RAG

The AI assistant for future

Introduction

1. Problem statement: Customer need to query and answer according to policy documents

Proposed Solution Phases Road map Overview

1. System design, interfaces and architecture
2. Pure Text document system
3. Accepting other file format
4. Tool calling
5. Graph-rag
6. Agent-rag
7. Analysis capability
8. Data collection for training/ fine tuning a customised LLM model.

Break down of the roadmap

System design, interfaces and architecture

– Why?

Top down design for macroscopic concern

Divide components for development in parallel

Early design decision can have long term impact

System design, interfaces and architecture

– Use case analysis

Overview : We are developing a policy query assistant using RAG-LLM
Policy Document update is much less frequent than queries

– Functional requirements

Answer query if the query relevant to policies

Do not answer if irrelevant

Will have table, image other data types

Pure Text document system

Scope includes:

1. RAG system able to answer queries directly related to pure text documents
2. Server serving with API
3. Ability to self correct and self detect errors
    a. Hallucination check, self evaluation when producing response, give chance for self-revision
4. Logging system
    a. Service monitoring and check model drift, with quality assurance

Accepting other file format

1. Image
    a. OCR
    b. Image embedding e.g. Vision transformer based encoder
2. text doc along with image
    a. Wil reuse modules in 1
3. tables in excel, csv formats
4. Connectors (LM assisted Adaptors) to help with queries
    a. E.g. LLM assisted sql statement generation to get data from sql/ graphql

Tool calling

Tool call is a generic interface for RAG to enhance feature.

Tools to analyse and correlate policy terms, look up for glossary/ dictionary

Tools to find deep insights with policy data

Tools to log its own data and provide shortcuts to future similar queries

Tools for webhook to inform about the current thinking steps

Tools for analysing table and images

Graph-rag

One of the important enablement for the rag to use knowledge graphs.

Store indirect relationship between documents

A form to store past queries in forms of graph, instead of simple cache

Link all forms of data

Agent-rag

One of the important enablement for the rag to use knowledge graphs.

Higher level API around previous stages.

Network of agents collaborate to achieve analysis capability.

Data collection for training/ fine tuning a customised LLM model

Data collected from beginning of the earlier version. Conversation data can be used to feed in to fine tune/ retrain our LLM.

This is continuous during the process of all previous steps and lasts.

Data exploration

It is in markdown but some parts need string replacement such as [Company Name]

```
### Comprehensive Data Privacy Policy

**1. Introduction**

**Purpose of the Policy:**
At [Company Name], safeguarding the privacy and security of personal data is a foundational principle of our business

**Scope of the Policy:**
This policy applies universally to all personal and sensitive information collected by [Company Name] from our custo
```

System design iteration

We will scale out from base case design

1. Everything in Ram
2. Embedding documents
3. Embedding documents with indexing

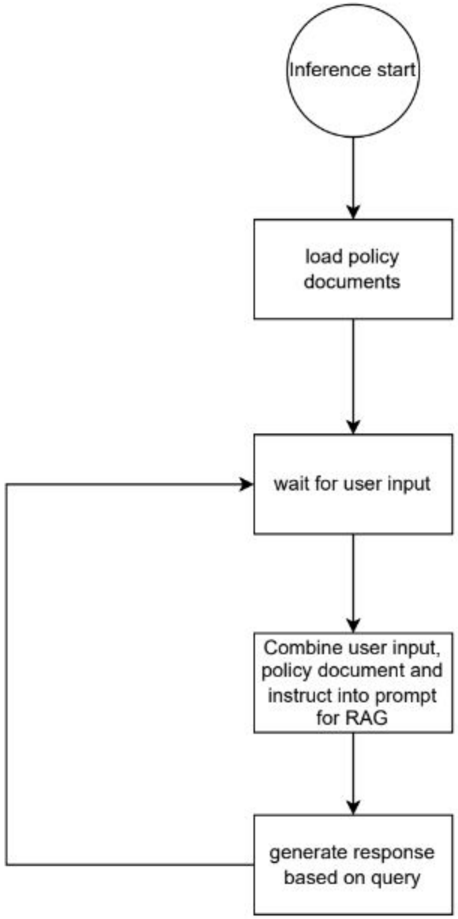
Base Case design

Only for a few documents that can fit in the context windows.

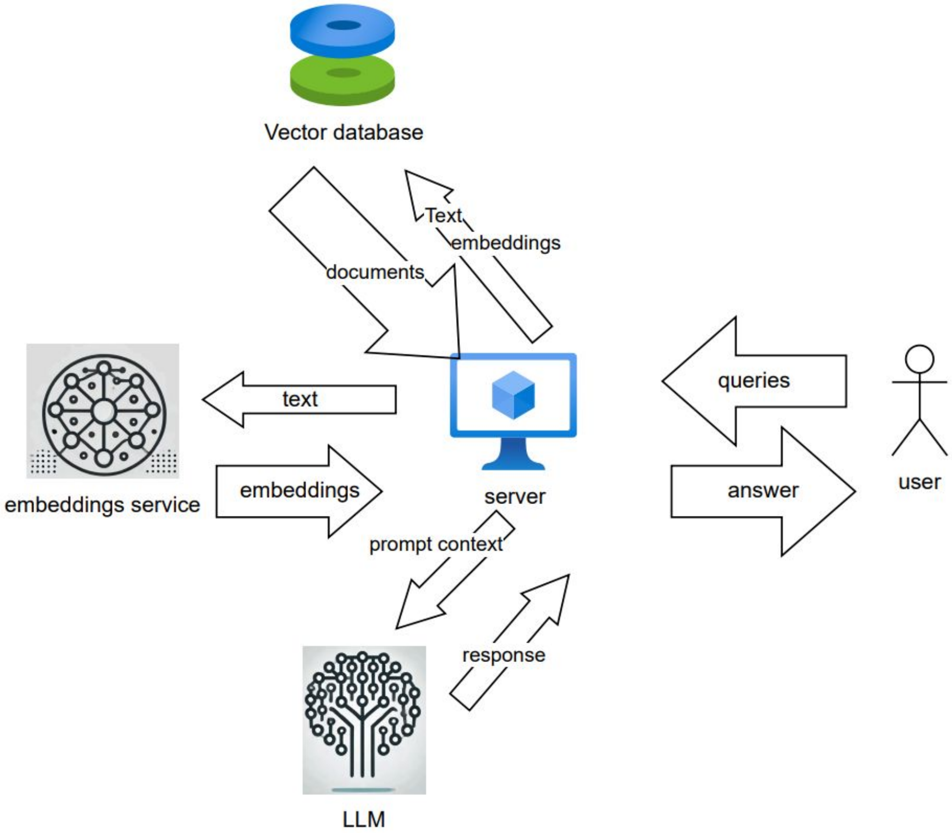
Everything in system RAM,

No embeddings

No vector databases



Components of the RAG system



Tech Stack design

LLM interface: Langchain – it has many features, but the main reason to use is to have adaptability and versatility.

Langchain has wide community support. Any component not work, has replacement from a separate vendor.

Easy to experiment with various LLM/ DB. Can switch to certain LLM if some are better are doing certain tasks

Language & ecosystem : python with large community and official support (microsoft has the creator of python as its employee, and Microsoft has Azure OpenAI, and a large stake in OpenAI)

Tech Stack design

Business perspective

Prevent vendor lock-in/ premature optimization with particular implementation of features.

Achieving loose coupling principle

Tech Stack design example

Focal

– Fast-API

– OpenAI

– Chroma-db

– Ada embedding

– Langchain

Implementation

Jupyter notebook illustration

Basic peeking of data

Markdown, with placeholder to be replaced.

Format:

A section-subsection format

Token each file: around 1000 tokens each, much less than the context window limit of popular LLMs

Basic prompt setup – all documents in prompt

"You are an expert policy query answering agent. You are given policy documents and you need to answer questions. "

"You cannot do any work for users. For example, you cannot send email, convert data or extract database files."

"If you are unsure about the content, just tell you are not sure. The documents are below: \n"

f"{article_delim.join(policy_doc_content)}"

f"User question : {user_query}")

Sanity test

Q: "Can you please convert my data into json format and sent to my email?"

A: "I'm sorry, but I cannot assist with converting data or sending emails."

Q: "When will user data be deleted?"

A: "User data will be deleted once it is no longer necessary to fulfill the stated purposes for which it was collected. After this period, the data is securely deleted or anonymized, in accordance with Canyon's strict data retention policy."

Time taks ~ 3.87 seconds

Rule of thumb: Guidance

Instruct the model think through step by step:

Steps for Question filtering:

is_query_only – statement? Asking for action? Asking for information?

Steps to give answer: (how to think)

Analysis, excerpts, reference, answer

Steps to review answer: (how to review)

answer_is_based_on_citation

Peek inside results after using Guidence

User input: Today weather is great

analysis="The user's input is a statement about the weather and does not pertain to any policy-related query. Therefore, it does not require an analysis based on the provided policy documents."

excerpts=[]

reference=''

answer='The statement about the weather does not relate to any policy content.'

answer_is_based_on_citation=False

is_query_only=False

Improvement proposals

Refining the base case 1 semantic splitting

1. Tell the LLM what document type, tell the model to parse semantically
2. Error check by the parsed task joins up makes up the original test.
3. Takes around 40 seconds for a document

Retrieval:

Look up by embedding vector against the embeddings of parsed document

Rag- Retrieval Augmented Generation Evaluation– Retrieval

Retrieval evaluation

Questions covering section of document and check the coverage of retrieval could cover the section it should lookup.

The retrieval has 85% coverage.

Retrieval evaluation

A case it does badly:

Q: How is transparency ensured in AI decision-making processes?

It shows that this retrieval method does not put focus into transparency. It did not search through the glossaries. And not every occurrence of “transparency” was retrieved.

Let’s add indexing and keyword lookup

Refining the base case 2 semantic splitting + index

1. Tell the LLM what document type, tell the model to parse semantically
2. Generate summary and Index words for lookup
3. Error check by the parsed task joins up makes up the original test.
4. Takes around 40 seconds for a document

Look up by embedding vector against the embeddings of parsed, indexed, and summaries

Evaluation of retrieval

Method	Coverage	Retrieval time for 20 queries	Embedding time
In memory all documents	100%	~0	~0
Semantic splitted embeddings	86%	5.89s	~3 minutes
Semantic splitted embeddings With index	93%	12s	~3 minutes

Rag- Retrieval Augmented Generation Evaluation– Generation

Evaluation Criteria

1. Check the user input is a question or not
2. Check if the answer generated is answer based

3. Any of the condition that return False will be classified as “reject answering”

Classification problems

(Response are logged for further analysis in future.)

Evaluation

Classification of

1. Is question vs not question
2. Answer is based on data

Both problems has false positive as more important

TN, FP

FN, TP

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

Evaluation metric

False positive– when the policy explanation is actually incorrect but classified as correct, misleading the end user

Not covered by traditional precision, recall or ROC

Testing scheme

60 questions asked

– 20 answerable

– 20 not answerable

– 10 are just statements not query

– 10 are non verbal requests

Results

In memory	predict negative	predict positive
Actual negative	28	12
Actual positive	0	20

RAG	predict negative	predict positive
Actual negative	19	21
Actual positive	0	20

Indexed RAG	predict negative	predict positive
Actual negative	35	5
Actual positive	0	20

Summary of results

	In memory	rag	Index rag
False positive rate	17%	30%	7%
Model usage cost (USD)	0.64	0.18	0.12
Query time (seconds)	4.6	3.83	3.45

Summary (2)

Quick development and have a base prototype: In memory

Fastest response Best improve inference accuracy: indexed Rag

Recommendations + To dos

Each LLM step should have an hallucination/ error fallback

E.g. the semantic parsing need to have a character splitter as a fallback in case LLM makes error for a specific document or content filtered by error.

Recommendations + To dos

To-dos

1. Evaluate content of the answer using LLM to check if the answer has covered all required content
2. Now only the embedding has index and keywords incorporated. We can preprocess the query such that it generate a few keyword items to look up in vector db
3. Adding other metadata such as llm detected section number
4. Become a QA agent

Discussions

In production, some of the requests can be processed in parallel such as the embedding or index resolving.

It is very tempting to make own LLM but time for training already could exceed and the cost is high with uncertain results.

Consider cost, embedding time, execution time

Long time for training/ embedding, quick inference - correspond to lambda architecture

Development and production

In actual development, packages such as logging, pytest ispreferred, and many reusable codes should be packaged

The rag model should be served in a server with API developed.

Thank you

Hope to talk and see you soon!

Please feel free to give feedback and comments

Other methods

Glossary lookup included workflow:

When embedding,

1. include the glossary when generating the subdocument embeddings
2. Multiple embeddings for same document i.e. index nodes
    a. Keyword index
    b. Subdocument summary