NetDesign
Jair Ruiz, Evan Perez, Usman Abbas
CSC 59866 Senior Design 1

NetDesign
Senior Design Project: Text Summarization and RAG playground

Introduction

Being able to automate the process of summarizing pieces of text has become a prominent issue for companies over the past decade. This is because the task of extracting specific information from documents has always been a manual task within corporate and academic settings alike. One of the most popular techniques to solve this problem is the use of Retrieval Augmented Generation (RAG) which involves breaking down text into smaller chunks, retrieving the chunks from a knowledge database, and then using an LLM to elaborate or summarize the original text. Less popular but noteworthy are machine learning models designed for specific tasks like summarization or question answering. These end-to-end systems are trained directly on the task without needing to retrieve information from a separate database. They offer advantages such as faster performance, reduced computational overhead, and the ability to be fine-tuned on specialized datasets for improved accuracy and relevance.

Goal and objectives

The goal of our project is to build and train a seq2seq2 machine learning model that performs text summarization, and develop an interface to create RAG systems to compare both approaches and gain insights of how well they perform depending on different scenarios. The first objective is to build a text summarizer that takes an input of a long sequence of words from a text body and outputs a short summary, by using a sequence-to-sequence model. This requires an Encode-Decoder architecture since the input text and output text may be of different lengths and we can achieve this by using a Long Short Term Memory (LTSM) model. The next objective is to develop a seamless no-code interface to create RAG pipelines where we will be able to upload documents that will be used as the knowledge database for the system. The last objective is to create visualizations for RAG pipeline performance, our seq2seq performance, and comparing both results to gain insights of the systems performances.

Abstract summarization using Seq2Seq2 model

Jin et al. (2024) define Automatic Text Summarization (ATS) as the process of condensing lengthy content into concise summaries, either through extractive or abstractive methods. Extractive ATS selects sentences from the original text, while abstractive ATS generates new sentences to capture the original meaning. We chose abstractive ATS due to its ability to produce more coherent and naturally phrased summaries. Abstractive methods include RNN-based models, such as LSTM and GRU, and transformer-based models like BERT and GPT. Chopra et al. (2016), showed that RNN-based models like RAS-Elman and RAS-LSTM achieved high performance in summarization tasks, the project decided to use a seq2seq LSTM-based model for its summarization goals.

Pai (2023) shows us that in order to build our sequence-to-sequence model, we will need to follow a preprocessing procedure where we read a text dataset, drop duplicate and null values, and clean the data by removing characters and text such as stop words and punctuation. After doing so, we need to prepare text tokenizers and build the model by implementing a multilayered LSTM within Python using the Keras library. Once this is done, we will need to train the model on our text summarization dataset and evaluate its performance with diagnostic plots in order to understand the behavior of our model over time and decide how to make adjustments. There are multiple ways to improve the model's performance such as implementing Bi-directional LSTM for better context vectors or implementing a beam search strategy for decoding our test sequence instead of using a greedy approach. It is difficult for the encoder to memorize long sequences into a fixed-length vector, so we can implement an attention mechanism, a concept that aims to predict a word by looking at parts of a sequence of text rather than an entire sequence. This all can be achieved using the Python programming language and the Keras library, a programming add-on that allows us to build deep learning models. We plan to train this model within Google Colab and then upload the trained model to a streamlit demo website along with our RAG interface.

After training our summarization model, we can compare its performance with LLM-based RAG pipelines. This technique is known for its powerful summarization capabilities by using the backbones of large language models such as ChatGPT or Google Gemini, so it will be interesting to be able to run our text summarizer against the RAG pipelines. To achieve this, we can develop higher-order functions that take helper functions as input in order to complete the overall process of RAG. This process consists of several steps including turning text into chunks, converting text chunks into embeddings, and then storing the embeddings in a vector database. Then, the LLM of choice will use retrieved data from the vector database to answer a question such as "Summarize the second paragraph of the context" and then generate the appropriate response using the generative features of the LLM in conjunction with the retrieved data.

The comparison will be insightful due to recent research on combining LLMs with RAG for text summarization. Liu et al. (2024) explore the robustness of LLMs in RAG-based summarization, an under-researched area despite LLMs' known capabilities. They introduce LogicSumm, an innovative framework to evaluate LLM robustness in realistic scenarios. Building on LogicSumm's findings, they develop SummRAG, a system designed to enhance model robustness through training dialogues and fine-tuning. SummRAG systematically tests LLM capabilities, resulting in improved logical coherence and summarization quality as demonstrated by experimental results.

Evaluation metrics

To evaluate the text summarization performance, the project will use ROUGE-N (ROUGE-1 and ROUGE-2) and ROUGE-L metrics. ROUGE, showed by Jin et al. (2024), is widely used for assessing Automatic Text Summarization (ATS) systems by measuring the overlap between generated and reference summaries using n-grams, longest common subsequences, word positions, and skip-bigram co-occurrence. ROUGE-N focuses on n-gram overlap, while ROUGE-L measures the longest common subsequence, assessing precision, recall, and F-score. Although ROUGE evaluates recall, it does not directly address fluency and conciseness. The formula for ROUGE-N is the following:

$$ROUGE\ N = \frac{\sum_{S} \sum_{gram_n \epsilon S} Count_{match}(gram_n)}{\sum_{S} \sum_{gram_n \epsilon S} Count(gram_n)}$$

The project will utilize Google Colab for training and evaluation due to its ease of use, powerful cloud GPU access, ample storage, and adaptability for prototyping. Post-training, the project will migrate to a local environment using Streamlit to build the application interface and front-end for model inference. Streamlit offers rapid development iterations, session state management, community plugins, effective display tools, and simple deployment to Streamlit Cloud Community.

Information collected

Collected ideas

We have intensively collected ideas for our application. Even though we did not find projects related with seq2seq summarizers and an interface, we found projects that are related to the RAG system part of our project.

- RAGArch: It is a Streamlit app for testing and comparing RAG pipelines with customizable parameters. It comes with real-time configuration of LLMs, embeddings, and vector stores. Key features as:
  - Export Python code for configured pipelines.
  - Supports various LLMs (Gemini Pro, Cohere, GPT) and embeddings.
  - Vector Stores: Simple, Pinecone, and Qdrant.
- **RAGs**: It is a Streamlit app for creating RAG pipelines using natural language. It has key features as:
  - Home Page to build RAG pipelines by describing tasks and datasets.
  - RAG Config file to customize parameters like top-k, summarization, and embedding models.
  - Generated RAG Agent to query the RAG agent with questions over the specified data source.

Collected datasets

After doing our research in different surveys about ATS, we collected three datasets that are good candidates for training and evaluation purposes.

- **WikiHow Dataset:** It is a large-scale summarization dataset from WikiHow knowledge base with a size of over 200K pairs of articles and summaries, comprising online articles outlining procedural tasks spanning diverse topics such as arts, entertainment, computers, and electronics.

- **CNN/Daily Mail Dataset**: English-language dataset with news articles from CNN and Daily Mail with a size over 300K unique news articles, each row containing an id, article, and highlights. The highlights are used as the summaries.
- **Scientific Papers Dataset**: Contains long and structured documents from ArXiv and PubMed OpenAccess repositories. ArXiv dataset has over 200K rows, PubMed dataset has over 100K rows. Both datasets have columns: article, abstract, section_names. The abstracts are used as the summaries.

Collected techniques

We intensively collected several techniques that could help us to achieve our goals. Here is a brief summary of the collection.

- Socratic Pretraining is a question-driven, unsupervised pre training objective designed to enhance controllability in summarization tasks. It trains models to generate and answer questions relevant to the context, improving adherence to user queries and the identification of pertinent content using only unlabeled documents and a question generation system. It is particularly useful for controllable summarization of long documents in domains such as short stories and dialogue, especially when labeled data is scarce.
- This paper proposes an LSTM-CNN based ATS framework (ATSDL) that constructs new sentences by using semantic phrases rather than entire sentences. ATSDL operates in two stages: first, it extracts phrases from source sentences; second, it generates text summaries using deep learning. Experimental results on the CNN and DailyMail datasets show that ATSDL outperforms state-of-the-art models in both semantic and syntactic accuracy and achieves competitive results in manual linguistic quality evaluations.

What we know and don't know

It is important to recognize our limitants to achieve our goals. As a group, we recognized a set of topics that we cannot address right now. Even though we did our research about Seq2Seq models, we still need to figure out how to create the model in code using the LSTM architecture. Also, we need to learn how to accurately pre-process the data to train our model, since we know that NLP requires some techniques to do that. Furthermore, we need to learn how to create the workflow for the RAG playground application. Based on the project we collected it does not look an easy task to put all the pieces together in a nice interface. On the other hand, we have been exposed to RAG pipelines before, which gives us a very good understanding of the concepts around this topic. Also, we know how to train and evaluate a machine learning model, which is crucial for our objectives. Finally, we know important concepts about NLP.

Timeline

- **August-September:** During August and September, the primary focus will be on data preprocessing and defining the model architecture and overall system.

- **October:** In October, the project will move into the training phase. This involves feeding the preprocessed data into the defined model architecture and allowing the model to learn from the data.
- **November-December:** November and December will be dedicated to application development and the creation of a RAG playground. The RAG playground will serve as a testbed for exploring and demonstrating the functionalities of Retrieval Augmented Generation. Towards the end of the semester, a showcase demo will be prepared to exhibit the project's progress and results.

References

Jin, H., Zhang, Y., Meng, D., Wang, J., & Tan, J. (2024). A Comprehensive Survey on Process-Oriented Automatic Text Summarization with Exploration of LLM-Based Methods. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2403.02901

Sumit Chopra, Michael Auli, and Alexander M. Rush. 2016. Abstractive Sentence Summarization with Attentive Recurrent Neural Networks. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 93–98, San Diego, California. Association for Computational Linguistics.

Pai, A. (2023, December 26). Comprehensive Guide to Text Summarization using Deep Learning in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

Liu, S., Wu, J., Bao, J., Wang, W., Hovakimyan, N., & Healey, C. G. (2024). Towards a robust Retrieval-Based summarization system. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2403.19889