Group Project - Individual Report
AI News Reader

Group-05
Arjun Bingly

DATS-6312
Natural Language Processing
Dr. Amir Jafari

December 11, 2023

Abstract

Our Smart News Reader employs advanced Natural Language Processing (NLP) to transform news consumption. With features like news parsing, summarization, zero-shot classification, keyword extraction, and conversational question-answering, it streamlines information retrieval. Despite encountering challenges in implementing the translation component, the application represents a leap in redefining personalized and dynamic news exploration. Through continuous integration of NLP techniques, it addresses information overload, catering to evolving user needs in the digital age.

Contents

1 Introduction

1.1 Features

- News Parser
- Summarization
- Key word Extraction
- Zero-Shot Classification
- Conversational Question Answering Chat-Bot

2 Contribution

2.1 News Parsing

For obtaining the news article body given a URL, the first attempt was to use BeautifulSoup to parse the HTML and extract the text body, but this proved to be difficult since each news website uses different structuring. This would restrict our app to a handful of websites. Upon looking further into this, we used the Newspaper3k python package which parses the News article given the URL. This extensive python package supports a lot of news websites and has rich features which includes extracting information like images, etc.

2.2 Summarization

The base summarizer was implemented by my teammates using BART-Large-CNN model, this is a BART Large Model fine-tuned on the CNN News dataset but the main limitation of this set-up is the token-limit. News Articles can be fairly large the max token limit on BART is 1024, with the initial set-up the part of the articles that do not fit into this limit is simply truncated.
To overcome this limitation, I used LangChain's RecursiveTextSplitter to split the article into chunks of less than the token-limit. These chunks are passed to our summarizer model sequentially and each summary generated is concatenated to form our final summary. The limitation with this approach is that we now loose direct control of the final size of the summary, but this can be addressed by recursive summarization, i.e. pass the long summary again to the splitter and then summarizer until the summary is of desired length.

2.3 Conversational Q-A

My team-members had initially implemented the BERT Large model fine-tuned on the SQAUD dataset. But this implementation has the similar token limit as explained above. Moreover, this model is not conversational, it is a comprehensional answering model, i.e. it just picks the answer from the context.

But to overcome the token-limit I first implemented Retrieval Augmented QnA using LangChain. This works by first splitting the text into small chunks and then embedding them using an embedding model and then storing these embedded vectors in a vector database. Given a question, a similarity search is done on the database and the top-k chunks are given to the QnA model as context along with the question. In our particular implementation we used (Chroma vector database since we wanted all the services to be run locally. We also noticed that the embedding model significantly affected the performance of our model. The best model we found was 'jinaai/jina-embeddings-v2-base-en' but this model requires running code outside HuggingFace and this requires a user input of 'yes' this can be alleviated by giving an argument in HuggingFace but the LangChain wrapper for HuggingFace did not support this argument. So we settled on the second best embedding model we found, 'sentence-transformers/all-MiniLM-L12-v2'. This is a sentence transformer that uses a Siamese-BERT model.

Now, to introduce conversatinoality to the model we had to move to larger LLM models. Since we wanted to run all our models locally, I opted to use the Llama-2 model. Upon submitting an online request to META, we were able to download the weights for all the Llama-2 models. Firstly to run these models on our computer we had to use the llama.cpp package to convert the weights so as to run the model in C/C++ (for faster inference). We then tried using the small (7B) and medium (13B) chat models but then we immediately realized that we were not even able to load the 13B model to our A10 GPU while we could load the 7B model to the GPU inferencing was very slow. This meant that we had to quantize our model we tried to use the same llama.cpp package to quantize the weights to a 4bit integer, but I ran into dependency issues while compiling llama.cpp. So we downloaded an already quantized model of the llama2 on HuggingFace, TheBloke/Llama-2-13B-chat-GGUF. We also infereced on the GPU using CuBLAS, it is a CUDA library for Basic Linear Algebra Subroutine.
I then used LangChain to interface with the model using llama.cpp (refer here). LangChain's ConversationalRetrivalQAChain was used, this chain works simmilar to the previously mentioned RAG model but does prompting in two parts: if the question is stand-alone, say the first question this just uses a prompt of

```
Answer the question based only on the following context:{context}

Question: {question}
```

whereas if the question is a follow up question the following prompt is used to rephrase the question into a stand alone question and the first prompt is used,

```
Given the following conversation and a follow up question, rephrase the
    follow up question to be a standalone question, in its original
    language.

Chat History: {chat_history}
Follow Up Input: {question}
Standalone question:
```

Moreover, for keeping the memory of the previous was implemented using LangChain's Memory Buffer. Since this LLM has a lot of general knowledge this is very useful in keeping a conversation and the prompting tries to ensure that the LLM sticks to the context and does not hallucinate.

2.4 The App

Since all the coding was done in object-oriented method, implementation of the app using StreamLit relatively easy but the challenges included:

- **GPU Memory**: due to the limited GPU memory we had to resort to only running the summarizer and chat-bot on the GPU leaving everything else on the CPU. This is clearly not a production ready set-up due to high cost of GPU machines.
- **Whole App Refresh**: Due to the way StreamLit works the entire app re-runs top to bottom upon any change in the inputs, this is very costly for our app. Hence, a multi-page approach was taken since in a multi-page set up only the active page re-runs. The chat-bot, which is the most computationally expensive module in our app was moved to a separate page
- **Whole App Refresh**: Due to the way StreamLit works the entire app re-runs top to bottom upon any change in the inputs, this is very costly for our app. Hence, a multi-page approach was taken since in a multi-page set up only the active page re-runs. The chat-bot, which is the most computationally expensive module in our app was moved to a separate page
- **Caching**: Since the app reloads every time, re-loading all the models is very wasteful, hence the caching functionality offered by StreamLit was utilized.

- **Session States**: Session states are the only elements that can be carried over across re-runs, hence they were effectively used in the effort to preserve user inputs and also to pass user inputs and other elements across pages.

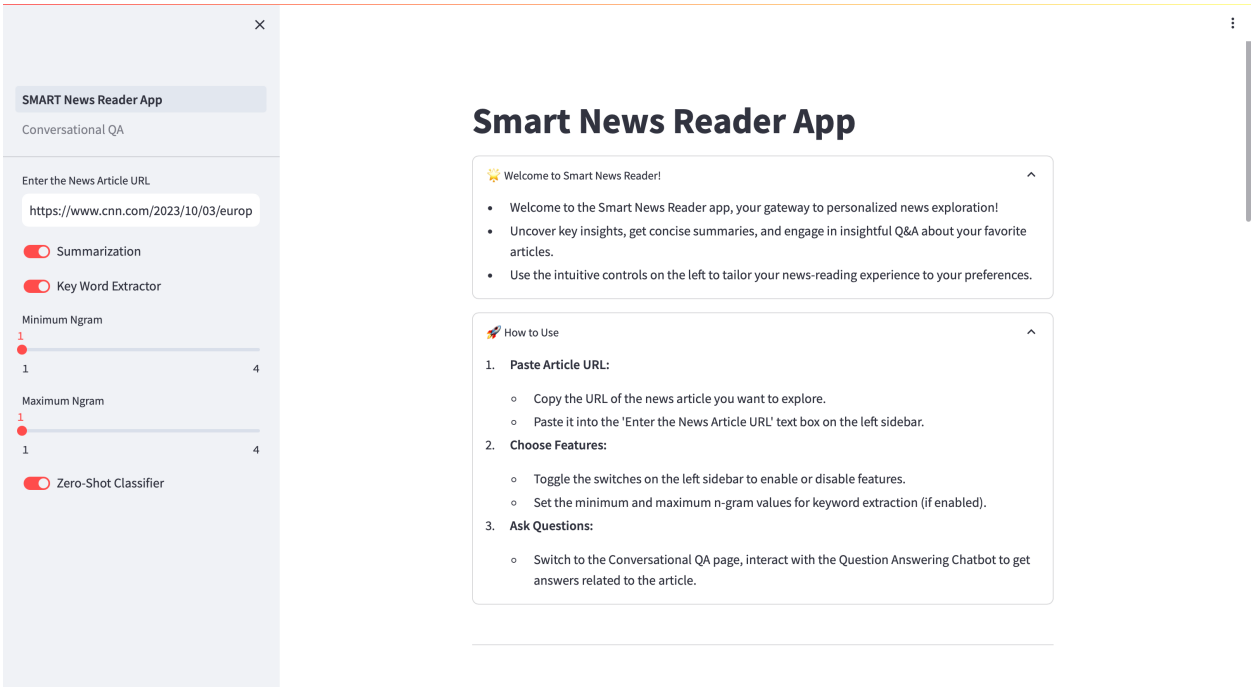

Figure 1: Prediction Plot using the Regression Model

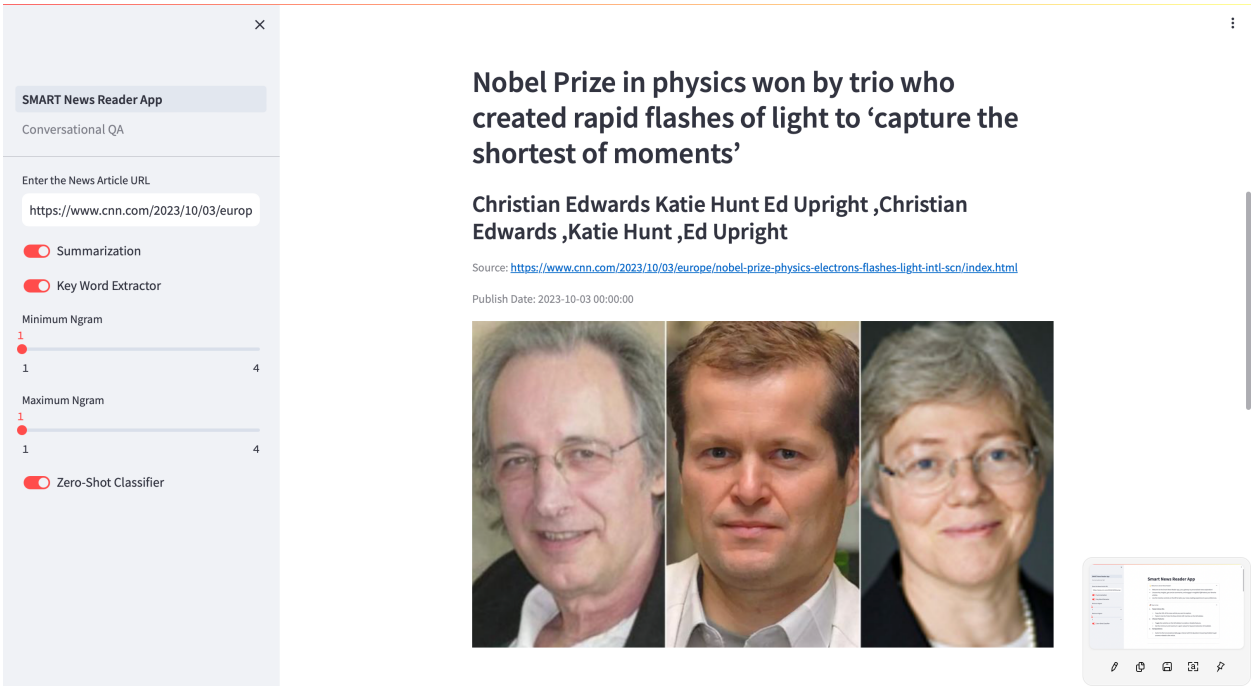

Figure 2: Prediction Plot using the Regression Model

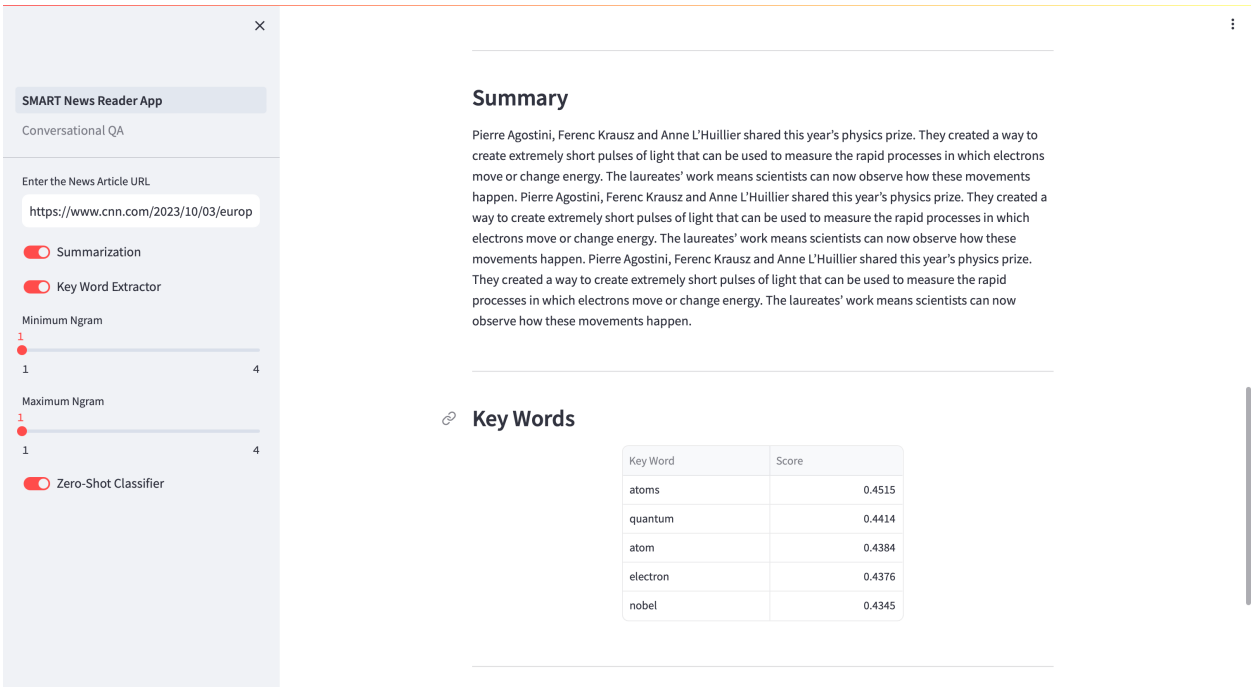

Figure 3: Prediction Plot using the Regression Model

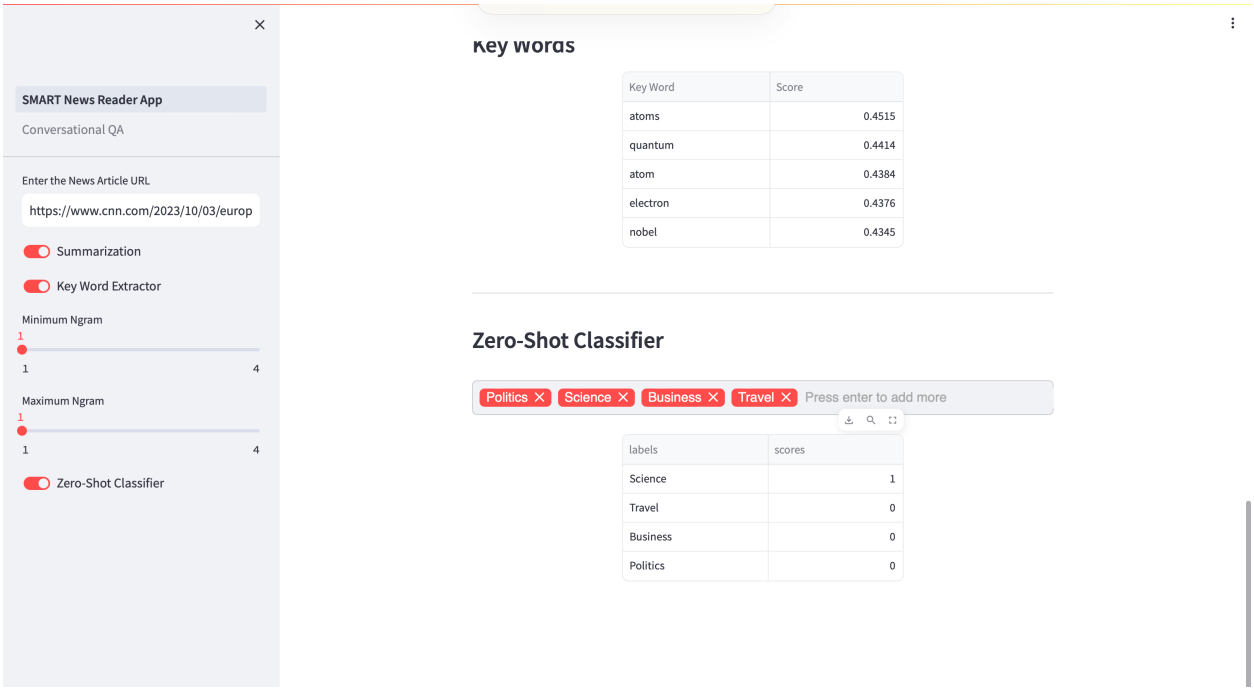

Figure 4: Prediction Plot using the Regression Model

Figure 1 shows the about and help section of the main app page while *Figure 2* shows parsed news article with the news title, authors, original url, publishing date and also an image from the original article. *Figure 3* shows the generated summary, the original news article had about 1200 words while the generated summary has about 150 words; the figure also shows extracted key-words which are clearly aligned to the article. *Figure 4* shows the zero-shot classification and it rightly classifies the news article. *Figure 5* shows

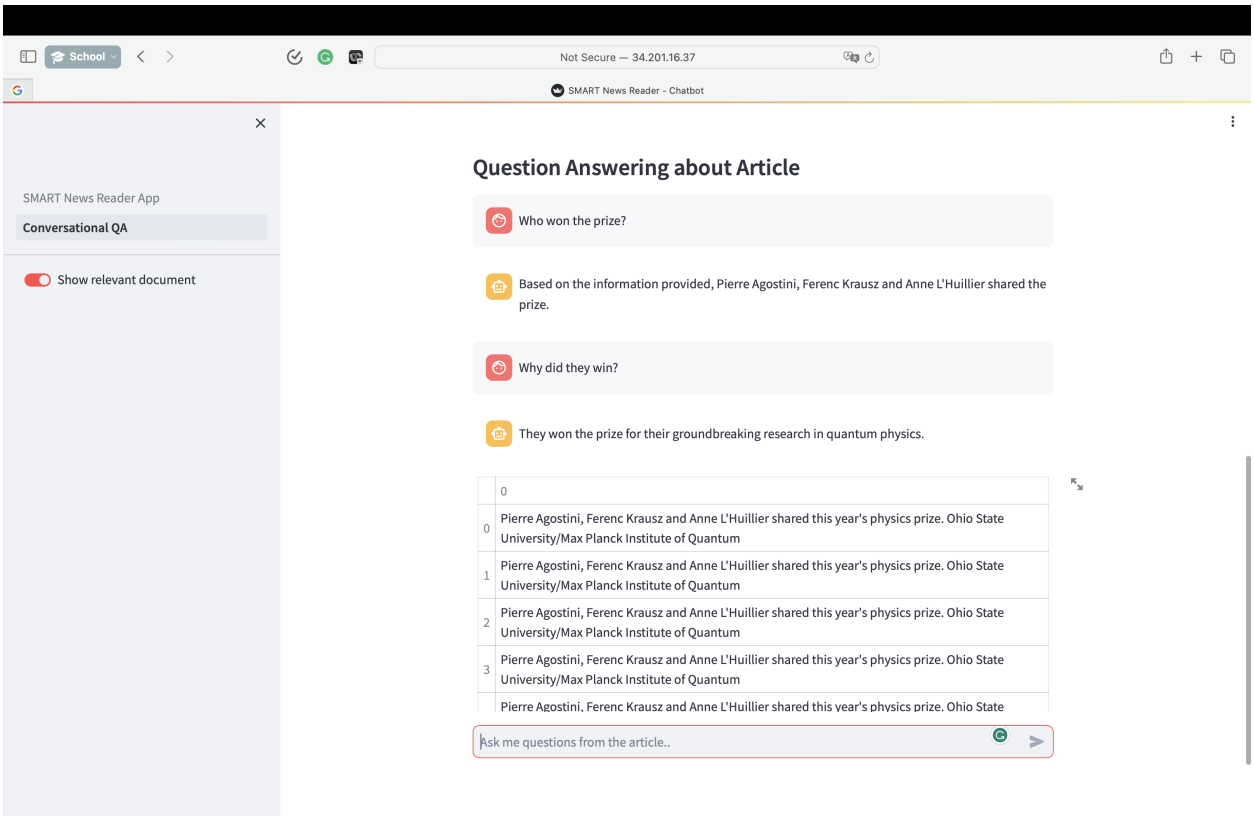

Figure 5: Prediction Plot using the Regression Model

the conversational chat-bot page, you can see that the questions were well answered and the relevant parts from the article used by the chat model displayed.

3 Possible Improvements

- The whole app takes a lot of computational resource and takes a fairly long time to run, optimizations of both computational resource and time could be done.
- The app uses multiple models, this could be condensed to a single LLM.
- Every input refreshes the whole app, even with caching more improvements needs to be made.
- Better UI.
- Implementation of the translation feature.
- App currently only supports one user.

Appox. code copied from Internet : 37%