Financial Statement Question-Answering with a Retrieval-Augmented Generation (RAG) System

Tianyi Zhu

1 Processing Details

1.1 Overview

The data processing workflow is essential for transforming raw 10-K financial filings into a clean, structured format suitable for downstream tasks in the Retrieval-Augmented Generation (RAG) system. This section provides a detailed description of how the data was cleaned, organized, and prepared for analysis.

1.2 Step 1: Directory Structure Organization

The raw financial reports were organized into a consistent directory structure to facilitate efficient access and processing:

- **Raw data directory:** `ORIGINAL_DATA_DIR`
- **Processed data directory:** `PROCESSED_DATA_DIR`

For each ticker in the sampled list, financial filings were stored as follows:

- **Ticker-level folders:** Named using the company's ticker (e.g., `FRT`, `CCL`).
- **Year-level subfolders:** Containing all filings for a specific year under a `10-K` folder.

This hierarchical structure ensured logical navigation and data segregation by company and year.

1.3 Step 2: Validation and Filtering

To ensure data integrity:

1. **File Validation:** For each ticker, the presence of a `10-K/full-submission.txt` file was checked. Missing files or folders triggered logged warnings, and processing continued with valid files.
2. **Year Extraction:** Using regular expressions, the year was extracted from folder names following the pattern `-YY-`. For instance, `-18-` was parsed to derive the year as `2018`.

Tickers from the provided sampled list `['FRT', 'CCL', 'IPG', 'WRB', 'IEX', 'TECH', 'PHM', 'LVS', 'ROP', 'ULTA']` were cross-checked to ensure all filings were processed within the required years (2010 to 2020).

1.4 Step 3: Cleaning Financial Reports

The raw content of each 10-K report was cleaned to remove irrelevant and redundant information:

1. **HTML Parsing with `BeautifulSoup`:**
   - Each report was parsed as HTML to identify and remove boilerplate content.
   - Tags such as `<script>` and `<style>` were decomposed to eliminate scripts and styles.
2. **Plain Text Extraction:**
   - The remaining content was extracted as plain text using `get_text()`.
   - Line breaks (`\n`) were added to preserve some document structure.
3. **Whitespace Normalization:**
   - Multiple consecutive line breaks were reduced to a single line break.
   - Leading and trailing whitespaces were stripped from the content.
4. **Fallback for Non-HTML Reports:** If HTML parsing failed, the raw content was cleaned as plain text to ensure no data was lost.

These steps ensured that the final text was concise, human-readable, and free from non-informative formatting.

1.5 Step 4: Processed Data Storage

The cleaned financial filings were saved systematically for easy access during analysis:

- **Folder Structure:** Processed data for each ticker was stored in a dedicated subdirectory under `PROCESSED_DATA_DIR`.
- **File Naming:** Cleaned content for each report was saved as `content.txt` in the respective subdirectory.

This naming convention and structure allow seamless retrieval and alignment of processed data by company and year.

1.6 Step 5: Error Handling

Comprehensive error-handling mechanisms were implemented to ensure robustness:

1. **Missing Files or Folders:** Warnings were logged for tickers or years without the required `full-submission.txt` file.
2. **Parsing Errors:** Reports that failed HTML parsing were cleaned as plain text, with the errors logged for review.
3. **File Saving Errors:** Any issues in saving processed files were logged with detailed messages for troubleshooting.

1.7 Summary of Sampled Tickers and Processed Data

The following tickers were included in the final analysis:

- **Sampled Tickers:** `['FRT', 'CCL', 'IPG', 'WRB', 'IEX', 'TECH', 'PHM', 'LVS', 'ROP', 'ULTA']`
- **Years Processed:** 2010 to 2020 (inclusive)

All reports for the sampled tickers within the specified timeframe were validated, cleaned, and stored in `PROCESSED_DATA_DIR`. This structured and clean dataset is ready for vector store construction and subsequent analysis within the RAG system pipeline.

2 Vector Store Construction

2.1 Chunking Strategy

To effectively process large financial filings, a chunking strategy was implemented to divide the text into manageable segments:

- **Sentence Segmentation:** The text was split into sentences using a regular expression that detects sentence-ending punctuation such as periods (`.`), exclamation marks (`!`), and question marks (`?`).
- **Chunk Formation:** Sentences were aggregated into chunks, ensuring that each chunk's length does not exceed a predefined size of 512 tokens. If a sentence caused the chunk to exceed this size, the current chunk was finalized, and a new chunk was started.
- **Finalization:** Any remaining text that did not form a complete chunk was saved as the final chunk, ensuring no data loss.

This strategy ensures that each chunk contains sufficient context for embedding generation while adhering to the input size limitations of the embedding model.

2.2 Document Embeddings

Document embeddings were generated using a transformer-based model from Hugging Face. The following steps summarize the embedding generation process:

1. **Model Selection:** A pre-trained model (e.g., `MiniLM`) specified in the configuration file was loaded along with its tokenizer.
2. **Tokenization:** Each text chunk was tokenized into subwords using padding and truncation to meet the model's input size requirements.
3. **Embedding Calculation:**
    - The tokenized input was passed through the transformer model, which produced contextualized embeddings for each token.
    - The mean pooling operation across the sequence dimension was used to generate a single fixed-length embedding vector for each chunk.

4. **Device Optimization:** The process was executed on GPU (`cuda`) if available, or CPU otherwise, ensuring computational efficiency.

The resulting embeddings are dense vector representations of the text chunks, capturing their semantic meaning.

2.3 Storing in the Vector Database

The processed embeddings and their associated metadata were stored in a FAISS vector database, designed for efficient similarity search. The following steps outline the storage process:

- **Index Initialization:** A FAISS index was initialized using the L2 (Euclidean) distance metric. The embedding dimension was set to 384, matching the output size of the embedding model.
- **Adding Embeddings:** Each embedding was added to the FAISS index. This step ensured that the vector store maintained efficient similarity search capabilities.
- **Metadata Storage:** Metadata for each chunk, including `ticker`, `year`, `chunk_id`, and the chunk's `content`, was stored in a NumPy array. This metadata was saved separately for later use during retrieval and analysis.
- **Storage and File Management:**
  - The FAISS index was saved as `vector_store.index`.
  - The metadata was saved as `metadata.npy`.

  These files were stored in the directory specified by the configuration file.

2.4 Role of Metadata

Metadata is critical in providing contextual information about the stored embeddings and enabling effective retrieval. The metadata fields include:

- **Ticker:** The company associated with the chunk.
- **Year:** The year of the financial filing from which the chunk was extracted.
- **Chunk ID:** A unique identifier for each chunk within the filing.
- **Content:** The raw text content of the chunk.

The metadata enables:

1. **Efficient Filtering:** Retrieval results can be filtered by company (`ticker`) or year (`year`) to narrow the search scope.
2. **Contextualization:** Metadata enhances the interpretability of retrieved results by providing additional context.
3. **Debugging and Validation:** Metadata ensures traceability by linking embeddings to their original text chunks, aiding error analysis.

3 RAG Baseline Implementation

3.1 Retriever

The retriever module uses a FAISS vector store to identify relevant documents based on their semantic similarity to the query. The process involves:

- **Query Embedding:** The query is converted into a dense embedding vector using a transformer-based embedding model.
- **Similarity Search:** The FAISS index searches for the top-k most similar embeddings to the query embedding. Results are ranked based on their L2 distance to the query vector.
- **Metadata Filtering:** Retrieved documents are filtered based on metadata fields such as `ticker` (company identifier) and `year`. Only documents matching the specified criteria are returned.

This approach ensures efficient and accurate retrieval of relevant documents, narrowing the scope for subsequent processing.

3.2 Reranker

The reranker module refines the initial set of retrieved documents by computing their cosine similarity with the query embedding. The steps include:

- **Query and Document Embeddings:** The query and document content are embedded into dense vectors using the same transformer model.
- **Cosine Similarity Calculation:** The similarity score is computed as:

$$\text{Cosine Similarity} = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\|\|\mathbf{d}\|}$$

  where $\mathbf{q}$ and $\mathbf{d}$ are the query and document embeddings, respectively.
- **Re-ranking:** Documents are sorted in descending order of their cosine similarity scores.

This process ensures that the most contextually relevant documents are prioritized for response generation.

3.3 Metadata Filters

Metadata is critical for filtering documents during retrieval. The metadata associated with each document includes:

- **Ticker:** The company identifier for the document.
- **Year:** The year of the financial filing.
- **Chunk ID:** The identifier for the specific chunk within the document.
- **Content:** The text content of the document chunk.

Filters are applied to ensure that only documents matching the query's metadata constraints (e.g., `ticker` or `year`) are considered for retrieval and reranking.

3.4 Response Generator

The response generator uses the Together AI API to produce a natural language answer based on the query and retrieved documents. The workflow is as follows:

- **Context Construction:** The content of the top-ranked documents is concatenated to form the input context.
- **Prompt Generation:** A structured prompt is created, consisting of the context and the query. For example:

```
Context: {Document Content}
Question: {Query}
Answer:
```

- **Response Generation:** The prompt is passed to a language model specified by the configuration (`LLM_MODEL_NAME`) to generate a coherent and contextually accurate answer.

3.5 Evaluation Framework

The system was evaluated on a set of predefined queries for each ticker. The evaluation process involved:

- **Query Processing:** Each query was processed through the retrieval, reranking, and response generation pipeline.
- **Result Logging:** Retrieved documents, their metadata, and the generated response were logged for review.
- **Summary Report:** A summary report was generated, including key insights for each query and a preview of the retrieved documents.

3.6 Summary

The RAG system pipeline integrates retrieval, reranking, metadata filtering, and generative response capabilities to provide accurate and contextually relevant answers to user queries. By leveraging FAISS for efficient similarity search and Together AI for natural language generation, the system achieves high performance and reliability in answering queries based on financial filings.

4 Evaluation

4.1 Retriever Performance

The retriever's performance was assessed by analyzing the relevance of retrieved chunks for various test queries. Key observations include:

- **FRT:** For the query *"What does FRT say about its real estate business?"*, the retriever successfully retrieved highly relevant chunks with cosine similarity scores around 0.57. However, for the query *"What risks are highlighted in FRT's filings?"*, no relevant documents were found, indicating a gap in capturing nuanced topics.

- **CCL:** Relevant chunks discussing income tax challenges were retrieved for *"What challenges does CCL highlight in recent filings?"* with cosine similarities above 0.53. However, responses to operational updates were weak, with retrieved chunks containing irrelevant definitions and financial metrics.
- **IPG:** For the query *"What business activities does IPG focus on?"*, the retriever performed well, retrieving relevant chunks with cosine similarities around 0.67. No relevant documents were retrieved for risks-related queries.
- **Other Tickers (WRB, TECH, IEX):** The retriever either failed to find relevant chunks or returned overly generic content with low cosine similarities (around 0.5), reflecting limitations in dataset coverage or chunk-level representation.

Strengths:

- Effective in retrieving well-represented topics, such as FRT's business focus and IPG's activities.
- Robust metadata filtering ensured precise document retrieval by company and year.

Limitations:

- Poor handling of nuanced topics like risks or operational updates for certain tickers.
- Lower precision for broad or ambiguous queries, leading to irrelevant results.

4.2 Response Performance

The quality of generated answers was evaluated for coherence, relevance, and factual accuracy. Key findings include:

- **Coherence:** Responses were grammatically correct and well-structured, especially for queries with high-quality retrieved context.
- **Relevance:** For queries like *"What does FRT say about its real estate business?"*, responses accurately summarized the retrieved documents. In cases with irrelevant or sparse retrieval (e.g., WRB's risks), responses were less relevant or missing.
- **Factual Accuracy:** Answers demonstrated strong factual alignment when the retrieved chunks were relevant, such as PHM's summary of housing market trends.

Strengths:

- High accuracy and fluency for queries with well-represented topics.
- Effective summarization of nuanced details when sufficient context was retrieved.

Limitations:

- Responses failed when no relevant documents were retrieved, often returning *"No relevant documents found."*
- Redundant or generic content occasionally appeared, particularly for broad queries.

4.3 Conclusion

The RAG system performs well for well-defined queries with strong representation in the vector store, but struggles with nuanced or less common topics. Improvements in data coverage and retriever robustness are necessary to enhance overall performance.

5 Appendix: Running output

5.1 System Generating output

```
Using device: cuda
Processing query: What does FRT say about its real estate business?
Query: What does FRT say about its real estate business?
Response: FRT is an equity real estate investment trust (REIT) that specializes in the ownership, management, and redevelopment o

Processing query: What risks are highlighted in FRT's filings?
Query: What risks are highlighted in FRT's filings?
Response: No relevant documents found.

Processing query: What operational updates does CCL provide?
Query: What operational updates does CCL provide?
Response: Based on the provided context, it appears that CCL provides operational updates related to onboard and other goods and

Processing query: What challenges does CCL highlight in recent filings?
Query: What challenges does CCL highlight in recent filings?
Response: Based on the provided context, it appears that CCL is highlighting challenges related to income tax examinations and th

Processing query: What business activities does IPG focus on?
Query: What business activities does IPG focus on?
Response: IPG focuses on digital and international growth markets.

Processing query: What risks does IPG mention in its filings?
Query: What risks does IPG mention in its filings?
Response: No relevant documents found.

Processing query: What does WRB say about its insurance offerings?
Query: What does WRB say about its insurance offerings?
Response: No relevant documents found.

Processing query: What are the main risks WRB mentions?
Query: What are the main risks WRB mentions?
Response: No relevant documents found.

Processing query: What industries does IEX primarily serve?
Query: What industries does IEX primarily serve?
Response: According to the context, IDEX's products are sold to a "wide range of industries" throughout the world, but it doesn't
```

5.2 Streamlit Mock Function