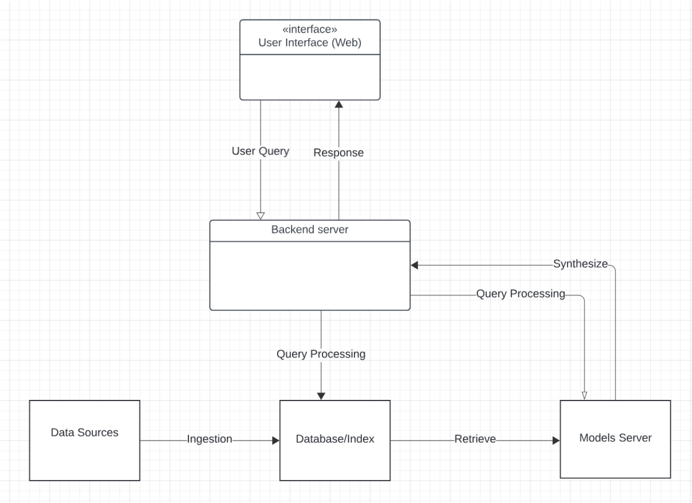
Candidate's Presentation

Minh Kha

04/01/2024

Architecture Overview: QA System for Field Engineers

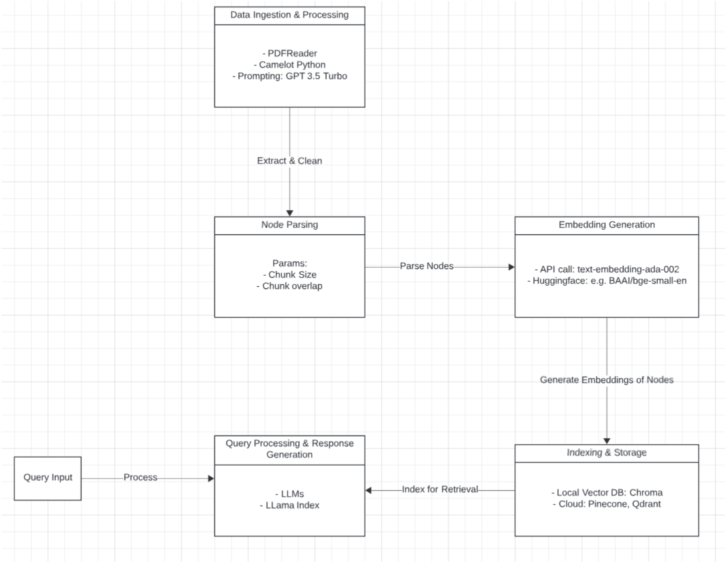


Architecture Overview (Cont.)

User Interaction Flow:

- Step 1: Field engineer inputs a query into the User Interface.
- Step 2: Query is transmitted to the Backend Server.
- Step 3: Backend Server communicates with the Model Server to process the query.
- Step 4: Model Server accesses the Database/Index to retrieve relevant information.
- Step 5: Processed response is sent back to the User Interface for the engineer to review.

Backend Components



Backend Components (Cont.)

- **Data Ingestion & Processing**: PDFReader and camelot libraries are used to extract and clean text data from refrigeration manuals in PDF format. Use of GPT-3.5 turbo for alignment and merging of text outputs.
- **Node Parsing**:
  1. Process of converting documents into a list of nodes using simple node parsing with parameters for chunk size and overlap.
  2. Add the option of the small-to-big retrieval approach for improvement.
- **Embedding Generation**: Options to use of various embedding models for retrieval, including OpenAI's "text-embedding-ada-002" and local models like "BAAI/bge-small-en"
- **Indexing & Storage**: Use vector databases to store index of nodes. Flexibility in choosing storage databases e.g.,
  - for small sized embedding models, use Chroma
  - for larger embedding models, use Pinecone
- **Query Processing & Response Generation**:
  - From query input, this retrieves suitable nodes (Llama index) to add additional and external contexts with input to get new input for LLM
  - LLM processes the new input and optional node postprocessor to process response

Interactive Frontend with Streamlit

- Frontend tool: Streamlit
- User's journey:

1. Accessing the app: https://qa-streamlit-test-c44c0e823dd8.herokuapp.com/
2. Selecting the language model: Mistral 7B, Mixtral 8x7B (Anyscale) (more can be added similarly)
3. Choosing the retrieval embedding model: via API or loading (small) model
4. Enabling/disabling the Cohere reranker.
5. Inputting the query.
6. Click Submit button
7. Viewing the generated response.

Deployment: Local & Heroku

Local Deployment:

- Clone the repo at https://github.com/finoceva/qa-streamlit-test.git and install dependencies.
- Run the Streamlit app via command line or Docker.
- Access at http://localhost:8501.

Cloud Deployment on Heroku:

- Containerize with Docker for consistent deployment.
- Push Docker image to Heroku Container Registry.
- Release and access the app with a Heroku URL: https://qa-streamlit-test-c44c0e823dd8.herokuapp.com/

Why Heroku?
- Small/Medium projects
- Simpler than other clouds to setup and deploy quickly
- Cheaper & straightforward pricing

Data Ingestion and Text Processing

Naive approach:

- Text outputs after loading by PDF readers are not always in natural reading order. This can affect later steps in pipeline (e.g., chunking and embedding).

Semi-manual Processing:

- PDFReader from Llama Hub as initial text extraction from the compressor manual.
- Lightweight table parsing: Camelot Python to detect and extract contents from tables in manual accurately, and then simply transform these outputs to natural language texts.
- Write a prompt template for GPT 3.5 Turbo to align and merge text outputs from Camelot and PDF Reader.

Node Parsing

Node Parsing is the conversion of processed text into chunk texts (nodes).

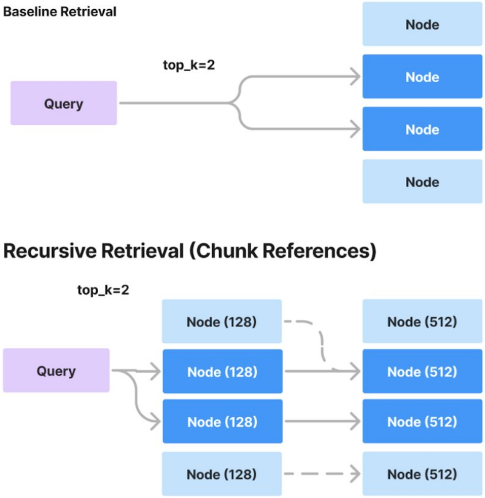
Simple approach:

- Chunk size: moderate size
- Chunk overlap: default setting

Small2Big approach:

- Main idea: retrieve on smaller pieces, expand into more context for LLM synthesis.
- Chunk references: Different smaller child chunk sizes (256) referring to bigger parent chunks (512).
- Recursive Retrieval
- Quite better than baseline using raw chunks (hit rate and MRR evaluation)

Node Parsing (Cont.)



Embeddings, Indexing, Storing & Querying

- Embedding Generation: try two quick options (OpenAI Embeddings via API and small specialized local models in retrieval in English) due to cost and time.
- Nodes are embedded into vector spaces, allowing for efficient similarity searches. First run with each different option would automatically store into a suitable vector database.
- For OpenAI embedding (dim = 1586), I used Pinecone (offers scalability and minimal maintenance). For small model embedding (dim = 386), I used Chroma DB (cheap and fast for small/moderate storage).
- Leveraging LLMs: opted for fast and cheap option calling via Anyscale API (Mistral AI models). Flexibility: Can add other models if needed.
- Optional Node Postprocessor: after initial set of results fetched by Llama index, reranker further analyzes results to improve relevancy (think as a secondary model acessing quality of the matches deeper).

Metrics-Driven Evaluation for QA system

- The evaluation framework for RAG system is designed to measure the performance and effectiveness of the Q&A system in providing accurate and relevant answers to field engineers.
- I follow RAGAS framework at https://github.com/explodinggradients/ragas
- Key Criteria Metrics:

- **Faithfulness**. Calculated from answer and retrieved context. The generated answer is regarded as faithful if all the claims that are made in the answer can be inferred from the given context.
- **Answer Relevance**. Computed using the question and the answer to evaluate how pertinent the generated answer is to the given question/prompt. Incomplete or redundant answers have lower scores.
- **Context Precision**. Computed using the question and the contexts to evaluate whether all of the ground-truth relevant items present in the contexts are ranked higher or not.

Metrics-Driven Evaluation for QA system (Cont.)

Some Details of RAGAS evaluation.

- Use of GPT-4-turbo (costly!) for evaluation on a synthetic QnA dataset created by GPT-3.5-turbo
- Ex: Evaluate for baseline and Small2Big on a 100-samples of the above test dataset

Evaluation Metrics Comparison

Retriever	Context Precision	Faithfulness	Answer Relevancy
Base	0.780	0.8239	0.9482
Small2Big	0.800	0.8177	0.9560

- Can run more eval for tuning params (chunk params, choices of LLMs, embeddings; choices of node postprocessors etc)

Thank you!