

Utilizing Class Assignment's Database For RAG(Retrieval augmented generation) Based Response Generation

Anurag Dhiman Department of Mathematics IIT Madras	Rahul Ghosh Department of Mathematics IIT Madras	Tanmoy Ghosh Department of Mathematics IIT Madras	Ajay Kumar Department of Mathematics IIT Madras
---	---	--	--

Abstract

In this project, we present a Retrieval-Augmented Generation (RAG) system designed to intelligently utilize a database of ten academic assignments for generating contextually grounded and accurate solutions. The system integrates principles of object-oriented programming with modern information retrieval and language modeling techniques. Each assignment is stored as structured text chunks within a vector database, enabling efficient semantic search through embedding-based retrieval. When a user submits a query, the system retrieves the most relevant assignment content and generates a coherent, fact-based response using a language model. The modular architecture—comprising classes for data storage, retrieval, and generation—ensures scalability, reusability, and maintainability. This project demonstrates how OOP principles can be effectively applied to implement RAG pipelines for educational and reasoning-based applications, reinforcing learning through retrieval-driven knowledge synthesis.

Keywords: Retrieval-Augmented Generation, Object-Oriented Programming, Semantic Search, Educational AI, Vector Database.

1. Introduction

Academic assignments are valuable resources containing detailed explanations, algorithms, and examples that reflect conceptual understanding. However, retrieving information from a collection of assignments is often tedious and inefficient. Traditional search methods rely on keyword matching, which fails to capture the semantic meaning of queries.

To overcome this limitation, this project proposes an *Assignment Retrieval-Augmented Generation (RAG)* system. The system combines semantic retrieval and generative modeling to deliver factually grounded responses using existing academic data. It is implemented using *Object-Oriented Programming (OOP)* principles, ensuring modularity, scalability, and maintainability. Each assignment is processed into structured text chunks and stored as embeddings in a vector database, supporting efficient semantic similarity search. The integration of retrieval and generation results in accurate, contextual, and educational responses.

2. Related Work

Traditional retrieval systems such as TF-IDF and BM25 are based on lexical overlap and fail to recognize semantic equivalence. Recent advances in representation learning, particularly *Sentence-BERT* and *Dense Passage Retrieval (DPR)*, have enabled semantic embeddings that capture contextual similarity.

Lewis et al. (2020) introduced the Retrieval-Augmented Generation (RAG) model, which merges dense retrieval with generative language modeling. This architecture forms the theoretical foundation for our system. While commercial models (e.g., GPT, Bard) use proprietary APIs, our implementation is fully open-source, employing *ChromaDB* for storage, *Sentence-Transformers* for embeddings, and *distilGPT2* for text generation.

3. System Architecture

3.1. Overview

The proposed RAG system consists of five primary components:

- **AssignmentPreprocessor:** Extracts and cleans text from .py and .ipynb files.
- **VectorDatabase:** Stores and retrieves document embeddings using ChromaDB.
- **Retriever:** Performs semantic search to identify the top-k most relevant text chunks.
- **Generator:** Produces contextually grounded responses using a transformer model.
- **RAGPipeline:** Integrates all modules and provides a single user-facing interface.

3.2. Data Flow

The workflow begins with assignment preprocessing, followed by embedding creation and database storage. When a user query is provided, it undergoes the same embedding transformation. The retriever computes semantic similarity between the query and stored embeddings, returning the most relevant chunks. The generator then synthesizes the retrieved information into a coherent and factual answer.

User Query → Retriever → Vector Database → Generator → Final Answer

This architecture ensures separation of concerns, encapsulation, and extendibility through OOP principles.

4. Methodology

4.1. Data Preprocessing

Each assignment file is read and processed using the nbformat library. Non-informative elements (metadata, code-only cells) are removed, while meaningful text (explanations, comments) is extracted. The data is segmented into smaller chunks for uniform embedding and retrieval.

4.2. Embedding Generation

The SentenceTransformer model (*all-MiniLM-L6-v2*) converts each text chunk c_i into a 384-dimensional dense embedding:

$$\mathbf{v}_i = f_\theta(c_i), \quad \mathbf{v}_i \in \mathbb{R}^{384} \quad (1)$$

where f_θ denotes the neural network parameterized by θ . These embeddings capture semantic meaning and facilitate efficient similarity comparison.

4.3. Retrieval and Similarity Computation

For a query q , the system computes its embedding $\mathbf{v}_q = f_\theta(q)$. Semantic similarity is measured using cosine similarity:

$$\text{sim}(\mathbf{v}_q, \mathbf{v}_i) = \frac{\mathbf{v}_q \cdot \mathbf{v}_i}{|\mathbf{v}_q| |\mathbf{v}_i|} \quad (2)$$

The retriever selects the top- k most relevant chunks, forming a context set $\mathcal{C}_k(q)$ for generation.

4.4. Answer Generation

The generator employs a transformer-based model (*distilGPT2*) to produce the final response. Given the query q and retrieved context $\mathcal{C}_k(q)$, the model predicts:

$$P_\phi(y|q, \mathcal{C}_k(q)) = \prod_{t=1}^T P_\phi(y_t|y_{<t}, q, \mathcal{C}_k(q)) \quad (3)$$

where P_ϕ represents the model's learned parameters.

5. Object-Oriented Design

This project adheres to fundamental OOP principles:

- **Encapsulation:** Each class contains its data and methods; internal logic such as embedding or retrieval operations is hidden.
- **Abstraction:** The `RAGPipeline.run()` method abstracts the underlying complexity from the user.
- **Composition:** The `RAGPipeline` integrates the Retriever and Generator modules.
- **Modularity:** Each component operates independently, facilitating maintenance and scalability.
- **Reusability:** Components can be reused for future RAG-based educational applications.

6. Implementation Details

6.1. Tools and Technologies

- **Language:** Python 3.10
- **Libraries:** chromadb, sentence-transformers, transformers, torch, nbformat
- **Models:** SentenceTransformer (for embeddings), distilGPT2 (for generation)

6.2. Dataset

The dataset comprises ten academic assignments. Each assignment is processed into smaller text chunks (50–150 words), resulting in approximately 300 stored segments.

7. Results and Discussion

The system was evaluated by testing queries on the stored assignment dataset. Representative examples include:

Table 1: Sample Query and Generated Responses

Query	Retrieved Context	Generated Answer
Quick sort partitions the array. Average case $O(n \log n)$, worst case $O(n^2)$.	Quick sort has $O(n \log n)$ average and $O(n^2)$ worst case complexity.	What is the time complexity of quick sort?
Breadth-first search uses a queue for traversal.	What data structure does BFS use?	Breadth-first search uses queue, while DFS uses stack.

The system effectively retrieves relevant assignment segments and generates factually accurate and semantically coherent responses. Results demonstrate that retrieval-based grounding significantly improves factual correctness compared to isolated text generation.

The integration of OOP ensures maintainability and clarity, while modular design allows future scaling (e.g., adding new assignments or updating models) without code restructuring.

8. Conclusion and Future Work

The *Assignment RAG System* demonstrates the practical synergy between Object-Oriented Programming and Retrieval-Augmented Generation. It provides a structured, reusable, and open-source framework for creating educational AI assistants that reason over academic datasets. The system successfully generates accurate, context-aware solutions derived from real assignments.

Future work includes extending the dataset to include additional course materials (e.g., lecture notes and PDFs), integrating reinforcement learning from user feedback, and developing an interactive graphical user interface. The framework can also be adapted for domain-specific retrieval systems such as legal, medical, or technical document repositories.

References

1. Lewis, P., et al. (2020). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. ACL.
2. Reimers, N., Gurevych, I. (2019). *Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks*. EMNLP.
3. ChromaDB Documentation. (2024). <https://docs.trychroma.com>
4. Hugging Face Transformers. (2024). <https://huggingface.co/docs/transformers>