Corpus Q&A Tool

COL106: Data Structures and Algorithms, Semester-I 2023–2024

Honor Code

Following is the Honor Code for this assignment.

- Copying programming code in whole or in part from another or allowing your own code (in whole or in part) to be used by another in an assignment meant to be done individually is a serious offence.

- The use of publicly available codes on the internet is strictly not allowed for this assignment.

- The use of ChatGPT and other AI tools is strictly forbidden for writing code for this assignment. If any student has been found guilty, it will result into disciplinary action.

- Collaboration with any other team would be construed as academic misconduct.

- Outsourcing the assignment to another person or "service" is a very serious academic offence and could result in disciplinary action.

- Sharing of passwords and login ids is explicitly disallowed in this Institute and any instance of it is academic misconduct. Such sharing only compromises your own privacy and the security of our Dept./Institute network. • Please ensure that your assignment directories and files are well-protected against copying by others. Deliberate or inadvertent copying of code will result in penalty for all parties who have "highly similar" code. Note that all the files of an assignment will be screened manually or by a program for similarity before being evaluated. In case such similarities are found, the origin as well as destination of the code will be held equally responsible (as the software cannot distinguish between the two).

Introduction

Suppose you have a large corpus consisting of several books and documents. This could be a corpus of legal documents, novels, textbooks, research papers, etc. For this assignment, we have provided you with two different corpuses.

The first corpus is *The Collected Works of Mahatma Gandhi* which contains 98 books attempting to cover everything he said or wrote, translated to English. The second corpus contains books by Mahrishi Ramana (30 December 1879 – 14 April 1950) – a recent spiritual powerhouse who taught the method of *self-enquiry* to find out the answers to hard questions of life and this world. According to him "Our real nature is Liberation, but we imagine that we are bound and we make strenuous efforts to get free, although all the while we are free. This is understood only when we reach that state. Then we shall be surprised to find that we were frantically striving to attain that we already were and are". His writings contain several deep spiritual, psychological and practical insights that have helped many people from all walks of life and from all over the world. While we were unable to get access to the complete works of Mahrishi Ramana, the current president of Ramanashram has contributed two books for this project. For part 2 of the assignment ("Quering the LLM") you must choose one of the two corpuses and fine tune your algorithm and prompts for getting best possible results. This part will be evaluated subjectively during viva where the quality of your results will play a significant part in your evaluations.

Suppose we would like to know what were the Mahatma's views on the Partition of India. Or we would like to find out what was his state of mind in months around Independence of India, we should be able to answer such

questions using the first corpus. We would expect that the answer to such questions somewhere lies within this huge corpus of 98 books. It may be distributed throughout several paragraphs across the different books of the corpus. To learn the answer, we would have to painstakingly go through thousands of paragraphs throughout the corpus, which seems hopelessly inefficient.

However, we now have powerful Large Language Models (LLMs) such as ChatGPT at our disposal, and this seems like something they would be good at. Unfortunately, one can only provide these LLMs with a limited *context*, i.e. amount of information on which they can provide a response. Here is where you will get to apply your data structures and algorithm skills. Before we begin, here are some notes.

Getting Started

Here are a few things to get you started:

1. The StarterCode for this assignment can be found here.

2. This assignment is to be done in teams of 4, Please merge 2 teams from A6 to form new teams. No reshuffling of teams will be allowed under any circumstances. (The only exception is when your partner has withdrawn the course - in which case, you should first inform the Instructors/TAs).

1 Search Results Ranking (Finding Top-k Paragraphs) (50 Marks)

Since we can only feed a limited amount of context to an LLM, our first task should be to try and identify which paragraphs in the text are "important" for any given query. Here, we will describe a simple algorithm for identifying the top-k paragraphs for an input query: • Search for the words of the input query in the corpus, using the search mechanism you developed in A6. Using this, you will obtain a list of matches.

- By iterating over the matches, you need to increment the *score* of each paragraph which contains the word. We will describe how this score is calculated shortly.

- Sort the paragraphs in which the matches are found according to these scores.

- Return the top-k paragraphs which you find.

Upon spending some time on it, one will realize that this is a fairly simple algorithm. What remains to be discussed is how to calculate the score of each paragraph.

Note: In the above algorithm, we are searching for each word now, instead of the "exact" input query (as in A6). Thus, you will need to pre-process the input query, before you can use the same search function, that you implemented in A6.

1.1 Scoring a Word

We wish to score a word based on its importance to the given corpus, adjusted for the fact that some words appear more frequently in general. Given a word *w*, we calculate its score as follows:

$$s(w) = \frac{\text{frequency(w) in specific corpus} + 1}{\text{in a general corpus} + 1} \text{frequency(w)}$$

Thus, words which have a higher score are more important in the context of the corpus, as compared to words which have a smaller score. We will provide you with a CSV file containing the frequency of words occurring in a typical setting. You can count the frequency of words in the corpus using the dictionary you built in Assignment 6.

1.2   Scoring a Paragraph

Suppose a paragraph p contains matches for the query words $w_1, w_2, \cdots w_k$, having scores $s(w)$, and let the number of occurrences in the paragraph of each word in paragraph p be $f_p(w)$. Then, the score of a paragraph is given by,

$$\text{score(p)} = \sum_{i=1}^{k} f_p(w_i) s(w_i)$$

Therefore, the score of a paragraph is simply the weighted sum of the score of all the query words present in it.

Your submission will be graded based on the following parameters:

Correctness (35 Marks)

The correctness of your implementation will be checked by making get top-k paragraphs for several input queries. The marks for this section will be determined by the fraction of evaluation test cases which your implementation passes.

Please note that for this part, you **must** follow the algorithm we have described above. In Part II, if you wish to use a different algorithm for ranking paragraphs, you should create a separate function. Note that for Part I, the queries will only be from the MK Gandhi corpus.

Competitive Part (15 Marks)

Since the task is fairly straightforward, if your implementation passes all the evaluation tests, it will become eligible for the competitive part. In this part, we will rank your submissions by the runtime on a set of testcases, which will include a large number of input queries, and marks will be given both by comparing your runtime to a baseline implementation, and according to the ranking of your submission in the leaderboard.

2   Querying the LLM (50 Marks)

In the first part, we demonstrate a *possible* algorithm for identifying important paragraphs to feed to ChatGPT or another open sourse LLM. However, there are still some important questions for you to think about:

- If we are to simply use the algorithm described above, what is a good value of *k*? Is more always better?

- The quality of answers we get depends on the context we feed to the LLM. Does this method really give us the *most relevant* information for a given input query? Can we do better and come up with a way to feed more refined information to the language model, so that we receive better responses? • Is this algorithm optimal? Can we come up with an algorithm using other data structures and algorithms, which is faster and yet produces a similar quality of responses from LLMs?

- Prompt engineering is an important method to get high quality relevant results from LLMs. Can we engineer the prompt in such a way that we receive a better quality of responses from ChatGPT or other open source LLMs?

- Is ChatGPT really the holy grail for this task? Can we try using other free, open-source LLMs? Does it improve on the quality of responses we receive? Can we perhaps use multiple LLMs to get the best of both worlds?

In this part, you have ultimate creative freedom - you are free to choose any algorithm and any data structures of your choice to try and get the most relevant information to feed to the large language model you are using for

this task. You are even free to play around with the LLM you use for this task, as long as you feel it optimizes your query.

An example of an optimization which you can perform is as follows: Instead of spending time ranking paragraphs according to common words such as "a", "the", "of", etc. you can try creating a list of such words to ignore while ranking the paragraphs.

You will be given a function query_llm which will take your ChatGPT API Key and the context, consisting of the query and the relevant information, as an input. It will call a Python script which makes the ChatGPT API call using the context which you supply. You will be graded on the basis of the response to the query provided by ChatGPT. You are not required to use this function, but we are providing it to hide the details of the API call, in case you are unfamiliar with Python or with using APIs.

Viva (50 Marks)

For this part, you will be graded on the basis of a viva which will be conducted by the course instructors. You will be required to download your submission and make it run in the presence of instructors. You will be given several queries, and you will be required to demonstrate the output of your LLM query during the limited viva time. You will also be asked to explain the algorithm and any prompt engineering which you have used to prepare the query to the LLM (or any other experimentations that you may have done for best quality results). You will be graded on the quality of the responses to the input query, the time taken to prepare the query to the LLM, as well as on your algorithm and understanding. Specifically, take care to ensure that (a) the responses are meaningful and pertinent (b) responses only use the corpus supplied to come up with the answers (c) show suitable references and quotations from the supplied corpus to support the answers.

Bonus (15 Marks)

Free does not always mean worse! If possible, we would like you to explore using other free, open-source large language models available online, in lieu of ChatGPT. For example:

- You may use the unofficial HuggingChat API which is available on Github

- Besides this, there are several available on HuggingFace which can be accessed via their free inference API.

Of course, you will have to figure out the interface of these models. There may also be a difference in the quality of outputs, so you may have to work a bit harder on choosing precisely what information to supply to these models, and on some other related aspects as well. In order to compensate your efforts, if your final submission uses a free and open-source LLM API, and its quality is comparable to (or better than) that of ChatGPT, you will be eligible for these bonus marks.

Things to Note

Please note the following regarding the second part of the assignment very carefully:

- For ChatGPT we will only be using the gpt-3.5-turbo model, which costs about $0.0015 / 1K tokens for a 4K context. You should have $5-$18 credits per member of your team post renewal, which should be sufficient for the completion of this assignment and testing, but please plan accordingly.

- You will **not** be allowed to use any other paid models. Further you should **not** pay for more ChatGPT credits apart from the free credits you already have.

- If you would like to do a lot of experimenting, you may try using other free open source APIs initially if you are worried about running out of credits.

- Please reach out to us in case your team's credits run out, we will try to provide you with some ideas and alternatives.

3 Implementation Details

You have been given a header file qna_tool.h containing the declaration of the class QNA tool. You must implement the following functions in QNA_tool.cpp:

- QNA_tool(): Create an instance of QNA_tool. In the constructor, you may initialize any data structures that you may need.

- ~ QNA_tool(): If need be, you may use the destructor to do the finishing up, and delete any memory that you still hold.

- Node* get top k para(string question, int k): Given a query as a string, preprocess it and return the head of a linked list of size *k* containing the top *k* paragraphs, scored by the query as mentioned in Section 1. Each Node in the linked list must contain the book code, page number and paragraph. There is no need to set the other parameters.

- void query(string question, string filename): This is the crux of A7. In this function, you will be given a question (as string), and a filename where you should be writing the answer to the question. Make sure you print the final query or queries you are sending to the LLM to stdout. You may also print some other details regarding the query which you may want to demonstrate in the viva. No changes will be allowed later

Note that we have already implemented the function query_llm for you. This function will take the Linked List containing top-k paragraphs, the integer k, your ChatGPT API KEY, and the python file which interacts with the ChatGPT API. You are free to use/not use this function, or make any changes to it. You can show your prompt engineering skills in this function too!

We are also giving you the file api_call.py. This file uses the ChatGPT API via its interface, sends the query message, and prints the response to the console. Again, you may choose to use/not use this file. In case you are using any other LLM, you will need to modify this file to match its interface.

The Node.h and Node.cpp file provided to you are same as before. Use the dict.h, dict.cpp, search.h, search.cpp files that you have implemented in A6.

4 Submission Instructions

You are required to submit the all your source files on Gradescope. You are free to create and use other files, but these should include at least the following: qna_tool.cpp, qna_tool.h, dict.cpp, dict.h, search.cpp, search.h.

Further, you are required to submit a typed report, report.pdf, which contains:

- Your algorithm to identify the text to feed to the LLM

- Other optimizations performed and prompt engineering techniques used by you