Content Engine Documentation

Aman Vishwakarma

November 14, 2024

1 Methods Tried and Evaluation

1.1 Large Language Models (LLMs)

We evaluated several LLMs for our task, each providing different levels of performance and capabilities.

- **GPT-3:** A state-of-the-art transformer-based model that excels in generating text and answering queries but has limitations in understanding specific domain knowledge.
- **GPT-4:** Chosen for its superior ability to handle complex queries and generate coherent responses.
- **BERT:** Primarily used for tasks that require contextual understanding, but its generation capabilities are limited compared to GPT-based models.
- **RoBERTa:** A robust model designed to improve upon BERT, though it did not outperform GPT models for our specific task.
- **T5:** A versatile transformer model that performs well in text generation tasks, but GPT-4 still showed better overall performance for our use case.

The results indicated that GPT-4 outperformed other models in terms of generating relevant and coherent responses for semantic search.

1.2 Vector Databases

We explored different vector databases to store and retrieve the embeddings generated from the text.

- **FAISS:** An open-source library that provides efficient similarity search and clustering. It was chosen due to its flexibility and scalability.
- **Pinecone:** A managed service that offers vector search capabilities but was less appealing due to cost and limited customization compared to FAISS.
- **Milvus:** A scalable vector database suitable for large datasets, but FAISS was preferred for its simplicity and ease of integration.

- **Weaviate:** A highly customizable open-source vector search engine, but FAISS offered more efficient indexing and retrieval for our dataset.

FAISS was selected due to its open-source nature, scalability, and high performance in similarity search.

1.3 Embedding Techniques

We evaluated several embedding techniques to convert textual data into vector representations.

- **TF-IDF:** A simple and effective method for text representation, but it struggled with capturing semantic meaning compared to newer techniques.
- **Word2Vec:** Captures word semantics, but does not handle sentence-level context effectively.
- **Sentence-BERT:** An advanced embedding technique that uses BERT models fine-tuned for sentence-level embeddings. It performed best in capturing semantic context and was chosen for this project.
- **OpenAI Embeddings:** A commercial API offering state-of-the-art embeddings, but Sentence-BERT offered comparable performance with more control and no ongoing costs.

Sentence-BERT was selected as the final embedding technique due to its high-quality sentence embeddings that better captured the meaning and context of queries and documents.

2 Final Approach

2.1 Chosen LLM

We chose **meta/llama-2-70b-chat** due to its superior ability to generate meaningful and contextually accurate responses. meta/llama-2-70b-chat outperformed other models like BERT and RoBERTa, especially in handling complex queries.

2.2 Chosen Vector Database

We selected **FAISS** as the vector database due to its open-source nature, scalability, and efficient similarity search capabilities. FAISS provided the best performance for handling the embeddings generated from Sentence-BERT.

2.3 Chosen Embedding Technique

We decided to use **Sentence-BERT** for generating sentence embeddings, as it provided the most accurate semantic representation of text. It excelled in capturing the contextual meaning of sentences and performed better than traditional methods like TF-IDF.

3 Utilization of Open-Source Tools

Throughout this project, we leveraged a variety of open-source tools and libraries:

- **Hugging Face Transformers:** Used for accessing and fine-tuning GPT-4 and Sentence-BERT models.
- **FAISS:** Utilized for storing and querying the embeddings generated by Sentence-BERT.
- **Sentence-Transformers:** A key library for generating high-quality sentence embeddings using pre-trained models like Sentence-BERT.

The use of open-source tools reduced costs, provided flexibility, and allowed for easy integration with existing frameworks.

4 Challenges Faced and Solutions

4.1 Technical Challenges

- **Scalability:** The initial choice of database could not handle large-scale queries efficiently. We switched to FAISS for better performance.
- **Embedding Quality:** Early embeddings using TF-IDF did not capture semantic meaning well. Switching to Sentence-BERT improved the accuracy significantly.

4.2 Solutions Implemented

We addressed scalability by opting for FAISS and improved embedding quality by using Sentence-BERT. The combination of meta/llama-2-70b-chat, FAISS, and Sentence-BERT created a powerful system for semantic search.

5 Conclusion

In conclusion, by experimenting with different LLMs, vector databases, and embedding techniques, we found that meta/llama-2-70b-chat, FAISS, and Sentence-BERT provided the best performance for building a semantic search system. The use of open-source tools significantly reduced costs and allowed for greater flexibility in customizing the solution.

5.1 Future Improvements

Future work could involve further fine-tuning the models for domain-specific queries or exploring newer vector databases for scalability and performance.

6 User Guide

To run the application, follow these steps:

1. **Clone the repository:**
   Clone the project repository from GitHub using the following command:

   ```
   git clone the repository
   ```

2. **Create a virtual environment:**
   Navigate to the project directory and create a virtual environment to isolate the dependencies:

   ```
   python -m venv venv
   ```

3. **Install dependencies:**
   Install the necessary dependencies listed in the `requirements.txt` file using pip:

   ```
   pip install -r requirements.txt
   ```

4. **Run the Streamlit app:**
   Start the application by running the following command:

   ```
   streamlit run app.py
   ```

   This will open the app in your web browser, and you can interact with it.