Help Website Q&A Agent

GitHub - https://github.com/reyanalam/AI-powered-question-answering-agent.git (Private , only access to vatsal@pulsegen.io)

Video - https://drive.google.com/file/d/1GodG3DrcDPRtFCCYmCaFbvQjxz6cMYeo/view?usp=sharing

This project is an AI-powered question-answering system that processes help website documentation and answers user queries related to product features, integrations, and functionalities.

Technical Architecture Overview

The system architecture follows a modular pipeline, ensuring clarity, extensibility, and ease of debugging. The major components include:

1. Web Crawler

- **Purpose**: Extracts meaningful and structured data from static HTML content on help websites.
- **Technologies Used**:
  - `requests` – for HTTP requests
  - `beautifulsoup4` – for parsing and extracting HTML content
  - `urllib3`, `tldextract` – for robust URL handling
- **Features**:
  - Recursive crawling with customizable depth
  - Filters out irrelevant links and external domains
  - Collects paragraphs, lists, tables, and headers

2. Preprocessing Module

- **Purpose**: Cleans and tokenizes the content for further semantic analysis.
- **Technology Used**:
    - `spaCy` with the large English model
- **Steps**:
    - Stop word removal
    - Lemmatization

3. Chunking Engine

- **Purpose**: Segments text into semantically similar chunks.
- **Approach**:
    - Uses cosine similarity between token vectors
    - Groups tokens until similarity drops below a defined threshold
    - Ensures each chunk is meaningful and self-contained

4. Semantic Search

- **Purpose**: Embeds user queries and document chunks into vector space for comparison.
- **Approach**:
    - Averages spaCy token vectors to represent full queries and chunks
    - Computes cosine similarity between the query vector and each chunk
    - Returns top 3 most relevant chunks

Implementation Approach

1. **Input Handling**: User provides the base URL of a help website (e.g., https://help.zluri.com/).

2. **Crawling**: Internal pages are crawled recursively up to a user-defined depth.

3. **Extraction & Cleaning**: HTML content is parsed and non-relevant tags are discarded.

4. **Text Preprocessing**: The clean text is tokenized, lemmatized using spaCy.

5. **Chunking**: Text is chunked based on semantic cohesion measured by cosine similarity.

6. **Vector Embedding & Search**: Both chunks and queries are embedded and compared using cosine similarity.

7. **Answer Retrieval**: Top 3 matching chunks are returned as answers to the user query.

Testing Approach

- **Unit Testing**:
  - Core modules like web crawler, chunker, and vector generator are tested individually.
  - `test.py` script verifies overall functionality using a sample URL.
- **Manual Verification**:
  - Compare returned answers with original documentation content to verify relevance and accuracy.

Future Improvement Suggestions

1. **Enhanced Chunking**
   - Replace custom chunking with LangChain or other NLP-based segmenters for improved semantic segmentation.

2. **Transformer-based Embeddings**

    - Use models like BERT, RoBERTa, or SentenceTransformers for contextual embeddings to boost answer quality.

3. **Improved Search Algorithms**
    - Implement FAISS or ElasticSearch for scalable and faster vector-based search over large corpora.
4. **Support for Dynamic Content**
    - Integrate Selenium or Playwright to scrape JavaScript-heavy websites.
5. **Web Interface**
    - Add a frontend UI for better user interaction instead of terminal-based input.