Design and Implementation of a Context Retrieval System for Automated Answer Generation

Seung Je Woo
Stanford University
sjwoo@stanford.edu

Original: April 2024
Updated: February 8, 2025

Abstract

Recent advancements in large language models (LLMs) have greatly enhanced automated answer generation systems. However, extracting the most relevant context from diverse documents—such as financial reports, legal submissions, training materials, and annual reports—remains challenging. In this work, we design and implement a retrieval system on a local instance that leverages vector-based search using Milvus to efficiently match embeddings generated from document texts and user queries. We describe our design decisions, including our choice of transformer models (e.g., BERT [2], MiniLM [5], MPNet [6], and Longformer [1]), as well as our data preprocessing, embedding generation, and search techniques. Experimental evaluations highlight the trade-offs involved and demonstrate that proper tuning can significantly improve context narrowing for automated answer generation.

1 Introduction

Large language models (LLMs) have transformed automated answer generation; however, significant challenges remain in retrieving the most relevant context from heterogeneous documents. In many applications, documents such as financial reports, legal submissions, training materials, and annual reports are originally available in PDF format and then converted into JSON for text processing. Extracting the pertinent segments that enable LLMs to generate accurate responses is nontrivial and has been the focus of recent research in dense retrieval and retrieval-augmented generation [3, 4].

In our work, we propose a context retrieval system that generates embeddings from both document texts and user queries using transformer models. These embeddings are stored in a vector database (Milvus [7]), and similarity search techniques are employed to retrieve the most relevant document segments. Our contributions are:

1. A robust system architecture that leverages state-of-the-art transformer models for embedding generation and employs Milvus for efficient similarity search.
2. A detailed experimental evaluation comparing transformer models, tokenization strategies, and hyperparameter settings to optimize the retrieval of context necessary for automated answer generation.

2 Methodology

2.1 System Overview

Our system follows a nine-step workflow. First, raw text extracted from PDFs is provided in JSON format. Next, a transformer model generates document embeddings from this text, which are then stored in the Milvus vector database. When a user submits a query, the same transformer model is used to convert the query into an embedding. The system searches Milvus for document embeddings most similar to the query embedding and retrieves the associated text. Finally, these retrieved contexts are provided as input to an LLM, which processes them to generate the final answer. Figure 1 illustrates this workflow.

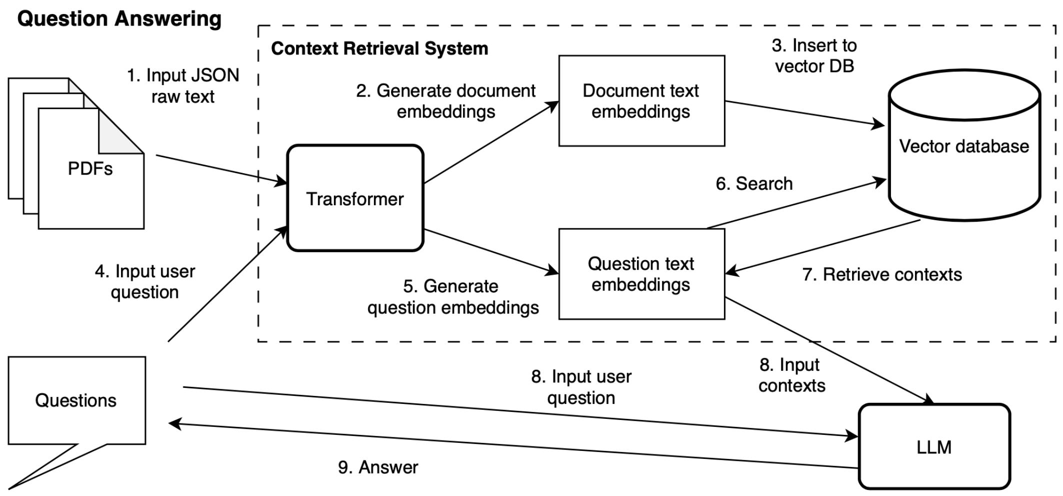

Figure 1: Workflow of the context retrieval system for automated answer generation. The process includes: (1) input raw JSON text, (2) generate document embeddings, (3) store embeddings in the vector database, (4) input user query, (5) generate query embedding, (6) search the database, (7) retrieve relevant contexts, (8) supply contexts to the LLM, and (9) generate the answer.

2.2 Design Considerations and Rationale

Our design is motivated by the need to efficiently handle high-dimensional vector data and ensure compatibility between document and query representations. We selected Milvus for its scalability, advanced indexing techniques, and seamless integration with Python, as described in [7]. The diverse documents used in our system—spanning financial, legal, educational, and annual reports—simulate real-world scenarios where document structures vary widely. Our evaluation queries, which include fact-based, semantic, and opinion-based types (as well as some irrelevant queries), further test the robustness of our system. Both document and query texts are converted into embeddings using the same transformer model, and similarity is computed via Euclidean distance and cosine similarity.

2.3 Embedding Generation and Similarity Search

We experiment with several transformer models:

- `bert-base-uncased`: Maximum sequence length of 512 tokens, 768 dimensions [2].
- `all-MiniLM-L6-v2`: Maximum sequence length of 256 tokens, 384 dimensions [5].
- `all-mpnet-base-v2`: Maximum sequence length of 384 tokens, 768 dimensions [6].
- `longformer-base-4096`: Maximum sequence length of 4096 tokens, 768 dimensions [1].

For longer documents, we employ a sliding window approach with overlapping token segments to preserve context. Similarity is measured using both Euclidean distance and cosine similarity, consistent with techniques in dense retrieval research [3, 4].

2.4 Milvus Collection Schema

Each document is stored in its own Milvus collection to optimize retrieval. Table 1 shows the schema, which includes an `id` (primary key), an `embeddings` field for the vector data, and a `text` field for qualitative analysis.

#	Field Name	Field Type	Attributes
1	`id`	VarChar	Primary key
2	`embeddings`	FloatVector	Indexed
3	`text`	VarChar	–

Table 1: Milvus Collection Schema

3 Implementation

3.1 Setup and Tools

We deploy a dockerized Milvus instance (see `https://milvus.io/docs/install_standalone-docker.md`) to store and index embeddings. For text preprocessing and embedding generation, we use transformer models from Hugging Face [2, 5, 1, 6] and the NLTK library. Raw document text is ingested from JSON files (located at `./raw_text`), and query data is read from a CSV file (converted from XLSX, located at `./questions/document_questions.csv`). Environment variables are stored in a `.env` file.

3.2 Embedding Generation and Storage

The `generate_embeddings` function cleans the text (e.g., lowercasing, HTML tag removal, whitespace cleaning), optionally tokenizes it into sentences, and uses a sliding window with overlap for long texts to generate multiple embeddings. These embeddings are normalized and stored in Milvus using the `store_data_to_milvus` function, which creates the collection as defined in Table 1.

3.3 Retrieval Process

Query embeddings are generated using the same process as for document embeddings, ensuring compatibility. The `search_embeddings` function queries Milvus with the query embedding and retrieves the top K most similar document embeddings using either cosine similarity or Euclidean distance.

3.4 Hyperparameter Tuning

We tune hyperparameters such as:

- **Model Selection:** Comparing transformer models (BERT, MiniLM, MPNet, Longformer) to balance semantic understanding with efficiency.
- **Tokenization Settings:** Adjusting maximum token lengths and overlap percentages to optimize context preservation.
- **Preprocessing Options:** Experimenting with text cleaning steps (e.g., stopword removal, lemmatization) to retain necessary context.
- **Similarity Metrics:** Evaluating cosine similarity versus Euclidean distance.

3.5 Data Labeling

In the absence of ground truth, we manually label a subset of embeddings. An embedding is marked as relevant if it (1) contains direct information necessary to answer the query, (2) provides related information that guides the reader, or (3) offers contextual clues that clarify the primary content.

4 Experiments and Results

4.1 Dataset Description

Our dataset comprises four documents, each representing a different document type:

- **Document A:** An educational/training document outlining tasks for learners related to scheduling and travel management. It contains tables and structured schedules.
- **Document B:** A financial report detailing a company's quarterly performance, including sales figures, earnings, and growth projections. It includes financial tables.
- **Document C:** An annual report summarizing leadership, membership, finances, and activities. It features structured meeting summaries and financial details.
- **Document D:** A legal submission discussing workplace issues and regulatory frameworks, including recommendations for policy improvements. It consists of structured legal arguments.

4.2 Evaluation Metrics

We evaluate our system using both quantitative and qualitative metrics:

- **Quantitative:** Similarity scores, retrieval accuracy (the fraction of relevant contexts within the top K), and search latency.
- **Qualitative:** Manual inspection of retrieved contexts compared with the original document content.

4.3 Results

While detailed sample results are provided in the supplementary material, we summarize key findings below.

4.3.1 Embedding Generation

Table 2 shows the number of embeddings generated per document by each transformer model. Notably, Longformer produces fewer embeddings due to its capacity to process up to 4096 tokens in a single pass.

Document	BERT	MiniLM	MPNet	Longformer
Document A	6	11	7	1
Document B	9	18	12	1
Document C	7	14	9	1
Document D	25	50	34	3

Table 2: Number of Embeddings per Document

4.3.2 Retrieval Accuracy

We define a custom scoring metric:

$$\text{Score} = \frac{1}{N_{\text{Ctx}}} \sum (\text{Ctx} \times \text{Distance}^{-2})$$

where Ctx = 1 for a relevant embedding and 0 otherwise. Figure 2 compares score values among the models (noting that differences in embedding dimensions may affect direct comparisons).

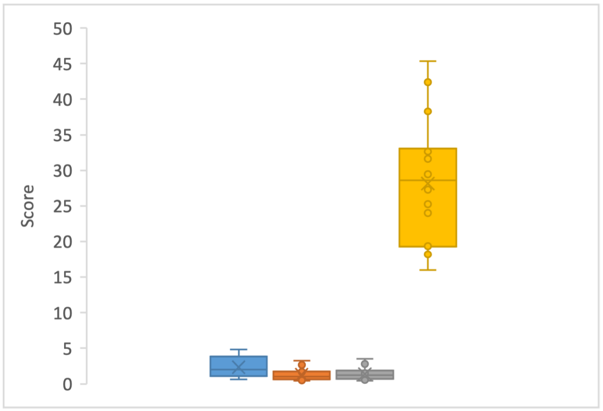

Figure 2: Score comparison among BERT, MiniLM, MPNet, and Longformer.

4.3.3 Latency

Table 3 presents the search latency (in seconds) for each model using both Euclidean (L2) and cosine similarity metrics. The differences are minimal, likely due to the modest dataset size.

Model	**L2 (s)**	**Cosine (s)**
BERT	0.27	0.26
MiniLM	0.27	0.29
MPNet	0.28	0.23
Longformer	0.25	0.23

Table 3: Search Latency Comparison (in seconds; standard error ≈ 0.02s)

4.3.4 Top K Context Retrieval

Table 4 shows the fraction of relevant contexts retrieved for different K values. Both MiniLM and MPNet perform strongly, with MPNet exhibiting a slight advantage.

	K			
Model	**1**	**3**	**5**	**7**
BERT	0.138	0.455	0.628	0.745
MiniLM	0.381	0.721	0.835	0.915
MPNet	0.333	0.754	0.898	0.915

Table 4: Top K Retrieval Accuracy for Different Models

4.3.5 Document-Level Retrieval

For overall document retrieval, all documents were combined into a single Milvus collection. A document is considered correctly identified if any of its embeddings is retrieved. Table 5 shows the document-level retrieval accuracy for $K = 5$ (Longformer is excluded because it typically produces only one embedding per document).

$K = 5$	BERT	MiniLM	MPNet
Accuracy	0.65 ± 0.07	0.96 ± 0.02	0.95 ± 0.02

Table 5: Document-Level Retrieval Accuracy with Top $K = 5$

5 Discussion

Our experiments indicate that while vanilla BERT performs reasonably well on text-centric documents, it struggles with documents containing additional structures such as tables and charts. MPNet, with its superior semantic understanding, is generally preferred when computational resources are sufficient, whereas MiniLM is a viable alternative in resource-constrained settings. The optimal K value for retrieval depends on the query; although a fixed K between 5 and 10 performs well in many cases, a dynamic approach may further enhance performance. Limitations include the small dataset size and the subjectivity introduced by manual labeling. Future work will involve expanding the dataset, developing dynamic query interpreters for adaptive K selection, and exploring additional transformer models and vector databases.

6 Conclusion

We have presented a comprehensive context retrieval system that extracts relevant information from diverse documents to support automated answer generation. By leveraging transformer-based embeddings and a scalable vector database (Milvus), our system effectively narrows down the context provided to an LLM, thereby enhancing answer accuracy. Our experimental evaluations comparing multiple transformer models and hyperparameter settings underscore the trade-offs involved in optimizing retrieval performance. Future work will extend this research through dynamic parameter tuning and evaluation on larger datasets.

References

[1] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. In *ACL*, 2020.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL-HLT*, 2019.

[3] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Wen-tau Yih, Zhuyun Chen, and Alan Yuille. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, 2020.

[4] Patrick Lewis, Ethan Perez, Adam Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Hamed Hassani, Sebastian Riedel, and Veselin Stoyanov. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020.

[5] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*, 2019.

[6] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding, 2020.

[7] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pages 2614–2627, New York, NY, USA, 2021. Association for Computing Machinery.