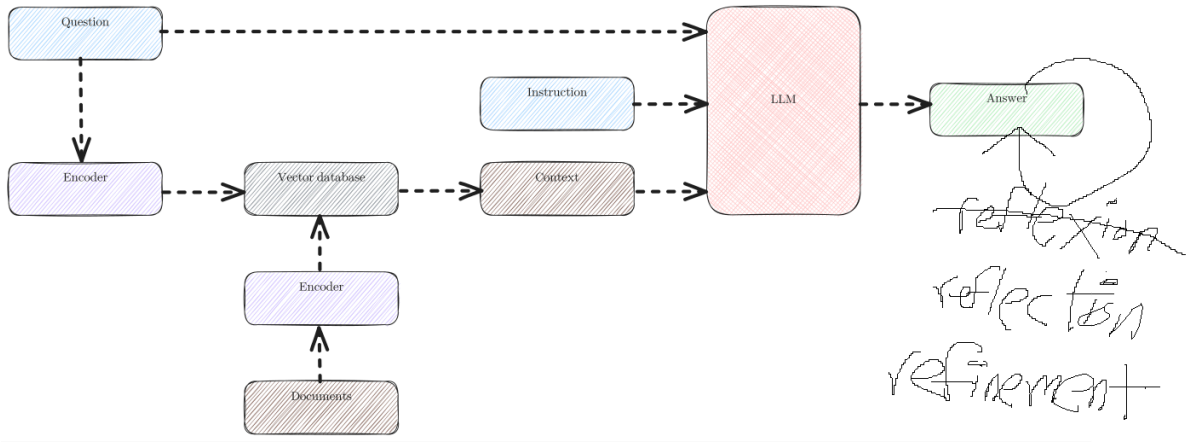
CSTU GPT Application w/ prof Yang Sun
Final Project 8/31/2024
Team 4 - Justin Chen, Xu Liu

This document will talk about our movie recommendation LLM program aka Film Finder GPT.

The code is all self-contained within the film_finder_gpt.ipynb Jupyter notebook.
You must have the movies_metadata.csv within the same folder to load the initial pandas dataframe.

Movies-16384-1724889754.pkl is a serialized dataframe that enables you to run the notebook from the cell 9 checkpoint, where the focus starts shifting from RAG concepts like embedding generation and vectorstore similarity search retrieval to Self-Reflection prompt engineering and LLM agent concepts.

Here is an extremely rough flowchart diagram of our program, mostly taken from wikipedia..



Here is an overview of the algorithm:

We begin by loading 45000+ csv rows into a dataframe.
We then cut all columns except for title, rating, release_date, genres, and overview.
We then drop any rows with bad data, and drop all except the first [:16384] rows (2^14).

Then for every row we combine the content of the remaining columns to create a movie_text, and we count the number of tokens each text costs using the c100k encoding that is used for OpenAI's "text-embedding-3-small" model.

Next we generate the [1 x 1536] vector embedding for each movie_text, being sure to stay under the 150k token limit per batch.

Then we index the embeddings into facebook's FAISS vectorstore so that we can perform similarity search with the user request embedding later. This lets us retrieve the k most similar (in meaning) movie texts to the user's request string.

The retrieved movie texts are passed as context along with the user request string to compose the first HumanMessage in our initial prompt.

This initial prompt consists of 2 messages, a SystemMessage assigning the LLM the role of a movie recommendation expert, and the HumanMessage describing what the user feels like watching along with the 6 nearest movie_texts as context.
The SystemMessage tells the LLM to pick 2 out of 6 of the context movies to recommend along with the reasoning behind its selection.

This initial 2 message prompt makes the LLM (gpt-4o-mini) act as a movie recommendation article generator.

Next we want to make the LLM act as a very skilled literary critic, generating critique for the movie recommendation article we just produced. This will then begin the self-reflection loop of continuously revising our movie recommendation article with our thoughts and observations about it.

The initial reflection prompt also consists of 2 messages, a SystemMessage followed by the AIMessage recommending movies we just generated.

The SystemMessage instructs the LLM to act as the editor in chief of a highly acclaimed pop culture magazine. We tell the LLM to score the quality of the movie recommendation article from 1 to 10 along with the reasoning behind the score.
We then tell the LLM to suggest the three best ways to improve the score without changing the 2 recommended movies (so that we don't have to do any more vectorstore retrieval).

With these 2 connected but very different prompts, we direct the LLM to act as 2 separate agents, the first being a movie recommendation expert and the second being a world class literary critic.

In order to create a looping structure of revising our movie article with the critic's feedback, followed by generating new feedback for the revised article, repeating as long as we want, we use LangGraph to connect the output messages or our generate and reflect chains (using the Runnable interface created by LangChain).

At this point I realized I needed a small update my initial generator prompt, updating the very first System Message to let the LLM know that if critique is provided to its movie

recommendation article, that it should revise its article by incorporating said feedback, with an explicitly stated goal of trying to raise the reflector's score as much as possible.

With our generate and reflect chains defined as LangGraph nodes and connected in a loop with directed edges to form a MessageGraph, with our entry point set on the generator node, our final task is to set our exit condition to finish the loop.

Quick aside, I also defined a RAG_chain before LangGraph/reflection comes into play, with the purpose of taking an arbitrary string as the user's request and using runnable chaining + RAG to engineer the initial HumanMessage in our initial prompt to the movie recommendation generator agent.

This HumanMessage becomes the only message in our initialized state once we invoke our MessageGraph generate/reflect looping structure. From there each time the LLM outputs a generation or reflection article, it appends that article to our MessageGraph state, which is just the list of messages our graph generates in order.

I wanted to exit the loop once our reflection agent gave our generator agent a high enough score, but for fear of running out of time I chickened out and simply exited the graph once it added the 8th message to its state, which is the 3rd revision of the movie recommendation article.

To be extremely explicit, the state message list at that moment will look like.

1. HumanMessage with initial movie request + RAG context
2. AIMessage initial movie recommendation article
3. HumanMessage with first reflection + score on initial movie article.
4. AIMessage with first revision of original recommendation article incorporating reflection feedback.
5. HumanMessage with second reflection + score
6. AIMessage with 2nd revision of article
7. HumanMessage with third reflection + score
8. AIMessage with 3rd revision of article

To my amazement each reflection iteration provided massive gains to the movie recommendation article. In the submitted notebook the reflection score went from 6/10 to 9/10 to 10/10. In another trial I've seen the score jump from 4/10 to 8/10 to 10/10. My prompts are not even very detailed, I basically just learned what prompt engineering even is. The reflection framework basically feels like you just get big quality increases for free, though I am aware it's not truly free and costs cpu power and a bit of effort to set up the agent.

But simply by telling the LLM "make this thing better", it does just that, and it seems super generalizable to whatever domain you want, not just movie recommendations.

Anyways this was a very interesting class, before this term I had no idea what NLP LLM or GPT even was. Taking deep learning with professor Shen definitely also helped my understanding a lot, to understand the underlying neural network concepts like parameters, training methods, back propagation etc. It's quite cool to see how much image and language processing have in common, despite them existing in totally separate senses.

Also my python skill improved considerably while working on this project, this language is actually great. Now I am looking forward to the python for AI course offering next term. Prior to this javascript was my forte, and that was quite a while ago. If you actually read to this point, thank you very much, and wish you the best in all your future endeavors.

Also I should perhaps mention that I considered using Pydantic to explicitly define the score as an integer field for each of the reflection articles. I saw you can also do similar structure formatting with JSON. But I got distracted by so many rabbit holes during the course of making this notebook that there is no way I had time for that.

I also considered using one or few shot learning to give some example movie recommendation articles of what would be considered a 1/10, 5/10, and 10/10. However even with zero shot learning the reflection agent does a very nice job of providing scores.