Report: System Design, Technologies, Challenges, and Future Development

Dominik Olejarz

09.04.2024

System Design

The provided code implements an article search system that retrieves relevant articles based on a user query. The system preprocesses the data, calculates the similarity between the query and articles using TF-IDF and cosine similarity, and retrieves the most relevant article fragments. It utilizes various libraries and modules for data processing, including the embedding model *BAAI/bge-small-en-v1.5* from Hugging Face.

Technologies Selected

- **Pandas and NumPy:** Utilized for data manipulation and numerical operations.
- **Scikit-learn:** Employed for TF-IDF vectorization and cosine similarity calculation. Cosine similarity is widely acknowledged as a standard metric for comparing vectorized data. Additionally, using TF-IDF with provided stop words is considered a good choice for determining the importance of words. Alternatively, Word2Vec could also be considered.
- **langchain:** A custom library for text processing, embedding, and retrieval, including modules for text splitting, embeddings using Hugging Face models, and document retrieval.

I selected:

- **RecursiveCharacterTextSplitter:** I chose this method for text splitting because I believe it's the most efficient splitter that doesn't rely on LLM (though LLM-based splitters are undoubtedly superior due to their contextual understanding). However, this method is significantly faster.
- **BAAI/bge-small-en-v1.5 embedding model:** I chose this model because it's small, open-source, and integrates well with the project due to its speediness. Additionally, in my alternative solution where I don't use a

pretrained model, I employ a statistical retriever using the BM25 algorithm, commonly used in document retrieval, text search, and information retrieval tasks.

Challenges Encountered

- **Data Preprocessing:** Efficiently handling and preprocessing textual data. Initial attempts didn't utilize TF-IDF for first selecting the most relevant articles, resulting in significant slowness when processing the entire dataset.
- **Fragment Retrieval versus Answer Size:** Several attempts were made to optimize the parameters to maximize this 'heuristic.' The solution aligns with expectations, but if they differ, a class capable of accommodating various types of expectations was developed.
- **Integration of External Libraries:** Ensuring compatibility and proper integration of external libraries like scikit-learn and langchain into the system.

Potential Areas for Future Development

- **Performance Optimization:** Explore methods to improve the efficiency and scalability of the system, such as parallel processing or optimizing the retrieval algorithm.
- **Enhanced Embeddings:** Experiment with different embedding models or techniques to capture semantic meaning more accurately and improve search results.
- **User Interface:** Create a user-friendly interface for the system, enabling easy interaction with the search functionality and offering feedback on search results to improve relevance. This interface should be particularly useful for novice programmers seeking crucial information about code projects within code documentation.
- **Extension to Multi-modal Data:** Extend the system to support multi-modal data, such as incorporating images or audio transcripts into the search process for more comprehensive results.
- **Integration with External APIs:** Integrate with external APIs or databases to fetch real-time or additional data sources, expanding the scope and usefulness of the system.
- **Integration with LLM:** Reference and explain returned fragments using LLM, enhancing user experience and expanding potential applications.

Conclusion

The current implementation provides a foundation for an article search system, but there are several opportunities for enhancement and refinement to meet the evolving needs of users and address potential challenges in the future.