Plaksha Chatbot: A private chatbot to answer all plaksha-related queries

Aayush Gupta Gaurav Singh Kirubananth Sankar Shashank Prasad

1. Introduction

Plaksha University is a relatively new establishment. All the necessary information regarding plaksha, from the admission processes to class schedules for the students inside the college, is scattered throughout many websites and is not easily accessible. There is a clear need for a portal where all the stakeholders can access their necessary information. To achieve this and to make the experience more user-friendly, we have decided to build a chatbot that can answer all these questions.

Apart from user-friendliness, another critical aspect to consider is privacy. Because of this, we cannot use API calls to third-party models and need to deploy a model locally.

Finally, we have limited this project's scope to creating a chatbot catering to the students and parents looking to get admission into plaksha and need information regarding the process.

2. Data

The first step to creating the chatbot is to create the knowledge base. Ideally, we should set up a pipeline where the information can be added, and the knowledge base can be updated in real-time. But for the sake of this project, we went with a static knowledge base. The current knowledge base is a local folder with PDFs which were obtained by scraping multiple plaksha website links that had information ranging from admission processes to campus life and activities.

3. System Design

The main components of our system are:

1. **Chunking Layer:**
   The first layer of the system is the chunking layer. This breaks down the source documents into text chunks of predetermined size.

2. **Embedding Layer:**
   Next is the embedding layer. This summarizes each of the text chunks created above into a vector.
3. **Vector Store:**
   The summarized vectors are then stored in this vector store. These vectors are later accessed during similarity search.

4. **Transformer Model:**
   The final part of the system is the Large Language Model which takes in the prompt generated and gives out the output to the user.

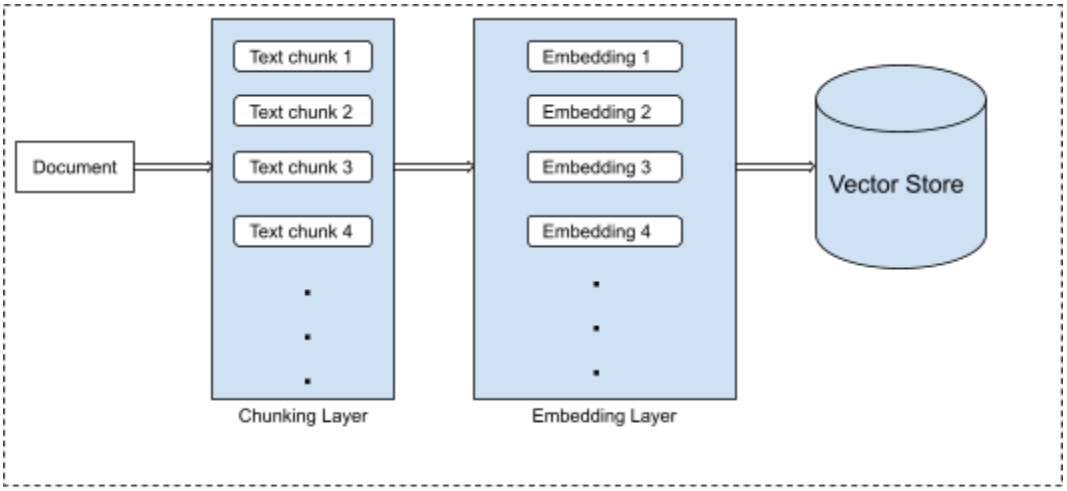

Fig.1: Preprocessing

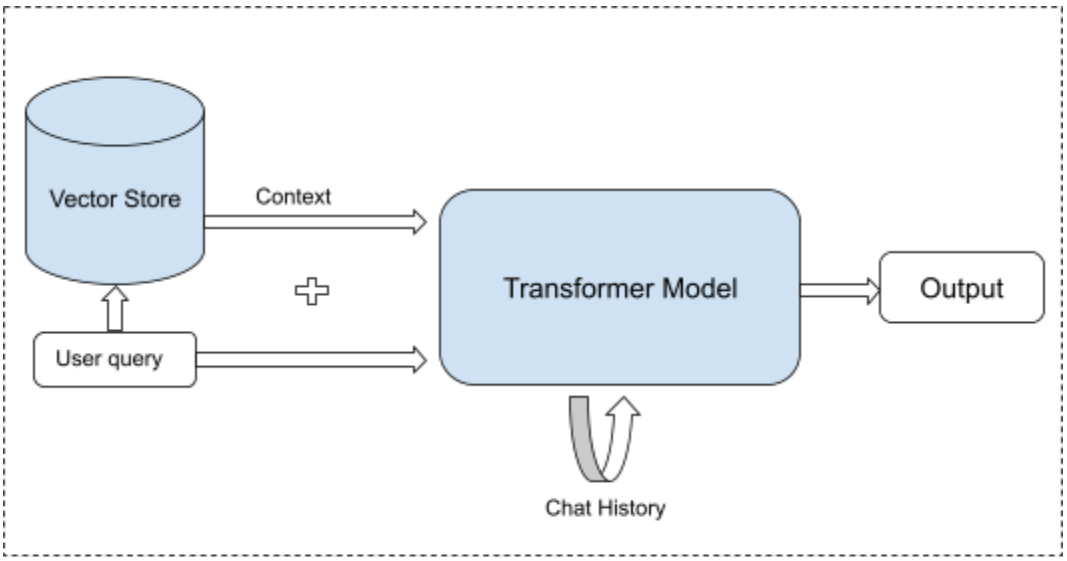

Fig.2: Chatbot Workflow

4. Solution Overview

4.1 Preprocessing:

The first step in our preprocessing was to break down the source documents into smaller text chunks which will later act as the context to our model. We did a fixed chunking technique where each page in the document is broken down to 8 chunks. The next step is to convert these text chunks into vectors. We used the 'sentence transformer embeddings' from hugging face as our embedding layer. This converted each of the text chunks into vectors of each size 768. Next step is to store all of these vectors in a vector store. We needed a vector which is lighter and can be used locally in a laptop. So we chose the 'Chroma db' as our vector store.

4.2 Chatbot workflow:

When the user query comes to the chatbot, it is first converted to a vector using the same embedding from the preprocessing step. Using this vector a similarity search is done on the vector database, which is finding the cosine similarity between the query vector and all the vectors present in the database. Top four vectors are selected from the database and their corresponding text chunks are combined to form the context. Finally the user query is combined with the context and the history of the chat using a prompt template function and the resulting prompt is passed in as the input to the transformer model which will act as the chatbot and produce the desired output.

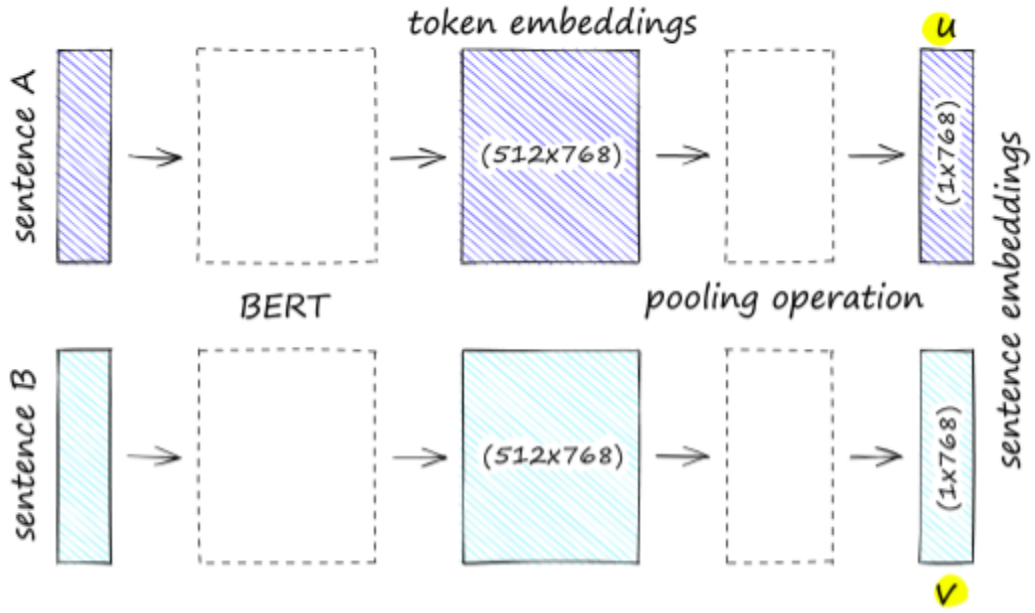

Fig.3: Similarity Search
(The cosine similarity is given by the dot product of the vector u and vector v)

For the transformer model we chose the GPT4ALL models. The reason being we needed something that can run locally on a CPU. Among these models we first experimented with lighter models and chose the best one according to the documentation which was ‘ggml-gpt4all-j-v1.3-groovy’. It was the lightest model we could find with 7B parameters. Although the latency of this model was low, it was producing low accuracy results and in many cases was hallucinating. This led us to switch to a more robust but larger model ‘nous-hermes-13b.ggmlv3.q4_0’. This has 13B parameters and a much larger latency when run on a CPU but it was giving accurate results and so we went with this model as our final choice for the project.

4.3 Results:

Results from the smaller model:

Fig.3: Sample result from the groovy model.

But as discussed above, the accuracy of the groovy model was very low and in many cases it was hallucinating.

Results from the larger model:

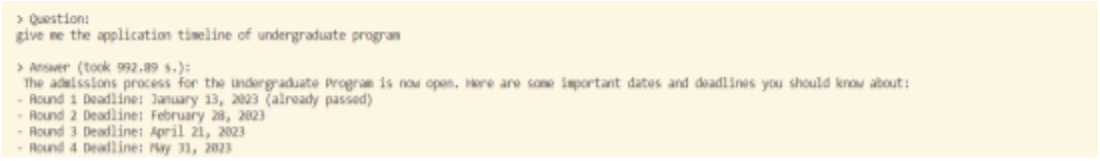

Fig.4: Sample Result from Hermes Model

5. Evaluation

We divided our team into roles of two students and two parents. Each of us then came up with ten questions and ten follow up questions. Additionally from test users we gathered ten more questions and follow up questions. Then as a team we came up with a sufficient answer and a complete answer for each of the first questions and an accepted answer for each of the follow up questions. We defined sufficient answers as those where the model has understood the context and has given an answer where we have at least a lead to our queries (we considered answers that are incomplete but have necessary information as sufficient answers). A complete answer is where the model has closely matched the answers we have come up with.

Example:
Query: *What are the requirements for admission into the U.G computer science program?*
Our answer: *Class 12 : 90%, Python.*
Sufficient answer: *Completing class 12 with minimum marks of 90%*
Complete answer:*Completing class 12 with minimum marks of 90%. It is also recommended that the candidate have a working knowledge of Python.*

Follow up Query: *What if I don't know python?*
Our answer: *Bridge course.*
Accepted answer: *There will be a bridge course for students who are not well versed in python.*
Rejected answer: *It is recommended that the candidate have a working knowledge of Python.*

The final test dataset consisted of 50 questions and 50 follow up questions and their corresponding answers.

Also the final evaluation was only done on ‘nous-hermes-13b.ggmlv3’ which for the sake of this project will be the final model.

5.1 Accuracy:

a.Single question query:
This measures the ability of the chatbot to answer a simple query. The dataset consisted of the first 50 questions and their corresponding answers. We gave a score of 0.5 for every sufficient answer that the chatbot gave and a score of 1 for every complete answer and 0 for everything else. The maximum score that the chatbot can obtain is 1 for each question.

There were 29 complete answers, 19 sufficient answers and 2 rejected answers giving a total score of 38.5 bringing the accuracy of the chatbot for a single question query to 77%.

b.Follow up question:
This measures the ability of the chatbot to remember the history and conduct a human-like conversation. The dataset consisted of the 50 questions, the corresponding 50 follow up questions and the 50 accepted answers. We gave a score of 1 for every accepted answer that the chatbot gave for the follow up question. For every other case including the failure to give a proper answer to the first question itself, we gave a score of 0.

There were 22 accepted answers bringing the accuracy to 44%. The chatbot does not perform well on follow up questions and this issue needs to be investigated further.

5.2 Latency:

Ideally our chatbot would be deployed on a local server with a GPU. But for the sake of this project it was deployed in a laptop without a GPU. This significantly impacts the latency increasing it manyfold.

We measure latency as the time difference between the user query and when the model stops responding. We took an average time taken for the entire query dataset created above. Our final model that we used : ‘nous-hermes-13b.ggmlv3’ with 13 billion parameters took on average 4 minutes per question.

6. Conclusion

After two weeks of working on the project we were able to successfully create a chatbot that can answer the questions related to Plaksha University with passable accuracy.

6.1 Key Areas of Improvement:

1. **Dataset:**
   The current dataset we used was the openly available data from the Plaksha Website, including details about the admissions for TLP, B. Tech, PhD, YTS, and other programs, as well as information about the professors, founders, and other relevant data from the Plaksha University's website. Although the model accuracy was acceptable, when we tested our chatbot by changing the dataset to articles collected form wikipedia, the chatbot performed significantly better leading us to believe that there is a limitation in the data we were using. One reason for this can be that in the wikipedia articles it is easier for the model to

understand the context much easier because the data is clean. In essence we need to focus on cleaning up the dataset which will lead to a better result. The second reason could be the limited size of the dataset. For our evaluation of the chatbot the input dataset only consisted of 3 PDFs with information directly scrapped from the websites. By taking the help of the program team and increasing the size and the quality of information in this dataset, we hypothesize that performance of the chatbot can be made better.

2. **Poor accuracy with follow up questions:**
   As we saw above in the evaluation, the chatbot does not perform well when it comes to follow up questions. One reason could be the poor quality of the dataset which will be improved in the future iterations. This issue needs to be investigated further.

3. **Latency:**
   The average latency per question is around 4 minutes. The main reason for such heavy latency is the fact that our model was running on the CPU of a laptop. We hypothesize that when it is deployed in a local server with a GPU this latency will be significantly reduced. We could also experiment with more lighter models. But overall in theory this should be the fastest method because that alternative of using API calls would always take longer than the local model.

4. **Chunking:**
   When observing the top 4 sources where the model is looking at by similarity search for incomplete and rejected answers, we noticed that these source text chunks, although they were complete sentences, were broken off in the middle of a topic thereby losing the context. We tried increasing the window length for the text chunks but this just resulted in the loss of data as now the bigger chunk of text was summarized to the same length vector. We could try to play around with the size of the embedding and the size and number of the text chunks but the core issue would remain the same. We need to investigate this further and adopt a better chunking technique.

6.2 Future Work:

1. **User Interface:**
   The next obvious step in the project would be to build a user-friendly interface with the Plaksha theme. We are planning to make use of Streamlit to make this happen.

2. **Using Open AI API with added security layer:**

The main reason for deploying a local model was privacy. But this impacts the model accuracy compared to the SOTA models. One alternative which we plan to explore is using a local model to mask the personal information, sending the masked context and query to the SOTA models through API calls, then using the local model once again to append the answers with the needed information before answering the users. This would solve the problem of accuracy and privacy. But it may end up increasing the latency of the system.

3. **Dataset pipeline:**
   We also need to build a pipeline to update the knowledge base of the chatbot in real time. This includes building a web scraping and cleaning pipeline from the Plaksha websites. Also a portal where the program team adds additional information in the form of pre set forms or by dragging and dropping PDFs to the source documents storage. In addition to this, we should also set up a timed update of the vector database.

7. Acknowledgments

We're sincerely grateful to Professor Monojit Choudhury and Kabir Ahuja at Plaksha for giving us the opportunity and guidance to build this project in his class.

8. Citations

[1] LangChain. (2023, June 24). *Tutorial: CHAT GPT over your data*. LangChain. https://blog.langchain.dev/tutorial-chatgpt-over-your-data/

[2] Schlesinger, D. (2023, July 7). *Build a chatbot to query your documentation using Langchain and Azure Openai*. TECHCOMMUNITY.MICROSOFT.COM. https://techcommunity.microsoft.com/t5/startups-at-microsoft/build-a-chatbot-to-query-your-documentation-using-langchain-and/ba-p/3833134#:~:text=Its%20primary%20goal%20is%20to,%2C%20private%20repositories%2C%20and%20more.