Individual report final assignment

Cecilia Nyberg

December 23, 2024

1 Contribution

1.1 Creating the DB

The first task I did was creating the database. The database should include pre-text, post-text, and tables from the FinQA dataset. Several alternatives for structuring the database were suggested in the hints. I experimented with these options and evaluated the agent's performance. Initially, I embedded the pre-text, post-text, and table together while keeping the table in its original structure. However, this approach resulted in poor accuracy, around 15%.

To improve the results, I tried other methods. The most effective approach involved structuring all tables into JSON objects (using an LLM, since the tables lacked a consistent pattern) and extracting the column names for each table. I then placed the structured tables as chunks in the text database and included column names in the metadata. Although the hints recommended the opposite setup, this approach yielded better results. The database could have been further optimized by fine-tuning the embeddings using the training set, but time constraints prevented this. I also experimented with chunk sizes and overlap. A chunk size of 1100 with an overlap of 300 provided the best results. The accuracy was now around 25 %.

1.2 Tools

Once the vector database was created, I focused on developing additional tools for the planning agent. These tools were guided by the LLM's shortcomings in answering questions. For some queries, the correct data was retrieved, but slight errors in calculations led to incorrect answers. For example, when calculating a percentage, both values in the calculation were correct, but possibly due to data type issues (e.g., integers), the division gave incorrect results. Fixing such issues seemed straightforward, so I prioritized implementing them first.

I developed the tools: `operating_profit_margin`, `percentage_change`, `calculate_ratio`, and `calculate_difference`. These improvements raised the overall accuracy to around 30%.

1.3 Query Decomposition

The next challenge was improving the agent's ability to handle complex queries. These queries, often composed of multiple sub-questions, needed to be broken down to simplify the planning process and improve context retrieval. Initially, I considered allowing the planning agent to decompose the queries itself, but this approach risked overloading the agent with too many responsibilities.

To address this, I implemented a **query_decomposer**, an LLM specifically prompted to decompose

queries into sub-queries. Using few-shot prompting, the query decomposer was fine-tuned to perform this task.

1.4 Query Classifier

To enable query routing, I implemented a query classifier to categorize queries into low, medium, and high difficulty levels. My teammates worked on this in parallel. Initially, the classifier struggled, classifying most examples as high difficulty. To improve its performance, I added tools such as allowing it to perform database retrieval, which gave the classifier more context for making decisions. This enhancement significantly improved the classifier's accuracy and made the results more useful for downstream tasks.

1.5 Chain-of-Thought

After implementing the classifier, I introduced functionality tailored to different difficulty levels. The planning agent was updated to account for the difficulty level in its planning. For high-difficulty queries, the agent used advanced RAG methods.

Since the retriever often struggled to find relevant documents, a chain-of-thought (CoT) approach was introduced. The agent first decomposed the query using the designated tool, classified each sub-query, and then selected the appropriate tools and retrieval methods based on the difficulty level. Advanced methods were used when necessary, while simpler methods were used for less complex queries.

1.6 Advanced RAG - Hybrid Search

For advanced retrieval, I implemented hybrid search, which combins semantic similarity with keyword matching. Keyword search relies on the `bm25_retriever`, which uses Inverse Document Frequency (IDF) for term matching. This approach prioritizes exact keyword matches over embeddings. The way it performs its search is described in the equation below.

$$\text{BM25}(q,d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{f(t,d) \cdot (k_1 + 1)}{f(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)}$$

$$\text{IDF}(t) = \log\left(\frac{N - n(t) + 0.5}{n(t) + 0.5} + 1\right)$$

Initially, I let the planning agent select the retrieval method and approach and the generative agent got the output. However, this added unnecessary complexity and confusion. Following the assignment's guidelines, I created a separate retrieval agent so that retrieval responsibilities were isolated. This way, the retrieval agent could reason about the result before returning it, so that the generating agent could focus on producing answers without being overwhelmed by irrelevant context.

1.7 Query Expansion

To enhance retrieval, I initially implemented query expansion using WordNet to extract synonyms. The retrieval agent could use these synonyms to broaden its search. Unfortunately, this approach caused confusion and degraded performance.

Instead, I instructed the retrieval agent to filter and rephrase queries when relevant documents were not found. This approach worked better.

1.8 Filtered Search

Retrieval limitations sometimes led to difficulty in finding relevant documents. While I considered implementing reranking and other search techniques, time constraints prevented this. Upon inspecting the database, I noticed that correct information often appeared in tables, and by looking at retrieved documents, I saw that these were less often retrieved. This may have been due to tables containing less compact information compared to pre- and post-texts. Pre- and post-texts contained filler words that matched queries more easily, favoring these documents in retrieval.

To address this, I updated the `retrieve_factual_data` function to incorporate multiple retrievers. The original retriever (retrieving three documents) was retained, and a new retriever specifically filtered for tables (retrieving two documents) was added. Optional retrievers for units and keywords were also included. All retrieved documents were merged and returned. After these adjustments and minor prompt refinements, accuracy improved to 54%.

1.9 Additional Attempts

I experimented with other approaches to improve accuracy, but many of these reduced performance. For example, making the planning agent rewrite queries using synonyms from a "think-tank" decreased accuracy. Similarly, implementing a lookup table of financial terms and definitions did not yield positive results. While new solutions sometimes corrected previously incorrect answers, they also introduced errors in previously correct responses. With more time, I believe query routing could have been further refined to only apply the new stragies to some questions.

2 What I Have Learned

I gained a better understanding of several advanced methods for RAG such as hybrid search, reranking, and query expansion and how these techniques can be applied to improve the performance of LLMs in complex tasks.

Additionally, I realized how sensitive an LLM agent can be to excessive information and responsibility. Dividing responsibilities among multiple specialized agents, as demonstrated in this project, can significantly improve performance. For example, separating query decomposition, retrieval, and generation into distinct tasks helped reduce confusion and improve accuracy.

I also discovered the balance required in providing instructions to LLMs. While specific instructions are very effective for narrowly focused tasks (e.g., query decomposition or classification), too strict guidelines for broader tasks seem to limit the agent's ability to adapt. For example, specific query rewriting and strategy rules caused the agent to select inappropriate tools and strategies.

Using the ARKMan architecture in this project was a valuable learning experience. The modular, multi-agent structure provided a clear framework for dividing tasks, which helped with common issues such as overloading an agent with too many responsibilities.

One of the key benefits of ARKMan was its adaptability. By isolating responsibilities, it was easier to experiment with different retrieval strategies and tools without affecting other components of the system. For example, implementing a separate retrieval agent allowed me to integrate hybrid search and filtered retrieval methods effectively.

However, the ARKMan structure also had some challenges. Configuring the interactions between agents required careful consideration to avoid confusion and too much information being passed.

Overall, using ARKMan deepened my understanding of modular AI architectures and their advantages in complex systems. I saw the importance of task-specific agents and clear interfaces in designing

solutions to complicated tasks.