Final Project: A Song of Ice and Fire Series K-RAG

Matias Barcelo Treiyer, Anirudh Valluru
Southern Methodist University - CS 5393-004 - Advanced Python

---

Abstract
Retrieval Augmented Generation (RAG) and Knowledge Representation Augmented Generation (KRAG) was implemented with OpenAI's gpt-3.5-turbo-instruct model using George R. R. Martin's *A Song of Ice and Fire* book series, better known in American pop culture for its HBO adaptation *Game of Thrones*.

Contents

1 Introduction

Given a markdown file at the beginning of the project by Professor Coyle, team members used the example code in the markdown file to complete the project with only slight modifications. This example code, written in Python, required packages for spacy, networkx, matplotlib.pyplot, FAISS, and langchain modules. In addition to these modules, members added the tiktoken module for token control. Notably, for home use, the system developed for this report must include an OpenAI API key in the environment for use of OpenAI's gpt-3.5 model.

The team chose the *Song of Fire and Ice* series due to a specific team member's familiarity to the series and due to its ease of availability online in the ".pdf" format. Books in the series include:

1. **A Game of Thrones** (1996)
2. **A Clash of Kings** (1998)
3. **A Storm of Swords** (2000)
4. **A Feast for Crows** (2005)
5. **A Dance with Dragons** (2011)

A user must ensure these books are in .txt format in a separate directory named "sif_text" (or a directory with any other name as long as the directory variable in FinalProject.py is changed) along with the source code for the developed system to function. So, ".pdf" files must be converted prior to programming the model. Furthermore, due to bandwidth/token limitations, only two books can be selected at a time for chunking per instance of system.

2 Methodology

2.1 Data Preparation

Books were gathered in PDF format and converted into .txt files using the PyPDF2 python library. RAG and KRAG techniques retrieve files from databases and optionally make knowledge graphs based off files; however, since the OpenAI model used has a token limit, text files were split up into chunks of a thousand characters to reduce the amount of tokens in the "retrieval" process and chunks were used instead of files to build knowledge graphs.

2.2 Named Entity Recognition (NER)

For the named entity recognition portion of the project, there were issues building the model from scratch; so, in order to make progress in a timely manner, spaCy's English "en_core_web_sm" model was used in addition to the "sentencizer" pipe to split up sentences in a given sequence. For each chunk of text, the NER model split up sentences using aforementioned sentencizer pipe and tokenized each sentence for entities, subjects, grammatical objects, and root relationships returning them as a field of "sents" objects in a "Doc" object.

2.3 Relationship Extraction/Knowledge Graph Construction

The knowledge graph was constructed using the NER model's returned "Doc" object for each chunk of text. Entities were retrieved from each sentence in this "Doc" object's "sents" field and relationships between entities for each sentence in given chunks were parsed for subject, grammatical object, and root relationship between the two. Entities and relationships were passed to the networkx module to construct a knowledge graph for given chunks.

Consider the following example: "Mary reads a book. Nancy eats pickles." Treating this string as a "chunk" passed to the NER model, it recognizes the entities "Mary" and "Nancy" and stores them in the "ent" attribute in a "Doc" object. The model splits the sentences into two, stores them in its Doc's "sents" attribute, and for each sentence stores the sentence's grammatical subject, object, and root relationship between the two. In this example, "Doc.sents" is an attribute of a generator object containing the two sentences:

["Mary reads a book.", "Nancy eats pickles"]

For the first sentence, "Mary" is stored as the subject, "book" is stored as the grammatical object, and "reads" is stored as the the root relationship between the subject and object. Passing these three variables to the networkx module and visualizing the graph using matplotlib.pyplot gives the following result.

Figure 1: Knowledge Graph for Example Chunk

Noticeably the direction of the arrow goes from grammatical subject to object and the edges between nodes do not visibly have the root relationships, but they are implied. The edge from "Mary" to "book" is implied to be "reads" and the edge from "Nancy" and "pickles" is implied to be "eats".

Repeating the same process for the first fifteen chunks of *A Game of Thrones* results in the following graph.

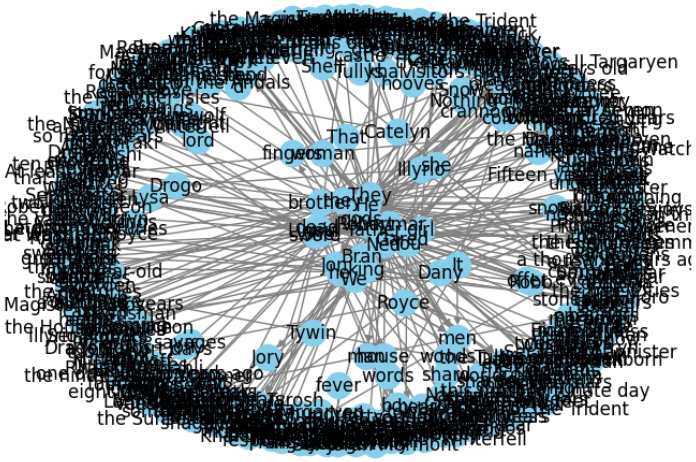

Figure 2: Knowledge Graph for First Fifteen Chunks of *A Game of Thrones*

This process was repeated to generate a knowledge graph for all chunks containing two given books in the series (not pictured).

---

3 System Architecture

3.1 Base Architecture

3.1.1 Summary

The Base Architecture is what is implemented in both RAG and KRAG architectures. The Base Architecture uses the dotenv module to get a user's API key from a .env file, loads up the embedding with the API key, loads up a vectorstorage variable for similarity search, and loads up large language model (LLM) using the API key. Both RAG and KRAG architectures also have token control functionality as OpenAI models have token limits to control bandwidth.

3.1.2 Similarity Search

Similarity search implements the "retrieval" part of "retrieval augment generate". The system retrieves OpenAI's embeddings and Facebook AI Similarity Search (FAISS) using the langchain module. The system passes the processed chunks of *A Song of Ice and Fire* books and OpenAI's embedding into the FAISS object and retrieves the k most similar chunks using its similarity_search method.

3.1.3 Token Control

OpenAI's gpt-3.5-turbo model has an total token limit of 4096 tokens and reserves 256 tokens for generating a response. This gives the system 3840 tokens for a query. Both RAG and KRAG implementations use the tiktoken module to make sure queries do not go over limit.

3.2 RAG

The RAG architecture passes two parameters to an LLM's query: a question and a context. First, a user enters a question into the system. As mentioned earlier, the "retrieval" in "retrieval augment generate" is done using FAISS. FAISS finds the k-nearest chunks or files of a given question, which in this implementation is the 3 nearest 1,000 character chunks of the books; this is the "context" of the query. Since a user's question cannot be cut from a query to get a proper response from an LLM, the system does a calculation to see how many tokens need to be cut from the context in order to meet the token requirements. Once the context tokens are cut appropriately, the context and question are passed to a template that serve as the query passed to the LLM for a response. The context and question stitched to a template for query act as the "augment" portion of "retrieval augment generate" and the response from the LLM acts as the "generate" portion.

3.3 KRAG

The KRAG implementation is nearly identical to the RAG implementation, but it includes an addition query item, knowledge graphs triples, and appropriate token controls for the additional item. Given a question by a user, the KRAG query, on top of what the RAG query already does, finds the entities in the given question that match those in the knowledge graph, finds the relationships that entity has with other entities/nodes, and returns that information in the form of triples, a data structure developed by Tim Berners-Lee. After some token control, those triples are passed to the query along with context and question, which is passed to the LLM to generate a response.

---

4 Experimental Results and Analysis

4.1 Testing

We began by asking several questions from the "A Song of Ice and Fire" series to both the KRAG model and the RAG model. Each model's responses were evaluated for correctness, depth, and alignment with canonical facts and speculative ability. The evaluation considered both factual correctness and whether the answers demonstrated deeper insights or contextually relevant elaboration. The fields of questions are as follows:

4.1.1 Basic Factual Recall

These questions are simply to ensure that the model can recall basic facts within the text, especially when the questions involve something obscure that is not often discussed by outside information like the show or fans. It is our belief that KRag should answer these better, since they will build relationship graphs based on the knowledge from the books, and have more in depth information.

E.g.

Question: What were Cotter Pyke's feelings on sailing to Hardhome to rescue the wildlings?

Krag Answer: Cotter Pyke was hesitant to sail to Hardhome to rescue the wildlings, as he believed there was no shelter there and that they would likely perish from cold and starvation.

He also mentioned the "screaming caves" and the dangers of the wildlings. However, Jon Snow was angered by this and reminded Cotter that they were still people who needed to be saved. ¿ RIGHT

Rag Answer: Cotter Pyke was hesitant and reluctant to sail to Hardhome to rescue the wildlings. He believed that there was no shelter there except for the caves, which his men called the "screaming caves." He also feared that Mother Mole and her followers would perish there from cold and starvation. However, he was also aware that there were thousands of wildlings, including women and children, who would die if they were not rescued. He was torn between his duty to the Night's Watch and his fear for the safety of his men. ¿ RIGHT

Question: What happens to Benfred Tallhart and the Wild Hares?

Krag Answer: Benfred Tallhart and the Wild Hares are killed by the Magnar of Thenn and his bronze-clad minions in the tunnels beneath the Wall. ¿ WRONG

Rag Answer: Benfred Tallhart and the Wild Hares are captured by the Night's Watch and taken to Castle Black. They are then trained and armed by Jon Snow and sent to various forts along the Wall to help defend against the wildling attacks. ¿ WRONG

4.1.2 Inference

These questions involve making conclusions from evidence offered in the story. We are of the belief that these questions should be better answered by KRag than RAG.

E.g

Question: Why does Ramsay Bolton claim, in the pink letter, that he had captured Mance Rayder? Is it truely Ramsay's letter, how did he know, and if so, how did he capture Mance Rayder?

Krag Answer: Ramsay Bolton claims to have captured Mance Rayder in the pink letter as a way to taunt Jon Snow and the Night's Watch. It is not truly Ramsay's letter, but rather a forgery created by Mance Rayder and his allies in order to trick Jon Snow into helping them rescue "Arya Stark" (who is actually Jeyne Poole, a friend of Sansa Stark's who has been posing as Arya). Mance Rayder was not actually captured by Ramsay, but rather he and his allies staged a fake execution and then escaped. ¿ MOSTLY RIGHT

Rag Answer: It is unclear why Ramsay Bolton would claim to have captured Mance Rayder in the pink letter. Some theories suggest that Ramsay may have been lying in order to provoke Jon Snow into taking action against him, while others believe that Ramsay may have actually captured Mance and is using him as a bargaining chip. It is also possible that the letter was not actually written by Ramsay, but by someone else who is trying to manipulate Jon Snow. The true author of the letter is still a mystery. As for how Mance was captured, it is possible that Ramsay's men were able to capture him during the chaos of the battle at Winterfell, or that he was betrayed by someone in his own camp. ¿ MOSTLY RIGHT

4.1.3 False Assumption Query

These questions make a false claim within the question itself, intending to determine if the model will hallucinate based off of the false information offered in the query or if it will correct the questioner.

E.g

Question: Why did Theon fear Shagga?

Krag Answer: Theon feared Shagga because he was a fierce and unpredictable warrior from the mountains of the Vale, known for his love of violence and brutality. ¿ FALSE

Rag Answer: Theon feared Shagga because he was a fierce and unpredictable warrior from the Hill Tribes of the Vale. Theon was not used to dealing with such wild and savage fighters, and he knew that Shagga and his people were not loyal to any one lord or king. This made them a potential threat to Theon's rule and he feared that they could turn against him at any moment. Additionally, Shagga was known for his brutal and violent tendencies, which made Theon fear for his own safety. ¿ FALSE

4.2 Analysis

KRag unfortunately did not do very well when examined. In two fields it performed worse than RAG, the Inference questions and the Basic Factual Recall, much to our surprise. When given basic questions it often misunderstood and delivered an answer way out of left field, often fixated on characters "pledging their swords" to an obscure character called "Inkpots," which neither member knew anything about until further research had been undertaken. Other times, however, when more obscure questions were asked, KRag would likely answer more faithfully. Inference based questions varied in response quality wildly. Whenever given the opportunity, the model would either deliver a rich response, considering multiple connections and factors, or it would deliver something convoluted and hallucinated. Strangely the field it performed better in than the RAG model was False Assumption Query, because oftentimes, the RAG model had responses tainted by shows and outside fan theories, so in multiple cases where both models responded somewhat accurately, KRag would put a greater emphasis on the knowledge generated by the books, and so gave more useful answers.

---

5 Challenges Faced and Solutions Implemented

5.1 NER Model/Mainframe 3 Access

5.1.1 Problem

At the start of the project, the team was running into issues when trying to implement a NER model from scratch. The model was taking an hour to generate on one of the group member's computers and was crashing immediately due to a segmentation fault on the other member's computer. This was diagnosed as a memory issue, and a member attempted to amend these issues by running the project's system on Southern Methodist University's Mainframe 3 High Performance Computing (HPC) cluster system. Unfortunately, after an hour or so of downloading a virtual private network, reading documentation, and tinkering with the remote desktop, the member realized that in order to run the project he needed access to a SLURM account on the system, which required a professor's approval.

5.1.2 Solution

Knowing that it would take too long for the school to give access to a SLURM account, already having this assignment extended from its initial deadline, and not wanting to bother the class' Professor upon the last part of his last semester until retirement, the group decided against requesting access to a SLURM account. Instead, the group looked at the spaCy documentation and tried to figure out how to fix this memory issue, until they realized that loading up a pre-built model instead of building an NER model from scratch avoided the issue all together. Thus, they decided against building a model manually and used the pre-built model instead.

---

6 Future Improvements and Potential Applications

6.1 Future Improvements

6.1.1 Separate Knowledge Graph Construction Script

The main Python file generates a knowledge graph in each instance of the system. There is likely a way that a knowledge graph can be generated once, stored, and loaded up during in order to avoid the unnecessary wait time of the current implementation.

6.1.2 Similarity Search Retrieval

The system only allows two books to be chunked due to bandwidth issues using OpenAI's API. There is a one million token per minute limit. When encoding three books, the limit is exceeded. There is surely a workaround which involves saving and loading an FAISS object. Instead of running all the books at once and exceeding the token limit, a viable option could be loading and saving a FAISS object at different times to not overwhelm the network. This object, similar to the prior improvement, could be built once, and then loaded up instead of generated during each instance of the system.

6.1.3 Chunk Splitting/Token Shortening Method

The manner in which the system splits chunks is brutish. The NER splits chunks according to sentences, and sentences are cut off unfinished in the current implementation due to chunks merely being a number of characters. A better implementation would have sentences not cut off while not exceeding a character limit.

A similar improvement could be made to the method which token control brutishly cuts context in order to meet token quotas both RAG and KRAG architectures. The current implementation cuts off tokens from start to the number of tokens needed to meet quota. A better implementation would retrieve different numbers of "K" chunks using similarity search, starting at one until reaching the quota limit for tokens, and if a chunk be too large token wise, some sort of token requirement summary method used to capture the idea. Or, if that summary method implemented, perhaps a summary method to summarize the K number of chunks retrieved by similarity search in the first place.

6.1.4 Automatic Token Control Calculation

The current implementation has a static calculation for token control only for the gpt-3.5-turbo model. It makes sense for there to be a dynamic calculation for future implementations with additional models.

6.2 Potential Applications

6.2.1 Web Application using RAG or KRAG for Attracting Recruiters and Clients

Personal websites are important assets for software engineers. A nice feature for a personal website, to demonstrate understanding in the AI domain, would be a RAG or KRAG system where any website visitor (hopefully a recruiter or client) can ask details about a candidates career and personal life.

6.2.2 Business/Legal Application

There are still big legal and business use questions in artificial intelligence related to copyright infringement and privacy. Companies do not want to feed classified or competitive information to LLMs for their private use. AI companies could be held liable for training their models outside of fair use and for replicating copyrighted media, which could change the landscape. There is also a time limit for each model's training, e.g. many of OpenAI's GPT models only know information up to a certain date.

Thus, people who know how to integrate LLMs locally or at a company level using RAG or KRAG, that know how to feed a model more information that it might not already "know", may have a sought after skill in the job market. User friendly software that streamlines local LLMs using RAG and/or KRAG may also be attractive ventures for business applications.