Documentation

Data Preparation

10 Medical Guidelines

Ten medical guidelines were manually created, each targeting a specific disease (e.g., diabetes, asthma, breast cancer). Each guideline provides essential diagnostic, treatment, and management instructions.

Example:
For diabetes:

- **Title**: Diabetes
- **Content**: Details about symptoms, diagnostic criteria, and treatment protocols.

These guidelines are converted into structured data formats for further processing.

100 JSON Datasets

A dataset of 100 JSON files is generated, with each entry simulating patient-specific questions and answers. The JSON format includes:

- **id**
- **Category**
- **question**
- **options**
- **correct_answer**
- **Reasoning**

Example JSON Structure:

```
{"id": 0,
  "Category": "First Aid",
   "question": "What does the 'D' in DRSABCD stand for in first aid?",
   "options": {
     "A": "Defibrillator",
     "B": "Danger",
     "C": "Diagnosis",
     "D": "Decision-making"
   },
   "correct_answer": "B",
   "Reasoning": "The 'D' in DRSABCD stands for Danger, which involves checking for danger to you, any bystanders, and the injured person before providing assistance."
 }
```

Indexing

The indexing process involves preparing the medical guidelines and datasets for efficient querying. Key steps include:

Document Conversion

Medical guidelines in PDF format are converted into structured documents using a PDF loader. This ensures that the content is accessible for further processing.

Text Splitting

To optimize the documents for embedding, they are split into smaller chunks. Each chunk is designed to be the right size for processing by the embedding model, ensuring overlap for context retention.

Embedding and Storage

The split documents are embedded into vector representations using OpenAI's embedding model. These embeddings are then stored in a ChromaDB vector database. This setup allows for fast similarity searches, enabling efficient retrieval of relevant information during queries.

Retrieval & Generation

Input Query

Users can input questions, which are tokenized and converted into embeddings for similarity matching against the indexed guidelines.

Response Generation

The top matching documents are retrieved, and a concise, evidence-based response is generated using LangChain's language model chain.

RAG-Fusion + Reciprocal Ranking

1. **Generate Sub-Queries**:
- Use an LLM (Large Language Model) to decompose the original query into multiple focused sub-queries.
2. **Retrieve Relevant Documents**:
- For each sub-query, retrieve relevant documents from the knowledge base or search engine.
3. **Rank Relevant Documents**:
- Rank the retrieved documents based on relevance using a ranking algorithm.

4. **Generate Final Answer**:
- Use the ranked documents as context and the original query as input to the LLM to generate the final answer.

Evaluation

- **Options**: High precision and recall values indicate the model performs well in predicting correct answers with minimal false positives or false negatives.

  Confusion Matrix:
  `[[19  0  2  0]` (19 out of 21 instances of class "A" were correctly predicted as "A")
  `[ 0 53  0  0]` (53 out of 53 instances of class "B" were correctly predicted as "B")
  `[ 0  1 17  0]` (17 out of 18 instances of class "C" were correctly predicted as "C")
  `[ 0  0  1  7]]` (7 out of 8 instances of class "D" were correctly predicted as "D")

  Precision: 0.9579
  Recall: 0.9311

- **Reasoning**: ROUGE metrics show a tradeoff between precision and recall. The higher recall suggests the model captures relevant reasoning, but the lower precision and F1 scores indicate room for improvement in concise, accurate reasoning generation.

  ROUGE-1: Precision: 0.3117, Recall: 0.7881, F1: 0.4396
  ROUGE-2: Precision: 0.1928, Recall: 0.5026, F1: 0.2740
  ROUGE-L: Precision: 0.2597, Recall: 0.6582, F1: 0.3664

  Reasoning evaluation would be more accurate with LLM-based context validation, but due to credit constraints, ROUGE metrics are used as a proxy for reasoning quality.
