Intelligent Knowledge Base Using Retrieval-Augmented Generation (RAG)

David Elgas

CSCI E-104 Advanced Deep Learning, 2025

Harvard University Extension School

# Abstract

Problem and Solution

Large language models (LLMs) like GPT-3.5 are powerful, but they are limited by the static training data they were exposed to. As a result, they often underperform in specialized, high-knowledge domains such as aerospace, biotech, or industrial diagnostics—domains where context evolves, and the latest knowledge isn't part of a frozen training set. To resolve these issues, supplemental information can be added to aid in knowledge transfer. This process is known as Retrieval-Augmented Generation (RAG).

Dataset and Technology

The primary dataset for this effort was built by leveraging Python with Beautiful Soup to gather information from a publicly available online forum, The data was processed through SentenceTransformer embeddings, FAISS vector search, and OpenAI GPT-3.5, ultimately served in a Streamlit web interface allowing users to explore the results.

Uses and Benefits

This project demonstrates how enterprises can enhance commercial LLMs with domain-specific knowledge. By implementing RAG, organizations can extend capabilities into specialized domains without retraining, improve response accuracy for technical queries, keep knowledge current by updating the corpus independently, preserve institutional knowledge, and reduce hallucinations by grounding responses in verified sources.

Drawbacks

RAG systems face several limitations: response quality depends on corpus accuracy, potentially propagating misinformation; semantic matching limitations when terminology differs between queries and sources; higher computational costs for both vector database management and commercial LLM APIs; increased architectural complexity compared to standard LLM implementations; and inheritance of biases present in the original community discussions.

Challenges

The initial challenge with this effort was Beautiful Soup programming specific to the forum platform (XenForo 2.3). Additionally, attempts to train a custom RAG model produced unusable results. Migrating to a semantic model combining vector database with a commercial generator yielded better outcomes. Finally, creating an inexpensive, ephemeral model proved difficult as commercial ML hosting solutions struggled with the implementation.

Results and Impact

The results show that the supplemental RAG corpus contained a richness of data not found in the LLM. This satisfies the original use case for this effort.

A close-up of a page

Description automatically generated

*\* Note that the OpenAI response references an airbag which is not a technology available in these older automobiles.*

Demo Resources

YouTube: https://www.youtube.com/watch?v=Vf9RgCn8ow0

Table of Contents

[Abstract 2](#_Toc198053390)

[1. Problem Statement 4](#_Toc198053391)

[1. The RAG pipeline 5](#_Toc198053392)

[1.1. The Corpus 5](#_Toc198053393)

[1.2. Embeddings and Similarity Search 5](#_Toc198053394)

[1.3. The Generator 6](#_Toc198053395)

[1.4. The Application 6](#_Toc198053396)

[2. Environments, languages and libraries 7](#_Toc198053397)

[3. Installation and configuration 8](#_Toc198053398)

[3.1. Corpus 9](#_Toc198053399)

[3.2. RAG Embedding 10](#_Toc198053400)

[3.3. Streamlit Application 11](#_Toc198053401)

[4. Results and Observations 13](#_Toc198053402)

[5. What Worked / What Didn’t 14](#_Toc198053403)

[6. Future Enhancements 14](#_Toc198053404)

[7. Documentation 14](#_Toc198053405)

# 

# Problem Statement

Large language models (LLMs) like GPT-3.5 are powerful, but they are limited by the static training data they were exposed to. As a result, they often underperform in specialized, high-knowledge domains such as aerospace, biotech, or industrial diagnostics—domains where context evolves, and the latest knowledge isn’t part of a frozen training set.

This project investigates the application of Retrieval-Augmented Generation (RAG) as a technique to mitigate these limitations. RAG supplements a general-purpose LLM with a domain-specific corpus—indexed for retrieval—so that user queries are answered using the most contextually relevant passages available.

A diagram of a flowchart

Description automatically generated

Figure High level RAG pipeline

To demonstrate this technique, we use a real-world corpus scraped from a niche technical forum. While the domain chosen for testing happens to be classic car maintenance (BMW 3.0 series), the focus is squarely on illustrating how RAG pipelines can dramatically improve LLM performance in any narrowly scoped, content-rich vertical.

The RAG pipeline for this project is divided into the modules described below:

1. Corpus Development
2. Corpus Embedding
3. RAG Application

A diagram of a diagram

Description automatically generated

Figure Project Modules

# The RAG pipeline

## The Corpus

This project created a supplemental domain-specific corpus designed to enhance or replace the background knowledge of a general-purpose LLM. In this case, the corpus was constructed from a public technical forum built on XenForo 2.3. Although the forum content focuses on the maintenance of BMW 3.0 automobiles, the intent was to test RAG’s ability to operate in any unstructured, high-signal environment.

The forum used in this project was an HTML based forum generated by PHP. The resulting content structure was parsed into the required data with the program Beautiful Soup. This allowed for the extraction of the following content:

* Post ID: Used as the primary key of unique values per forum post
* Post Title: The title of the post by the author
* First Post: The inquiry of the author.
* All Posts: All responses by forum members

This resulted in a corpus of ~15k questions and ~100k answers.

## Embeddings and Similarity Search

To create a searchable knowledge base, each forum thread was first embedded into a 384-dimensional vector space using the MiniLM Sentence Transformer. This PyTorch-based model from the sentence-transformers library captures the semantic essence of text, enabling similarity comparisons beyond simple keyword matching.

|  |  |
| --- | --- |
| A blue and white speckled object  Description automatically generated |  |

Figure 3 High dimensional vector space reduced with TSNE for visualization and data exploration.

Each corpus entry is shown as both a data point (left image) and the underlying text values (right image).

FAISS (Facebook AI Similarity Search), an open-source library developed by Meta, was then employed to efficiently index and retrieve these dense vector embeddings. Using the IndexFlatL2 algorithm with Euclidean distance metrics, FAISS provides rapid similarity search capabilities that scale to large document collections. This implementation allowed the system to quickly identify and retrieve the top-k most semantically relevant documents for any given user query, forming the retrieval foundation of the RAG pipeline.

## The Generator

The RAG pipeline uses a generator model to synthesize retrieved information into coherent responses. This project employed OpenAI's GPT-3.5-Turbo for its balance of reasoning capabilities and cost-effectiveness.

When processing a query, the system feeds the most relevant forum threads from FAISS into a prompt template that instructs GPT-3.5 to consider only the provided context, directly address the question, and maintain technical accuracy. This approach leverages the model's language understanding while constraining its outputs to the domain-specific knowledge in the retrieved documents.

The result is a system that combines the fluency and natural response generation of a large language model with the factual grounding of the specialized corpus, producing contextually appropriate answers that reflect the technical expertise found in the original forum discussions.

## The Application

To validate the RAG system's real-world usability, a minimal user-facing application was built using Streamlit. The goal was to provide a simple, intuitive interface for submitting natural language queries and reviewing LLM-generated answers with and without RAG context.

Features:

* Single input box for entering a question
* Side-by-side comparison: one for the LLM response, one for the RAG-enhanced response
* Clear labeling of which result was derived from retrieval augmentation
* Linked information sources
* Temperature tuning
* Selection of top-k documents

A screenshot of a computer

Description automatically generated

Figure Streamlit app results

# Environments, languages and libraries

* 1. Corpus
* **Features:** The corpus system extracts and structures forum data from a XenForo 2.3-based website. It supports batch scraping, resilient error handling, incremental updates via thread IDs, and dual-format exports (CSV and JSON) and dual-storage formats (local CSV and cloud database via Snowflake).
* **Environment:** Development and execution were performed in Google Colab (CPU instance), leveraging Google Drive for persistent file storage and sharing.
* **Languages:** The scraping, parsing, and processing logic was implemented entirely in Python 3.10 using the library Beautiful Soup, with optional SQL used for schema definition and corpus upload validation in Snowflake.
* **Libraries:** Key Python libraries include:
  + BeautifulSoup: for HTML parsing
  + pandas: for DataFrame manipulation
  + requests: for HTTP thread retrieval
  + json and os: for formatting and file operations
  + snowflake.connector: for database interaction
  1. Embeddings
* **Features:** Transforms forum threads and user queries into dense semantic vectors using a lightweight transformer model (MiniLM-L6-v2). These vectors are stored in a FAISS index for similarity-based retrieval. The retrieved context is then passed into a foundation model (GPT-3.5) for natural language generation.
* **Environment:** Development and execution were performed in Google Colab (CPU instance), leveraging Google Drive for persistent file storage and sharing.
* **Languages:** Python 3.10
* **Libraries:**
  + sentence-transformers (MiniLM for embeddings)
  + faiss-cpu (vector similarity search)
  + openai (GPT-3.5 interface)
  + numpy (vector manipulation and search utilities)
  1. Application
* **Features:** Provides a lightweight Streamlit-based user interface for querying the corpus and comparing baseline vs. RAG-enhanced responses in real time. The app loads a pre-computed FAISS index and associated PKL-formatted corpus file. This design enables rapid retrieval and generation without reprocessing at runtime.
* **Environment:** Deployed on Streamlit Cloud via Github for public access and demonstration. Local development and testing were performed in Google Colab.
* **Languages:** Python 3.10
* **Libraries:**
  + streamlit: for UI development
  + openai: for calling GPT-3.5 completions
  + faiss: to load and query the vector index
  + pickle: for loading serialized document data
  + sentence-transformers: for query-time embedding

# Installation and configuration

Notebooks, data and application files are stored as follows. All files are sharable to anyone with the links.

**Folder Structure**

Week\_Project/

└── Notebooks/

├── LLM\_RAG\_ELGASDAVID\_Corpus.ipynb # The corpus notebook

├── LLM\_RAG\_ELGASDAVID\_RAG.ipynb # The embedding notebook

├── LLM\_RAG\_ELGASDAVID\_App.ipynb # App creation notebook

└── Datasets/

├── e9\_forum\_corpus.csv # Complete corpus

├── e9\_forum\_corpus\_batch\_1.csv # Batch corpus file

├── e9\_forum\_posts.csv # Raw post data

└── App/

├── LLM\_RAG\_App.py # Main Streamlit application

├── LLM\_RAG\_index.faiss # Vector database for semantic search

├── LLM\_RAG\_reqs.txt # Dependencies list

└── LLM\_RAG\_threads.pkl # Serialized thread data

## Corpus

**Overview**

This notebook scrapes the BMW E9 Coupe forum (e9coupe.com) to create a comprehensive knowledge base for RAG applications. It extracts thread titles, first posts, and all replies.

**Requirements**

* Google Colab environment
* Google Drive mounted to Colab (/content/drive)
* Python libraries:
  + BeautifulSoup4 (HTML parsing)
  + pandas (data manipulation)
  + requests (HTTP requests)
  + concurrent.futures (parallel processing)
  + snowflake.connector (optional - can be removed)
  + json (data formatting)
  + os, time, datetime (system utilities)

**File Access**

* All folders and files have been granted public access through Google Drive sharing settings

**Setup Instructions**

1. Open the LLM\_RAG\_ELGASDAVID\_Corpus.ipynb notebook in Google Colab
2. Run the first cell to mount your Google Drive and install dependencies
3. The BASE\_PATH variable is set to '/content/drive/Othercomputers/My Mac/CSCI\_104/Week\_Project/Datasets/' - it is recommended to leave this as is to maintain compatibility with the existing file structure

**Configuration Variables**

Adjust these values in the "Orchestration" section as needed

* NUM\_BATCHES: Number of batches to run (default: 1)
* THREADS\_PER\_BATCH: Threads to scrape per batch (default: 10)
* MAX\_WORKERS: Number of concurrent threads (default: 3)

**Running the Orchestration**

1. Runs all cells in sequence
2. The Orchestrator will:
   * Create/update thread ID tracking file
   * Fetch forum threads in batches
   * Parse HTML content using BeautifulSoup
   * Save data to CSV files
   * Build a complete corpus file

**Data Flow**

1. **Collection**: Forum thread IDs are tracked in e9\_forum\_thread\_ids.csv
2. **Scraping**: Raw HTML is fetched and processed into e9\_forum\_posts.csv and e9\_forum\_threads\_decorated.csv
3. **Aggregation**: Complete corpus is built in e9\_forum\_corpus.csv

**Synchronization**

Run the "Code for ensuring that all files are correctly synched" cell to:

* Ensure all tracking files are up-to-date
* Create backups of important files
* Repair any inconsistencies between files

## RAG Embedding

**Overview**

This notebook implements Retrieval-Augmented Generation (RAG) for BMW E9 technical information. It combines a domain-specific knowledge base from forum data with a language model to provide accurate, context-enriched responses to technical questions.

**Requirements**

* Google Colab environment
* Google Drive mounted to Colab (/content/drive)
* OpenAI API key (users must supply their own)
* Python libraries:
  + sentence-transformers (embedding generation)
  + faiss-cpu (vector similarity search)
  + pandas (data manipulation)
  + openai (API access to GPT models)
  + pickle (data serialization)
  + matplotlib (visualization)

**File Access**

* **All folders and files have been granted public access** through Google Drive sharing settings

**Setup Instructions**

1. Open the LLM\_RAG\_ELGASDAVID\_RAG\_FINAL.ipynb notebook in Google Colab
2. Run the first cell to mount your Google Drive and install dependencies
3. **Important**: You must provide your own OpenAI API key.

**Data Flow**

1. **Load Corpus**: The preprocessed forum data is loaded from a public Google Drive URL
2. **Embedding**: Text is transformed into vector embeddings using MiniLM Sentence Transformer
3. **Retrieval**: FAISS vector search finds the most semantically similar forum posts to a query
4. **Generation**: Retrieved context is fed to GPT-3.5-Turbo to generate informed responses
5. **Visualization**: Optional data visualization to analyze semantic retrieval distances
6. **Application files**: The App folder is loaded with the pre-processed FAISS index and PKL files

**Using the RAG System**

1. The notebook contains an example of a question about BMW E9 maintenance
2. It shows both standard LLM responses and RAG-enhanced responses for comparison
3. The core output is a FAISS index and thread data that powers the Streamlit application

**Output Files**

The notebook generates the following files used later in the Streamlit app. These files enable the Streamlit application to provide real-time responses without needing to recreate the embeddings for each query, resulting in a more efficient user experience.

* Vector index (FAISS format) stored at /App/LLM\_RAG\_index.faiss
* Thread data (pickle format) stored at /App/LLM\_RAG\_threads.pkl

## Streamlit Application

**Overview**

This system implements a comparison framework between Retrieval-Augmented Generation (RAG) and direct OpenAI responses. It demonstrates how domain-specific knowledge enhances LLM outputs compared to relying solely on the model's built-in knowledge.

**Requirements**

* GitHub account for Streamlit integration
* Streamlit Cloud account
* OpenAI API key (users must supply their own)
* Previously generated FAISS index and thread data PKL files

**Setup Instructions**

1. Open the LLM\_RAG\_ELGASDAVID\_App.ipynb notebook in your preferred environment
2. Run the cells to generate the application files

**Output Files**

The notebook generates:

* Streamlit application (RAG\_vs\_OpenAI\_App.py)
* Requirements file (requirements.txt)

**Note:** *Steamlit requires the loading of the main .py file via Github.*

1. Create a new GitHub repository
2. Upload RAG\_vs\_OpenAI\_App.py and requirements.txt to the repository
3. Go to Streamlit Cloud and connect your GitHub account
4. Select your repository and load the FAISS and PKL files.
5. **Important**: You must provide your own OpenAI API key
6. Deploy the application

Key screens in Stramlit.io

A screenshot of a computer

Description automatically generated

Figure Loading the main .py file from Github

A screenshot of a computer

Description automatically generated

Figure Loading the OpenAI API Key

A screenshot of a computer

Description automatically generated

Figure Loading FAISS and PKL files

# Results and Observations

The L2 histogram suggests the model for the top 1,000 documents shows marginal results with most documents having a high L2 distribution.

A graph of a graph

Description automatically generated with medium confidence

Despite these somewhat marginal results, a qualitative analysis of the results suggests that the RAG did outperform the pure OpenAI response.

A screenshot of a computer

Description automatically generated

Figure Full application output

# What Worked / What Didn’t

**What Worked:**

* FAISS provided high-quality semantic retrieval using MiniLM embeddings.
* Streamlit was effective for rapid prototyping and side-by-side comparison.
* The modular architecture allowed isolated testing and easy reuse of components.

**What Didn’t:**

* Early over-cleaning of the corpus (e.g., lemmatization, regex stripping) removed too much semantic signal.
* Snowflake insertions failed when datatypes weren’t explicitly cast.
* Hugging Face Spaces could not resolve deep Python package compatibility issues
* Source files are large, even for this limited model. Adequate ephemeral storage was an added complexity.

# Future Enhancements

The L2 distribution suggest there is additional work that can be done to improve performance. Lemmetization and additional corpus cleaning may improve these results.

While the rich targeted data offered by RAG was useful, additional value to end users could be added if images and video were included in the results. This will require significant additional code development to add this metadata.

# Documentation

* Application

<https://llmragapppy-j3wf5lc4xcu9bp9ysfovdd.streamlit.app/>

* Code and Data Repository:

<https://drive.google.com/drive/folders/1ypcdgxH8DRXSxoa5Ni3Ar-uBJe5xscM1?usp=drive_link>

* YouTube Recordings:

YouTube 2min: <https://youtu.be/Vf9RgCn8ow0>

YouTube 15m**:** <https://youtu.be/xAFkszSArU8>