Contents

Multimodal RAG Framework – Iterative development roadmap

Discovery & Requirement Specification

Construct the business case

1. Document use cases
- Gather user stories
- Data source identification

2. Identify constraints
- Security
- Regulatory
- Legal, etc

3. Exploratory Documents Analysis
- Classify docs (text,pdf, html,etc)
- Document structure and elements
- Identify rich content

4. Gather representative test queries&docs
-Collect test documents
-Collect test queries/unaddressed queries
- Evaluate quality (relevance, pertinence)

<u>5. Identify gaps/additional data</u>
- Assess alternatives data sources:
  -FAQ
  -Similar external standards/regulations
  - Real/synthetic user queries/docs
  -Add metadata
  -Tabular data annotations
  -Domain specific taxonomies
  -Query logs and feedback
  -User context
  -Expert reviews/contradictory examples

6. Cost benefit/estimation
- Estimate solution benefit if required
- Estimate costs
- Determine delivery timelines

System Design & Development

Tech & Tools Selection

Orchestration (LangChain)

Ingestion & Indexing

Loading & Chunking

1.Preprocess
- Clean
- Classify items
- Reformat
- Convert to text
- Add metadata

2.Select chunk method
- Custom code
- Sentence based
- Fixed size
- Pre-built model
-Custom model
- Classification

3.Chunk enrichment
-Cleaning
-Augment (add info)

4. Persist
-Separate load&chunk
-Combine load&chunk

Embedding

1.Choose embedding model

a)Domain specific
-Sentence transformer
  multi-qa-mpnet-..
  - Cohere
  - Anthropic's Claude
b)General model
-Open AI
  text-embedding-3-l
-Sentence transformers
  all-mpnet-base-v2
-Google
  USE

Storing

1.Identify storing requirements
-Query
-Embedding
-Search
-Non-vector data
-Scalability
-Deployment type
-Persistence
-Compatibility

2.Chose index strategy
-Index types
-Latency
-Update frequency

3.Evaluate
-Frameworks
-Embedding models

4.Choose a vector store
-Single Modality
-Multimodal
-Non-Vector Backend

Answer queries

Retrieval

1.Identify modality representation
- Text
- Image
-Table/Graphs
-Diagrams

2.Choose Retrieval Architecture
- Single Search
- Multimodal Search
- Query preprocess

3.Choose Search mechanism
- Similarity metric
- Keyword
- Hybrid Search
- Approximate
- Multimodal match

4.Filtering&Ranking
-Metadata filtering
-Result Ranking

Generation

1.Choice language model
- Pre-trained
  gpt-40-mini
- Fine-tuned
-Multimodal

2. Constraints Control
-Response length
- Temperature

3.Context construction
-Query
- Retrieved docs
-Prompt design

4.Context management
-Token limits
- Dynamic
-Structured output

5.Post Generation
-Tone
- Branding
-Multimodal processing

Evaluation

Test-Driven Development

End – to End Testing

Stages and its measurements:

1.Retrieval
-Relevance
  -Top k docs (F1)
  - MRR
- Efficiency
  - Response Time
  -Scalability
- Out of the domain

2.Generation
-Relevance
-Groundedness
-Completeness

3.Multimodal alignment
-Fusion Quality
-Visual consistency

RAG single modal implementation based on task requirement

Discovery & Requirement Specification

Construct the business case

1. Document use cases
- Gather user stories
- Data source identification

2. Identify constraints
- Security
- Regulatory
- Legal, etc

3. Exploratory Documents Analysis
- Classify docs (text,pdf, html,etc)
- Document structure and elements
- Identify rich content

4. Gather representative test queries&docs
- Collect test documents
- Collect test queries/unaddressed queries
- Evaluate quality (relevance, pertinence)

5. Identify gaps/additional data
- Assess alternatives data sources:
  - FAQ
  - Similar external standards/regulations
  - Real/synthetic user queries/docs
  - Add metadata
  - Tabular data annotations
  - Domain specific taxonomies
  - Query logs and feedback
  - User context
  - Expert reviews/contradictory examples

6. Cost benefit/estimation
- Estimate solution benefit if required
- Estimate costs
- Determine delivery timelines

System Design & Development

Tech & Tools Selection

Orchestration (LangChain)

Ingestion & Indexing

Loading & Chunking

1.Preprocess
- Clean
- Classify items
- Reformat
- Convert to text
- Add metadata

2.Select chunk method
- Custom code
- Sentence based
- Fixed size
- Pre-built model
- Custom model
- Classification

3.Chunk enrichment
- Cleaning
- Augment (add info)

4. Persist
- Separate load&chunk
- Combine load&chunk

Embedding

1.Choose embedding model

a)Domain specific
- Sentence transformer multi-qa-mpnet-..
- Cohere
- Anthropic's Claude

b)General model
- Open AI text-embedding-3-l
- Sentence transformers all-mpnet-base-v2
- Google USE

Storing

1.Identify storing requirements
- Query
- Embedding
- Search
- Non-vector data
- Scalability
- Deployment type
- Persistence
- Compatibility

2.Chose index strategy
- Index types
- Latency
- Update frequency

3.Evaluate
- Frameworks
- Embedding models

4.Choose a vector store
- Single Modality
- Multimodal
- Non-Vector Backend

Answer queries

Retrieval

1.Identify modality representation
- Text
- Image
- Table/Graphs
- Diagrams

2.Choose Retrieval Architecture
- Single Search
- Multimodal Search
- Query preprocess

3.Choose Search mechanism
- Similarity metric
- Keyword
- Hybrid Search
- Approximate
- Multimodal match

4.Filtering&Ranking
- Metadata filtering
- Result Ranking

Generation

1.Choice language model
- Pre-trained gpt-40-mini
- Fine-tuned
- Multimodal

2. Constraints Control
- Response length
- Temperature

3.Context construction
- Query
- Retrieved docs
- Prompt design

4.Context management
- Token limits
- Dynamic
- Structured output

5.Post Generation
- Tone
- Branding
- Multimodal processing

Evaluation

Test-Driven Development

End – to End Testing

Stages and its measurements:

1.Retrieval
- Relevance
  - Top k docs (F1)
  - MRR
- Efficiency
  - Response Time
  - Scalability
- Out of the domain

2.Generation
- Relevance
- Groundedness
- Completeness

3.Multimodal alignment
- Fusion Quality
- Visual consistency

Implementation Summary – PseudoCode

Please refer to notebook for full details

Exploratory Data Analysis

1. Setting up environment
2. Load Documents
3. Get documents length distribution
4. Find out most frequent word in the documents
5. World cloud of most frequents words
6. TD-ID features
7. Metadata summary, number of paragraphs, content, etc.
8. Text length distribution
9. Display top 5 longest sentences from each document
10. Sentiment analysis distribution
11. Get most frequent sentence length

Rag Model Implementation

1. Set up environment.
2. Load documents from a directory into memory (not required persistency by now).
3. Initialize models (embeddings and LLM).
4. Index documents into a vector store by chunking based on section headers(**)
5. Create a prompt template for the RAG process.
6. Define State to track question, context, and answer.
7. Define functions:
   Retrieve relevant documents based on the question.
   Generate an answer using OpenAI's API.
8. Define evaluation functions:
   Evaluate groundedness
   Evaluate.
   Evaluate completeness
9. Run evaluation for each test case and calculate the aggregated results.
10. Display evaluation results (groundedness, relevance, completeness scores)

Implementation Evaluation –Please refer to notebook for full details

The evaluation of the Retrieval-Augmented Generation (RAG) model involves calculating three metrics: Groundedness, Completeness, and Relevance.

1. Groundedness: Verify if the generated answer is grounded in the context provided by the retrieved documents.
:
The evaluate_groundedness compares the answer against the content of each document, specifically looking for exact sentences or parts of sentences (sentence-level overlap).If any sentence from the document appears in the answer, it returns a score of 1.0 (indicating full grounding).If no overlap is found, it returns a score of 0.0.

2. Completeness: Check if the answer includes all key points from the expected answer.

The evaluate_completeness function compares the words in the expected answer against those in the generated answer.It splits both the expected answer and the generated answer into words and checks for the presence of each word from the expected answer in the generated answer.
The completeness score is calculated by dividing the number of words in the expected answer that are present in the generated answer by the total number of words in the expected answer.

3. Relevance: Determine if the generated answer is relevant to the question.

The evaluate_relevance function computes the overlap between the tokens (words) of the expected answer and the generated answer. It does so by comparing the set of tokens (unique words) in both the expected and generated answers.The relevance score is computed as the ratio of the intersection of tokens in both the generated answer and expected answer, divided by the total number of tokens in the expected answer.The higher the overlap of words between the generated and expected answers, the higher the relevance score.

4. Overall Evaluation:
The evaluate_rag_model function iterates over multiple test cases, retrieves relevant documents, generates answers, and calculates the three metrics (groundedness, completeness, and relevance) for each test case.
Finally, it averages the individual scores for each metric across all test cases, returning an overall evaluation for the RAG model.

Final Evaluation Scores: groundedness. 1.0, completeness 0.856, relevance: 0.834

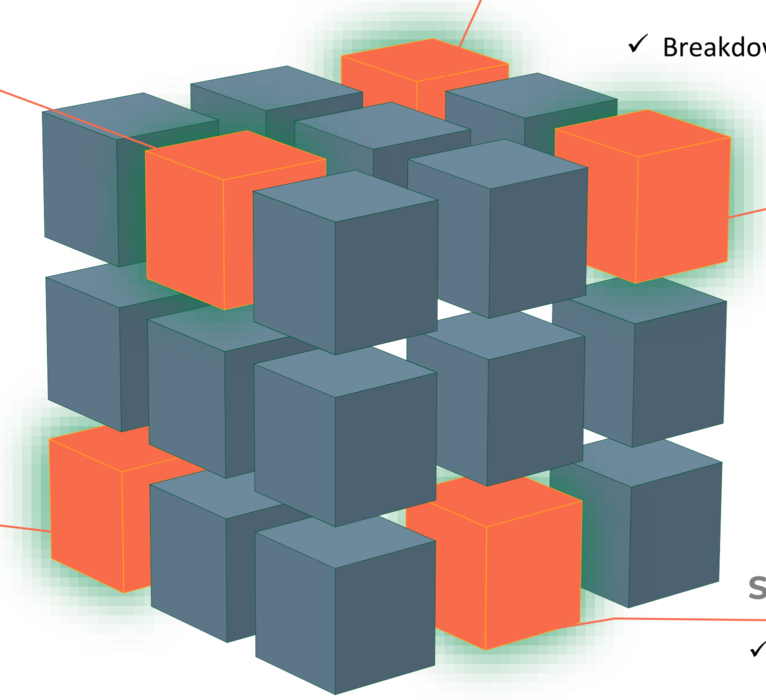
Considerations and Recommendations

Multimodal Integration Complexity

- ✓ Use specialized embedding techniques
- ✓ Fine-tune the model to prioritize multimodal reasoning
- ✓ Utilize attention mechanisms in multimodal transformers focus on relevant portions of text, graphs, tables, etc.
- ✓ Annotate datasets with multimodal relationships
- ✓ Breakdown complex user queries into multimodal tasks

Overcome Data quality issues

- ✓ Implement thorough cleaning pipelines
- ✓ Add Validation pipelines
- ✓ Regular update policy documents
- ✓ Add additional data sources listed previously
- ✓ Ensure complete representation of edge cases
- ✓ Avoid over-reliance on commonly retrieved docs
- ✓ Train models on diverse queries
- ✓ Audit regularly model outputs
- ✓ Granular metadata annotation



Performance

- ✓ Develop robust fallback mechanisms
- ✓ Error Analysis Workflow
- ✓ Automate aspects of RAG maintenance
- ✓ Leverage data version control to reduce the effort required to adapt to incoming data or feedback
- ✓ Fine-grained user personalization
- ✓ Implement re-indexing pipelines to ensure embedding reflect latest data
- ✓ Use serverless architecture or containerized deployments to handle workloads dynamically and ensure reproducibility

Ongoing Improvements

- ✓ Implement robust frameworks from the start
- ✓ Using Langchain to leverage best practices and orchestration capabilities
- ✓ Use modular components (OOP)
- ✓ Optimize indexing and retrieval mechanisms
- ✓ Strive for explainability
- ✓ Proactive adaptation
- ✓ Continuous monitoring

Security and Privacy

- ✓ Encrypt documents embeddings and retrieval request to prevent unauthorized access.
- ✓ Implement role-based access control