would gradually merge with the Carolingian-based cultures of West Francia. The distinct cultural and ethnic identity of the Normans emerged initially in the first half of the 10th century, and it continued to evolve over the succeeding centuries.”

Sample Question and Ground Truth answer:

Question: In what country is Normandy located?

Ground Truth Answer: France

8.1.3 Domain Specific Data Preparation

8.1.3.1 Implementation

The dev-v1.1.json file of SQuAD 2.0 contains a variety of topics for its contexts. They include Normans, Computational_complexity_theory, Packet_switching, etc. For making the project domain specific, only the contexts from this file under the topic “computer security” are chosen. These contexts are split into various documents to give an idea of how the documents are indexed in search engines. Length of each document can vary.

8.1.3.2 Pseudocode

```
import json
f = open("/content/drive/My Drive/train-v2.0.json", "r")
CT = ['Computer_security']
d = f.read()
d1 = json.loads(d)
l = d1['data']
data2 = []
for i in range(len(l)):
    if(l[i]['title']) in CT:
        data2.append(l[i]['paragraphs'])
```

Fig 8.1 Pseudocode for domain specific data

8.2 Training the model

There are two stages of preparing the BERT model for use. They are pre-training and fine-tuning.

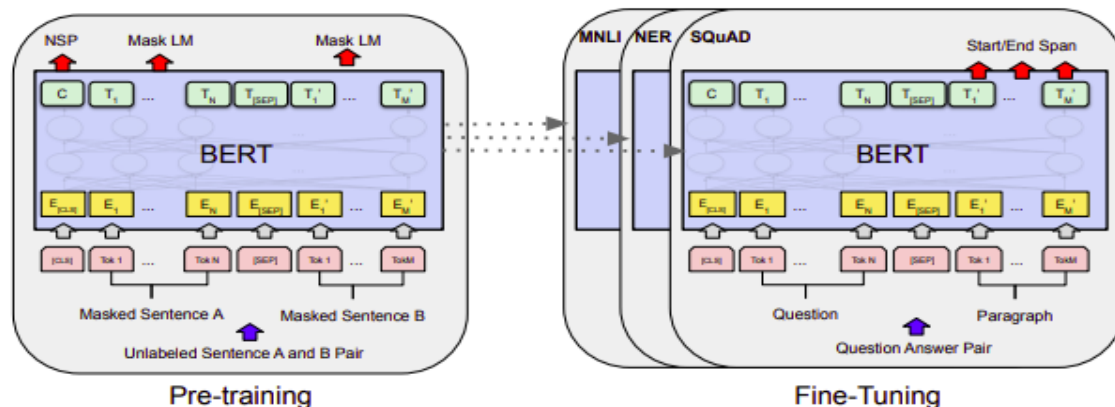


Fig 8.2 Training the model
Source: Adapted from [12]

8.2.1 Pre-Training

The pre-preparing method to a great extent follows the current writing on language model pre-training. For the pre-training corpus, the BooksCorpus (800M words) and English Wikipedia (2,500M words) were utilized. The whole process is fairly expensive and takes 4 days on 4 to 16 Cloud TPUs. Hence the pre-trained model released by Google is used.

8.2.2 Fine-Tuning

Fine-tuning is more straightforward than pre-training. It is inexpensive, takes at most 1 hour on a single Cloud TPU, or a few hours on a GPU. Time was significantly cut down using the TPU environment in Colab.

Two models were explored for their properties, and to examine which suits better for the purpose. The following table shows a comparison of the two.

```
!python /content/evaluate-v2.0.py /content/bert/dev-v2.0.json /content/bert_output_predictions.json
```

```
{
  "exact": 75.98753474269351,
  "f1": 79.25590666413233,
  "total": 11873,
  "HasAns_exact": 76.82186234817814,
  "HasAns_f1": 83.36797905250371,
  "HasAns_total": 5928,
  "NoAns_exact": 75.1555929352397,
  "NoAns_f1": 75.1555929352397,
  "NoAns_total": 5945
}
```

Fig 8.3 Training of the BERT-Large, Uncased model.

It is an approach where a snapshot of the state of the system is taken in case of system failure. If there is a problem, not all is lost. The checkpoint may be used directly, or used as the starting point for a new run, picking up where it left off.

During constrained networks, checkpoints helped in restarting the training process from where it was left off, rather than start it all over again.

```
[ ] !python /content/evaluate-v2.0.py /content/bert/dev-v2.0.json /content/bert_output_base_model_predictions.json
```

```
{
  "exact": 75.95384485808137,
  "f1": 79.1827983486221,
  "total": 11873,
  "HasAns_exact": 76.38326585695006,
  "HasAns_f1": 82.85043265742077,
  "HasAns_total": 5928,
  "NoAns_exact": 75.52565180824222,
  "NoAns_f1": 75.52565180824222,
  "NoAns_total": 5945
}
```

Fig 8.4 Training of the BERT-Base, Uncased model.

Feature	BERT-Base, Uncased:	BERT-Large, Uncased
Number of Transformer Blocks	12	24
Number of hidden layers	768	1024
Attention Heads	12	16
Exact Match	75.95384485808137	75.98753474269351
F1 Score Obtained	79.1827983486221	79.25590666413233

Table 8.1 Comparison of the two models

BERT-Large, Uncased model has a higher accuracy of 79.255%. Hence, this model was chosen.

The Local File System is not supported on TPU. Also output directories don't remain after the session is complete. So, the pre-trained models were moved to the GCS bucket and drive storage was used to store the codes.

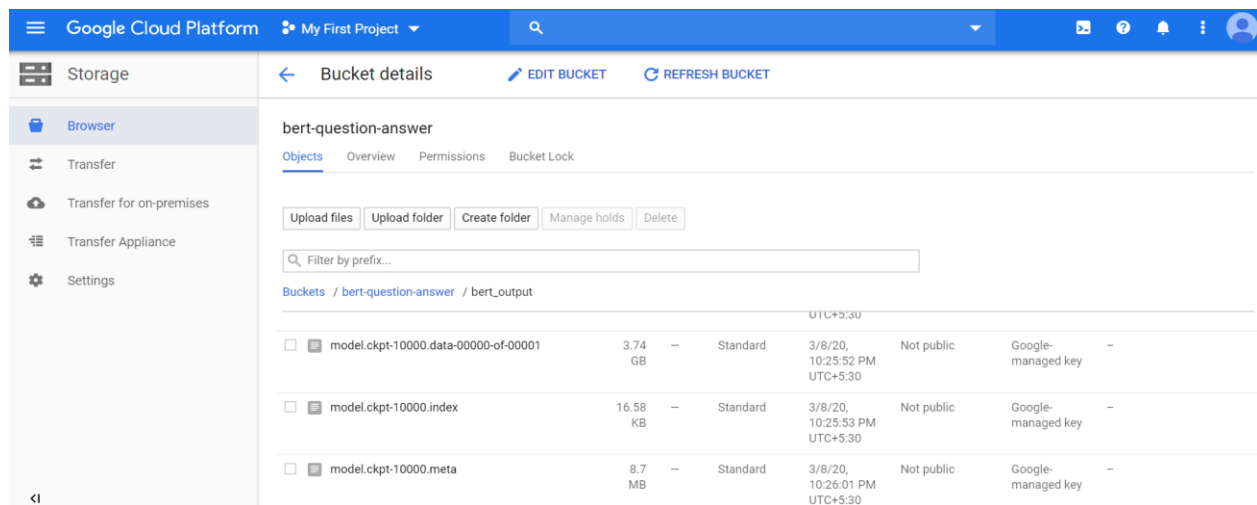


Fig 8.5 Model Checkpoints created in GCS bucket after training

8.3 Picking the Relevant Document

8.3.1 Handling the length constraint

8.3.1.1 Implementation

The BERT model tends to give higher errors while predicting if the context length is higher. The official maximum length of the context is declared as 512 characters. Hence, the documents that have length greater than 512 chars were broken down into chunks of sentences that have length less than or equal to 512 chars.

8.3.1.2 Pseudocode

```
51
52     #Handle length constraint
53     for i in doc_names:
54         file = open(os.getcwd()+'/context_Sec/'+i,"r")
55         for line in file:
56             file_text = file_text+line
57             if(len(file_text))>threshold_sequence_len:
58                 file_text_list.append(file_text)
59                 file_text = ""
60
```

Fig 8.6 Pseudocode Handling lengthier context

8.3.2 Technique to find the most relevant document

8.3.2.1 Implementation

The technique followed to calculate the similarity between query and the pool of documents is latent semantic index which uses singular value decomposition to form LSI vectors. A comparison of LSI vectors of query and documents is done with cosine similarity. The LSI model is prepared with 800 topics. The genism library in python was used to implement the same.

8.3.2.2 Pseudocode

```

46 def prep_model():
47     file_text_list = []
48     file_text = ""
49     doc_names = os.listdir(os.getcwd()+'/context_Sec/')
50
51
52     #Handle length constraint
53     for i in doc_names:
54         file = open(os.getcwd()+'/context_Sec/'+i,"r")
55         for line in file:
56             file_text = file_text+line
57             if(len(file_text))>threshold_sequence_len:
58                 file_text_list.append(file_text)
59                 file_text = ""
60
61     # LSI implementation
62     docs = [pre_process(doc) for doc in file_text_list]
63     dictionary = corpora.Dictionary(docs)
64     corpus = [dictionary.doc2bow(a) for a in docs]
65     tfidf = models.TfidfModel(corpus)
66     corpus_tfidf = tfidf[corpus]
67     lsi = models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=800) # Threshold A
68     corpus_lsi = lsi[corpus_tfidf]
69     index = similarities.MatrixSimilarity(corpus_lsi)
70     return dictionary,tfidf,lsi,index,docs,file_text_list
71

```

Fig 8.7 Pseudocode Document retrieval

8.4 Finding the right answer

8.4.1 Implementation

Once the right context segment is determined, this is then passed, along with the input question, as input to the model. The model sends the context and question and the following process takes place:

- Pre-processing - Conversion of input passage and question into a SQuAD Example by removal of whitespaces and tokenizing it.
- Addition of SEP and CLS tokens:
 - [SEP] is for separating sentences
 - [CLS] is placed at the beginning of the input example sentence/sentence pair.
- Collecting the following parameters essential for making input Tensor Dataset to the model:
 - input_ids
 - input_mask

- segment_ids
- example_index
- Sampling the elements using Sequential Sampler and loading the data using Dataloader.
- Evaluate the data batch wise and obtain the answer.

8.4.2 Pseudocode

```

45 def predict(self, passage :str, question :str):
46     example = input_to_squad_example(passage, question)
47     features = squad_examples_to_features(example, self.tokenizer, self.max_seq_length, self.doc_stride, self.max_query_length)
48     all_input_ids = torch.tensor([f.input_ids for f in features], dtype=torch.long)
49     all_input_mask = torch.tensor([f.input_mask for f in features], dtype=torch.long)
50     all_segment_ids = torch.tensor([f.segment_ids for f in features], dtype=torch.long)
51     all_example_index = torch.arange(all_input_ids.size(0), dtype=torch.long)
52     dataset = TensorDataset(all_input_ids, all_input_mask, all_segment_ids,
53                             all_example_index)
54     eval_sampler = SequentialSampler(dataset)
55     eval_dataloader = DataLoader(dataset, sampler=eval_sampler, batch_size=1)
56     all_results = []
57     for batch in eval_dataloader:
58         batch = tuple(t.to(self.device) for t in batch)
59         with torch.no_grad():
60             inputs = {'input_ids':      batch[0],
61                       'attention_mask': batch[1],
62                       'token_type_ids': batch[2]}
63             example_indices = batch[3]
64             outputs = self.model(**inputs)
65
66             for i, example_index in enumerate(example_indices):
67                 eval_feature = features[example_index.item()]
68                 unique_id = int(eval_feature.unique_id)
69                 result = RawResult(unique_id = unique_id,
70                                   start_logits = to_list(outputs[0][i]),
71                                   end_logits   = to_list(outputs[1][i]))
72             all_results.append(result)
73     answer = get_answer(example, features, all_results, self.n_best_size, self.max_answer_length, self.do_lower_case)
74     return answer
75
76

```

Fig 8.8 Pseudocode Answer Prediction

CHAPTER-9

TESTING

9.1 Strategies

1. Unit Testing: It was done when new features, document retrieval and cache, were added to the system. This was done before these units was added to the system.
2. Component Testing: It was done to check the interaction between the various modules.
3. System Testing: It was done to check the complete working of the system.
4. Acceptance Testing: It was done to test the final complete working of the entire system, and accept the current model.

These strategies are used in order to follow a systemized testing pattern and not have a big bang approach to it. Directly performing system testing may result in failures which could have been solved by just unit or component testing.

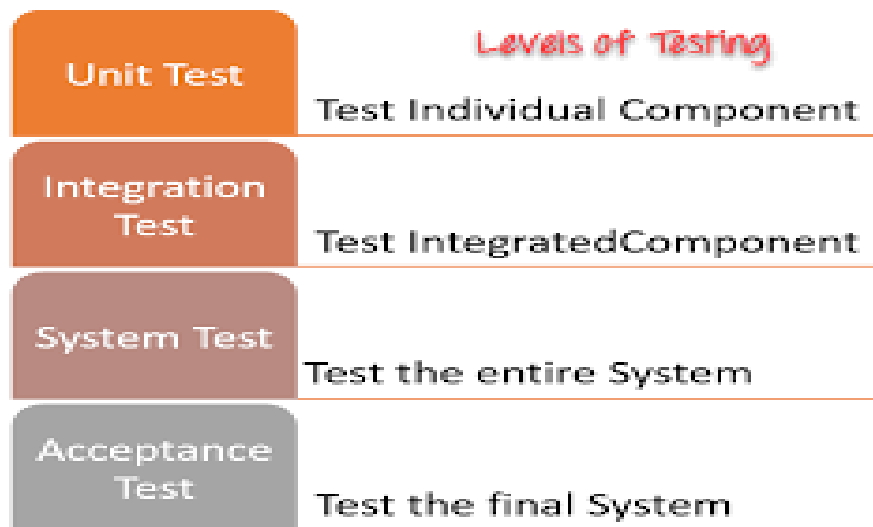


Fig 9.1 Testing Strategies
Source: Adapted from [10]

9.2 Test Cases

- Test case 1: Check interaction between UI and Document retrieval
- Test Case 2: Check if the proper document is picked given a query.
 - Tested with a lot of queries and it picks the right document 85 - 90% of the time.
 - Wrote a python script which automatically does this.
- Test Case 3: Check if the interaction between document retrieval and model.
- Test Case 4: Check if expected result is returned.

CHAPTER-10

RESULTS AND DISCUSSION

10.1 Model Evaluation

Fine-tuned the model with one output layer and accomplished accuracy of 80%. The evaluation of the model was based on two measures.

- Exact match: A double proportion of whether the framework yield matches the ground truth answer precisely
- F1 score: Harmonic mean of precision and recall, where
 - $\text{precision} = (\text{true positives}) / (\text{true positives} + \text{false positives})$
 - $\text{recall} = \text{true positives} / (\text{false negatives} + \text{true positives})$
 - $\text{F1 score} = (2 \times \text{precision} \times \text{recall}) / (\text{precision} + \text{recall})$

10.2 Document Retrieval for query

- Various procedures like TF-IDF, Latent Semantic Index for document retrieval were attempted. It was found that LSI works the best for the required system.
- The strategy followed to ascertain the comparability among inquiry and the pool of records is latent semantic analysis which utilizes particular worth disintegration to shape LSI vectors. LSI vectors of inquiry and reports with cosine likeness are focused at.
- The LSI model is set up with 800 themes.
- Utilized python gensim library to construct LSI model for document retrieval. Accomplished accuracy of about 90%.
 - The model was evaluated using a script with questions and manually checked if it matches the expected result.

10.3 Answer Prediction for given query from the fetched document

- Used the functions provided by the pre-trained BERT model.

- BERT has a length limit on the context provided. The lengthier contexts were taken care of by breaking down the context into smaller paragraphs and choosing the most relevant part.
- Model returns the correct answer 85-90% of the time.

10.4 Increase in response time by caching

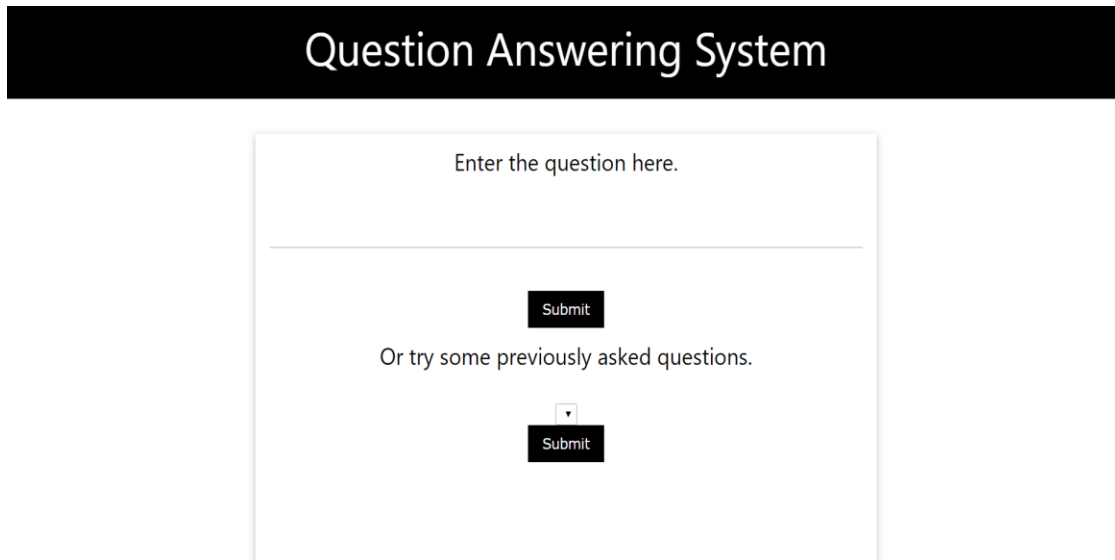
- By means of caching, previously asked questions, assuming question is most likely to be posed again, avoids going through other phases which are computation intensive.
- It also decreases the load on the server which performs some intensive tasks.

Question answering framework is one of the developing territories of research in common language preparing applications of Artificial Intelligence. QA frameworks intend to deliver precise answers, however the current QA built framework is succeeded just somewhat. As guaranteed before, all the predefined expectations have been met and furthermore the model is stretched out to deal with a pool of documents and a lengthier context. These expectations are accomplished with a decent exactness and likewise a UI is made for simple utilization and introduction. In this manner an attempt to expand the present use of the BERT model from a basic question answering model to incorporate document retrieval and furthermore quick access utilizing store has been made.

CHAPTER-11

SNAPSHOTS

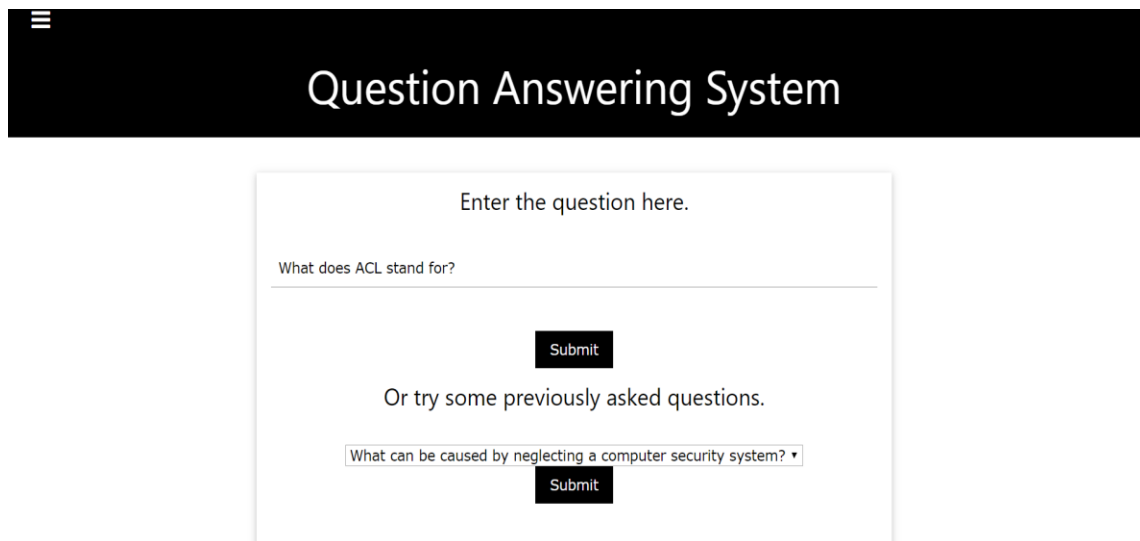
11.1 Homepage of the application



The screenshot shows the homepage of the 'Question Answering System'. At the top, there is a black header bar with the text 'Question Answering System' in white. Below the header, there is a white box with a light gray border. Inside this box, the text 'Enter the question here.' is displayed at the top. Below this text is a horizontal line. Under the line, there is a black 'Submit' button. Below the button, the text 'Or try some previously asked questions.' is displayed. Below this text is a dropdown menu with a small downward arrow icon, and below the dropdown menu is another black 'Submit' button.

Fig 11.1 Homepage

11.2 Query Submission



The screenshot shows the query submission page of the 'Question Answering System'. At the top, there is a black header bar with the text 'Question Answering System' in white. Below the header, there is a white box with a light gray border. Inside this box, the text 'Enter the question here.' is displayed at the top. Below this text is a horizontal line. Under the line, the text 'What does ACL stand for?' is displayed. Below this text is a black 'Submit' button. Below the button, the text 'Or try some previously asked questions.' is displayed. Below this text is a dropdown menu with the text 'What can be caused by neglecting a computer security system?' and a small downward arrow icon. Below the dropdown menu is another black 'Submit' button.

Fig 11.2 Query submission

11.3 Document Retrieved

Within computer systems, two of many security models capable of enforcing privilege separation are access control lists (ACLs) and capability-based security. Using ACLs to confine programs has been proven to be insecure in many situations, such as if the host computer can be tricked into indirectly allowing restricted file access, an issue known as the confused deputy problem. It has also been shown that the promise of ACLs of giving access to an object to only one person can never be guaranteed in practice. Both of these problems are resolved by capabilities. This does not mean practical flaws exist in all ACL-based systems, but only that the designers of certain utilities must take responsibility to ensure that they do not introduce flaws.[citation needed]

Fig 11.3 Document retrieved

11.4 Display Answer

Question Answering System

The answer is

access control lists

It is chosen from the context

Within computer systems, two of many security models capable of enforcing privilege separation are access control lists (ACLs) and capability-based security. Using ACLs to confine programs has been proven to be insecure in many situations, such as if the host computer can be tricked into indirectly allowing restricted file access, an issue known as the confused deputy problem. It has also been shown that the promise of ACLs of giving access to an object to only one person can never be guaranteed in practice. Both of these problems are resolved by capabilities. This does not mean practical flaws exist in all ACL-based systems, but only that the designers of certain utilities must take responsibility to ensure that they do not introduce flaws.[citation needed]

Fig 11.4 Answer returned

11.5 Previously Asked questions

Question Answering System

Enter the question here.

what does ACL stand for ?

Or try some previously asked questions.

what does ACL stand for ?

what does ACL stand for ?

What can be caused by neglecting a computer security system?

What does CCIRC stand for?

Firewalls and exit procedures are considered what?

What is computer security also known as?

Fig 11.5 Cached results

CHAPTER-12

CONCLUSIONS

Over the span of this semester, the various available models and frameworks for question answering systems have been explored. A deep dig into their functioning and their likening to the users has been made. A search for the various enhancements done and how recent they were in terms of development has been done. Finally the BERT model by Google is chosen, since at present, it is the best answer for this requirement. There are a lot of resources and developments done in this model and it also is fully supported by the Google community.

BERT is a pre-trained model which is utilized by a ton of specialists for developing state of the art models. The BERT was utilized and calibrated to make the question answer explicit. Integrating BERT with IR tackled the issue of a pool of documents. Storing the outcomes in the database improved the time unpredictability radically.

Google colab was very convenient to use and install all the python deep learning libraries.

At last, the result of the undertaking is a web application coordinated with a question answering model which is presently ‘computer security’ explicit. Coordinating the model with LSI and caching improved the exhibition of the framework.

CHAPTER-13

FURTHER ENHANCEMENTS

The model fabricated for this project is built in an area explicit to computer security. The long term objective is to make it a computer science question answering bot which can be utilized significantly for one-on-one learning. This will help in expanding the domain of the system from one sector of Computer Science to all sectors of it. This will also act like an encyclopaedia of the Computer Science domain which will be very helpful for learning the domain.

Likewise, another feature can be included to take the feedback from the client for the application and investigate the feedback taken for the present application and improve the model more by adjusting the model by differing parameters. This feedback can be used to fine-tune the model to better the latest user requirements.

The next enhancement to implement in the future would be to explore and use better and more proficient information retrieval techniques. These techniques would help in finding a better matched document with more accuracy for a given question. The IR approaches utilized here are the most fundamental strategies in light of the fact that lesser information is available in the form of documents. Utilizing progressively refined IR strategies will make the document retrieval process increasingly proficient.

REFERENCES/BIBLIOGRAPHY

- [1] C. McCormick. “BERT Research - Ep. 1 - Key Concepts & Sources.” mccormickml.com. <https://mccormickml.com/2019/11/11/bert-research-ep-1-key-concepts-and-sources> (accessed Jan. 15, 2020).
- [2] M. S.Z. Rizvi. “Demystifying BERT: A Comprehensive Guide to the Groundbreaking NLP Framework.” analyticsvidhya.com. <https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework> (accessed Jan. 15, 2020).
- [3] M. Kana. “BERT for dummies – Step by Step Tutorial.” towardsdatascience.com. <https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03> (accessed Jan. 20, 2020).
- [4] T. Rajapakse. “Question Answering with BERT, XLNET, XLM, and DistilBERT using Simple Transformers.” towardsdatascience.com. <https://towardsdatascience.com/question-answering-with-bert-xl-net-xlm-and-distilbert-using-simple-transformers-4d8785ee762a> (accessed Jan. 21, 2020).
- [5] N. Kiran. “Extending Google-BERT as Question and Answering model and Chatbot.” medium.com. <https://medium.com/datadriveninvestor/extending-google-bert-as-question-and-answering-model-and-chatbot-e3e7b47b721a> (accessed Jan. 22, 2020).
- [6] C. McCormick and N. Ryan. “BERT Word Embeddings Tutorial.” mccormickml.com. <https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/#2-input-formatting> (accessed Jan. 22, 2020).
- [7] P. Joshi. “How do Transformers Work in NLP? A Guide to the Latest State-of-the-Art Models.” analyticsvidhya.com. <https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models> (accessed Jan. 23, 2020).
- [9] J. Devlin and M. Chang. “Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing.” ai.googleblog.com. <https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html> (accessed Jan. 28, 2020).

- [10] “Levels of Testing in Software Testing.” guru99.com. <https://www.guru99.com/levels-of-testing.html> (accessed April 8, 2020).
- [11] R. Mervin. (2013). “An Overview of Question Answering System.”
“https://www.researchgate.net/publication/320978810_An_Overview_of_Question_Answering_System”.
- [12] J.C.B. Cruz and C. Cheng. (2019). “Evaluating Language Model Finetuning Techniques for Low-resource Languages.” 10.13140/RG.2.2.23028.40322.
- [13] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova. (2019). “Bert: Pre-training of deep bidirectional transformers for language understanding.” arXiv preprint, arXiv:1810.04805v2.
- [14] Y. Zhang and Z. Xu. (2019). “BERT for Question Answering on SQuAD 2.0.”
- [15] J. Cheng, L. Dong and M. Lapata. (Sep. 20, 2016). “Long Short-Term Memory-Networks for Machine Reading”. arXiv:1601.06733v7.
- [16] P. Rajpurkar, J. Zhang, K. Lopyrev and P. Liang. (Oct. 11, 2016). “SQuAD: 100,000+ Questions for Machine Comprehension of Text.” arXiv:1606.05250v3.
- [17] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer. (2018). “Deep contextualized word representations.” In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.
- [18] D. Chen, A. Fisch, J. Weston and A. Bordes. (Apr. 28, 2017). “Reading Wikipedia to Answer Open-Domain Questions.” arXiv:1704.00051v2.
- [19] W. Yang, Y. Xie, A. Lin, X. Li, L. Tan, K. Xiong, M. Li and J. Lin. (Sep. 18, 2019). “End-to-End Open-Domain Question Answering with BERTserini.” arXiv:1902.01718v2.

[20] S. Min, V. Zhong, R. Socher and C. Xiong. (July 15, 2018). “Efficient and Robust Question Answering from Minimal Context over Documents.”

[21] F. Alloatti, L.D. Caro and G. Sportelli. (Sep. 13, 2019). “Real Life Application of a Question Answering System Using BERT Language Model.”

[22] S.C. Tirpude and Dr. A.S. Alvi. (June 6, 2015). “Closed Domain Keyword based Question Answering System for Legal Documents of IPC Sections & Indian Laws.” Vol. 3.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser and I. Polosukhin. (2013). “Attention Is All You Need.” arXiv:1706.03762v5.

[24] B. van Aken, B. Winter, A. Löser and F.A. Gers. (Sep. 11, 2019). “How Does BERT Answer Questions? A Layer-Wise Analysis of Transformer Representations.” arXiv:1909.04925v1.

[25] S. Schwager and J. Solitario. (2019). “Question and Answering on SQuAD 2.0: BERT Is All You Need.”

Appendix A

DEFINITIONS, ACRONYMS AND ABBREVIATIONS

The keywords given underneath are utilized in the archive unambiguously. The importance of these keywords stay as characterized here, except if demonstrated in any case:

- BERT - Bidirectional Encoder Representations from Transformers. BERT's key specialized advancement is applying the bidirectional preparation of Transformer, a well-known consideration model, to language displaying.
- NLP - Natural Language Processing. NLP is a field of Artificial Intelligence that enables the machines to peruse, comprehend and get significance from human dialects. It can be used to pre-process the data.
- IR - Information Retrieval. IR is the study of scanning for data in a report, looking for documents themselves, and furthermore looking for the metadata that portrays information, and for databases of writings, pictures or sounds.
- QA - Question Answering System
- Colab - The environment provided by google to code in python with deep learning libraries like Pytorch, keras, tensorflow.
- LSI - Latent Semantic Indexing. LSI is a method in NLP-IR, specifically distributional semantics, of investigating connections between a lot of documents and the terms they contain by creating a lot of ideas identified with the reports and terms
- SQUAD - Stanford Question Answer Dataset
- GCP - Google Cloud Platform
- DB – Database
- URL - Uniform Resource Locator