Regarding my approach, I needed to consider the amount of computing power I had available for this task. I built my personal PC back in 2018 and have not made any upgrades since then, so it is lacking in computational power. That is part of the reason why I had to opt for a lower power LLM to summarize my documents. I originally attempted to run a deepseekr-1:32b and Ollama3.3 LLM, but both required much more RAM usage than my pc has available (I only have 16GB RAM on my PC). I then opted for deepseek-r1:14b and Ollama 3.2:3b. Ollama 3.2:3b did not provide such good results, and deepseek-r1:14b took a little too much time on my PC with the time constraints I have. Due to this, I eventually decided to proceed with deepseek-r1:8b to increase computational power and reasoning, but also not lessen the quality of output received from the LLM.

For my document analysis, I decided to obtain form 8-K URLS for the SEC EDGAR website for S&P 500 companies. The way I structured my code was with 3 functions:

***obtain\_urls:***which found working 8-K URL links for the requested companies from SEC EDGARS master file directory. This function took the following as input: **year**: in format 20XX as an integer, **quarter**: as a string in format "QTR#" where # is 1, 2, 3, or 4, **form**: string that is either "8-K", "10-K" or "10-Q", **total\_forms**: the number of total forms to parse as an integer, and **company\_list**: a dictionary where the key is the company's name as a string and the value is a list of strings in the form ['CIK#', 'Ticker']. With the inputs, you can decide what year, what quarter, what form, how many forms, and which public companies you would like to parse documents for.

***llm\_parser:***which downloaded the 8-K forms from the provided URLs and fed it to an LLM for summarization. The SEC EDGAR 8-K forms were provided as .htm, and so when using requests.get() I used ().text instead of ().content since we were focusing on language analysis with the LLM. The model used was deepseek-r1:8b. To obtain output in my desired format consistently, I used the BaseModel class from pydantic to make the output JSON structured for easier extraction. This function received the URLs and company information from the *obtain\_urls* function output as the input.

***write\_to\_csv*:** which wrote the output results from the LLM to a csv file for review.

Before the functions were run, I obtained the names for S&P 500 companies in the code through Wikipedia to specifically only download 8-Ks for those companies. The filtering was then automated in the *obtain\_urls* function.

In my case, due to computing power and time constraints, my final csv output is for only 100 documents as requested in the assignment instructions, although as can be seen in the function information above, that can be changed to whatever amount you are wanting. In the case you have a PC designed for LLM use, you can use a much better LLM model on a much larger number of documents with this code and be able to achieve much better summarizations.

The following are some of the challenges I faced during this project.

* Model Selection – Different LLM models are better at certain tasks. Although I could not see a huge difference between the deepseek-r1 and Ollama models I tested, I decided to go with deepseek-r1:8b as it gave me a better balance of computational power while not vastly increasing processing time.
* Time Constraints – Similar to the above challenge, better more accurate models would have taken my ancient PC a much longer time to process. Since we need to parse 100 documents minimum, I decided to go with deepseek-r1:8b. This allowed me to have better summarization than Ollama3.2 but still allowed me time to re-run and test on many documents.
* Computing Constraints – My PC does not have enough RAM available to run models more powerful than the deepseek-r1:14B. I ran out of RAM and my programs terminated for deepseek-r1:32B and Ollama3.3. This limits how well my LLM can summarize the data along with taking longer to process everything.
* LLM Prompt Instructions – Being able to write a specific prompt for the LLM turned out to be a challenge. Initially I only asked the LLM to find the requested information, but that did not turn out to be enough of a prompt. After many tests, I ended up with a long prompt that specified many things about the output I wanted. That included what wasn’t considered a new product, what to do if no new product was mentioned, providing generic product names, etc. I also had to request the LLM to not include their reasoning or thinking in the output, as that seemed to be a reoccurring issue included in the output data during certain test runs.
* LLM Output Accuracy – Even with the specific instructions as mentioned above, the LLM still seems to provide some accurate and some inaccurate. Even when running the same code and second or third time, the LLM can give different results. There were even cases where I reviewed the 8-K given to the LLM, and I noticed that there was no new product mentioned in the 8-K but the LLM still outputs a new product name and new product description. I attempted to remedy this by including in the prompt that if the output product is not in the document, then there should be no new product listed. Additionally, changing the temperature and top\_k of my LLM model did not produce more appropriate results.

Ultimately, most of my challenges on this project were related to the LLM itself. I believe if I could have worked with a PC dedicated to AI and language processing, I could have provided a CSV output with much more accurate output for new product names and new product descriptions. That would also allow for more testing on my end to see what LLM settings and prompts result in better output data.

Github URL: <https://github.com/arey06>

Project URL: https://github.com/arey06/CAP5619-LLM-Project/blob/main/parser%20output.py