Tomoro RAG System: Comprehensive Report

Mehdi Hosseini Moghadam

September 23, 2024

Contents

1 Project Overview

The Tomoro RAG System is a Retrieval-Augmented Generation (RAG) system designed to handle document embeddings, retrieval, and generation tasks. It uses FastAPI for the API layer and Poetry for dependency management. The system includes functionality for benchmarking and evaluation, making it suitable for various natural language processing (NLP) applications.

2 Setup Instructions

2.1 Installation

All installation steps and code can be found in the following Github repo

3 Project Structure

```
tomoro/
 .env.example            # Environment variables (not tracked in Git)
 .gitignore              # Git ignore file
 README.md               # Project documentation
 knowledge_graph.ipynb   # Knowledge GraphProject code
 pyproject.toml          # Poetry configuration and dependencies
 poetry.lock             # Lock file for Poetry dependencies
 openapi.json            # OpenAPI specification
 embeddings.db           # SQLite database for storing embeddings
 src/
    __init__.py
    main.py              # FastAPI application entry point
    config.py            # Configuration settings
    api/
       __init__.py
       routes.py         # API routes
       models.py         # Pydantic models for API
    core/
       __init__.py
       database.py       # Database operations
       embeddings.py     # Embedding-related functions
       rag.py            # RAG system implementation
       evaluation.py     # Evaluation logic

    data/
       __init__.py
       benchmarking.py   # Data for benchmarking the models
       embeddings.py     # Embedding-related functions
       train.json        # Training Data
```

```
    utils/
        __init__.py
        helpers.py      # Utility functions
tests/
    __init__.py
    test_api.py         # API tests
    test_rag.py         # RAG system tests
    test_evaluation.py # Evaluation tests
```

4 Algorithms

4.1 Hybrid Retrieval (Method 1)

4.1.1 Retrieval-Augmented Generation (RAG)

RAG is a hybrid approach that combines retrieval-based and generation-based methods for natural language processing tasks. The key idea is to retrieve relevant documents or passages from a large corpus and use them as context for generating responses. This approach leverages the strengths of both retrieval and generation, providing more accurate and contextually relevant answers.

4.1.2 Document Embeddings

- Document embeddings are vector representations of text documents. These embeddings capture the semantic meaning of the text, allowing for efficient similarity comparisons. In the Tomoro RAG System, embeddings are generated using a pre-trained text-embedding-ada-002 model from OpenAI. The embeddings are stored in a SQLite database for efficient retrieval.
- Since some documents are too long, the chucking method has been used to split text into smaller pieces. In this code chunk size of 500 with an overlap of 100 has been used
- Observations suggested that in order to have a better performance, it is helpful to add a table to each chunk (reason: some times retrieval manages to find the relative chunk but without its related table, on generation it is almost impossible to calculate the answer to the query)

4.1.3 BM25 Algorithm

BM25 (Best Matching 25) is a ranking function used by search engines to rank documents based on their relevance to a given query. It is a probabilistic information retrieval model that considers term frequency, document length, and inverse document frequency. In the Tomoro RAG System, BM25 is used to

create an index of the document corpus, enabling efficient retrieval of relevant documents.

4.1.4 Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors. It is calculated as the cosine of the angle between the vectors, ranging from -1 (completely dissimilar) to 1 (completely similar). In the Tomoro RAG System, cosine similarity is used to compare query embeddings with document embeddings, helping to identify the most relevant documents.

4.1.5 Hybrid Retrieval

Hybrid retrieval combines embedding-based similarity and BM25-based similarity to retrieve relevant documents. The Tomoro RAG System calculates both types of similarity scores and combines them to rank documents. This approach leverages the strengths of both methods, providing more accurate and relevant results (.7 weight to semantic embedding and .3 to BM25).

4.1.6 Reranking

Reranking is the process of reordering a list of retrieved documents based on additional criteria. In the Tomoro RAG System, reranking is performed using a language model gpt-4o-mini that evaluates the relevance of each document to the query. The top-ranked documents are then used as context for generating the final response.

4.1.7 Query Expansion

Query expansion involves generating alternative phrasings or related queries to improve retrieval performance. The Tomoro RAG System uses a language model gpt-4o-mini to generate expanded queries, increasing the chances of retrieving relevant documents.

4.1.8 Evaluation Metrics

The Tomoro RAG System includes functionality for benchmarking and evaluation. **RAGAS** library used for benchmarking and evaluation with following metrics:

- **Context Precision**: Context Precision is a metric that evaluates whether all of the ground-truth relevant items present in the contexts are ranked higher or not. Ideally all the relevant chunks must appear at the top ranks.
- **Faithfulness**: This measures the factual consistency of the generated answer against the given context. It is calculated from answer and retrieved context. The answer is scaled to (0,1) range. Higher the better.

- **Answer Relevancy**: The evaluation metric, Answer Relevancy, focuses on assessing how pertinent the generated answer is to the given prompt. A lower score is assigned to answers that are incomplete or contain redundant information and higher scores indicate better relevancy.
- **Context Recall**: Context recall measures the extent to which the retrieved context aligns with the annotated answer, treated as the ground truth. It is computed using question, ground truth and the retrieved context, and the values range between 0 and 1, with higher values indicating better performance.
- **Context utilization**: Context utilization is like a reference-free version of `context_precision` metrics. Context utilization is a metric that evaluates whether all of the answer relevant items present in the `contexts` are ranked higher or not. Ideally, all the relevant chunks must appear at the top ranks.
- **Answer Correctness**: The assessment of Answer Correctness involves gauging the accuracy of the generated answer when compared to the ground truth. This evaluation relies on the ground truth and the answer, with scores ranging from 0 to 1.

4.1.9 Evaluation Results for Hybrid Method

First 100 examples of the train.json data has been used for evaluation

Metric	Score
Context Precision	0.5649
Faithfulness	0.6660
Answer Relevancy	0.7194
Context Recall	0.4949
Context Utilization	0.6069
Answer Correctness	0.5820

Table 1: RAG Evaluation Metrics (Retrieve K = 10, Rerank K = 5)

Metric	Score
Context Precision	0.4999
Faithfulness	0.6698
Answer Relevancy	0.5870
Context Recall	0.5564
Context Utilization	0.7499
Answer Correctness	0.5211

Table 2: RAG Evaluation Metrics (Retrieve K = 5, Rerank K = 3)

Trade-offs between retrieval size and relevance: When the retrieval size (k) is larger (Scenario 1), the model has access to more potential context chunks, resulting in better answer relevancy and answer correctness, but the model might retrieve more irrelevant chunks, lowering context recall and utilization. When retrieval size is smaller (Scenario 2), context recall and utilization improve because the model has fewer irrelevant chunks to sort through, but at the cost of answer relevancy and correctness, as fewer context chunks are available to support the answer.

4.2 Knowledge Graph and Embedding (Method 2)

4.2.1 Knowledge Graph

In this section hybrid method of Knowledge Graph (Neo4j) and Embeddings will be explained. All related code and be found in `knowledge_graph.ipynb` jupyter notebook, this code is only for demonstration and unfortunately due to the lack of time evaluation and code refactor has not been done.

4.2.2 Why Knowledge Graph RAG?

- **Multi-Hop Reasoning**: In a simple RAG system, when you retrieve financial data (e.g., a company's quarterly report), you're limited to what was fetched. If the answer isn't directly in the retrieved text, the system won't be able to "connect the dots" between other related financial data. In a Knowledge Graph, financial data is connected through nodes and relationships. For example, if you ask about revenue in the latest quarterly report and follow up with a question about expenses, the system can easily retrieve the connected expenses node in the graph, even if it wasn't included in the initial context.

- **Contextual Understanding**: Financial data involves multiple interconnected concepts such as revenue, expenses, profit margins, assets, liabilities, and more. In a simple RAG, you might only retrieve a segment of data (e.g., a balance sheet), but a Knowledge Graph understands the relationships between these concepts and can explore those relationships as needed. For example, if you first ask about assets, then follow up with a question about liabilities, the Knowledge Graph can navigate to the correct node and return related data, while a traditional RAG would struggle to do so if it wasn't part of the initial retrieval.

- **Contextual Understanding**: In a simple RAG, you might retrieve a section of a company's annual report with revenue data, but other key sections like expenses, net income, or cash flow might not be included in the same retrieval. In a Knowledge Graph, all of these financial metrics are represented as nodes linked to the company, making it easy to dynamically retrieve interconnected information like cash flow or earnings per share

(EPS). This ensures that the system always has access to the complete set of financial data.

- **Handling Ambiguity**: A traditional RAG system may not be able to handle questions where the context changes, or where data needs to be synthesized from multiple sources. In a Knowledge Graph, if you ask a question about dividends and then follow up with a question about shareholder equity, the system can make multi-hop connections to related financial data nodes (e.g., from dividends to shareholder payouts to equity), helping the system provide an accurate and contextually relevant answer.

4.2.3 Data Ingestion and Storage

Financial data is stored in a Neo4j graph database. Each document is represented as a node with the following properties (below you can find the image of a single node):

- Document ID
- Pre-text content
- Post-text content
- Table content
- Pre-text embedding
- Post-text embedding

Algorithm 1 Financial Question Answering System

```
procedure QASystem(question)
    documents ← RetrieveDocumentsFromNeo4j()
    indexPre, indexPost ← IndexEmbeddingsBatched(documents)
    similarDocs ← FindSimilarDocuments(question, indexPre, indexPost, documents)
    topDocs ← ReRankDocumentsParallel(question, similarDocs)
    context ← PrepareContext(topDocs)
    answer ← AnswerQuestionWithContext(question, context)
    return context, answer
end procedure
```

4.3 Step-by-Step Explanation

4.3.1 Data Retrieval

The system begins by retrieving all relevant financial documents from the Neo4j database. Each document contains pre-text, post-text, table content, and their respective embeddings.

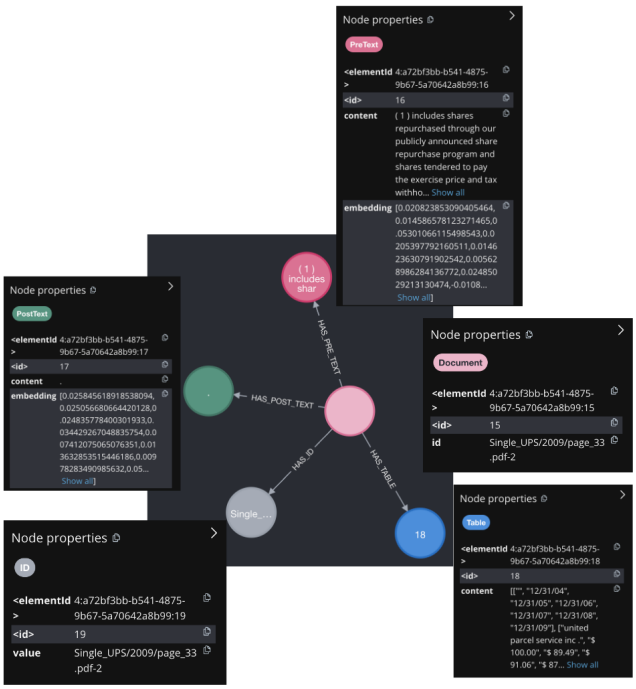

Figure 1: Example node in Neo4j knowledge graph

4.3.2 Embedding Indexing

The pre-text and post-text embeddings are indexed using FAISS. This process is done in batches to handle large datasets efficiently. The embeddings are normalized before indexing to ensure consistent similarity calculations.

4.3.3 Similarity Search

Given a question, the system generates an embedding for it and uses FAISS to find the most similar documents based on both pre-text and post-text embeddings. The results are combined and ranked based on their similarity scores.

4.3.4 Document Re-ranking

The top similar documents undergo a re-ranking process using a GPT-4o-mini model. This model assesses the relevance of each document to the given question on a scale of 0 to 10. This process is parallelized for efficiency.

4.3.5 Context Preparation

The system prepares a context for the question-answering phase by combining the content of the top-ranked documents. The context is limited to a maximum token count to ensure it fits within the model's input constraints.

4.3.6 Question Answering

Finally, the prepared context and the original question are passed to a GPT-4o-mini model. This model generates a comprehensive answer based on the provided information.