Local Retrieval Augmented Generation

Alexander Harris

Computer Scientist, Operations Research Analyst

github.com/alexanderusaf/local_rag_python

Overview

Core Concepts

Why use RAG? WIIFM?

Strengths and Weaknesses of RAG

Practical RAG scenario

Hands on RAG demonstration

Pre-requisites

Concepts you should know before this course

Python knowledge

Command line interface familiarity

Understanding of client/server relationships

Basic knowledge of large language models

Core Concepts

Foundational knowledge for retrieval augmented generation

Local - serving the large language model on your personal hardware

Retrieval - finding relevant information from a knowledge base

Augmentation - the integration of the retrieved information into context for an LLM

Generation - production of well-informed answers to a user’s question

What's in it for me?

Why use local RAG? Four critical advantages

Security - Ability to keep data and queries on your local machine

Accuracy - The LLM now references your authoritative sources

Currency - Information stays up-to-date as new documents are added

Control - You decide what knowledge the LLM will preference

Strengths of Local RAG

Increased Security

Accuracy of results

Currency and up-to-date information

Control over the large language model

Weaknesses of Local RAG

Reliance on the quality and relevance of the knowledge base

Potential for retrieval failures

Computational costs

Use Cases

How RAG can be used in the real world

Medical documentation

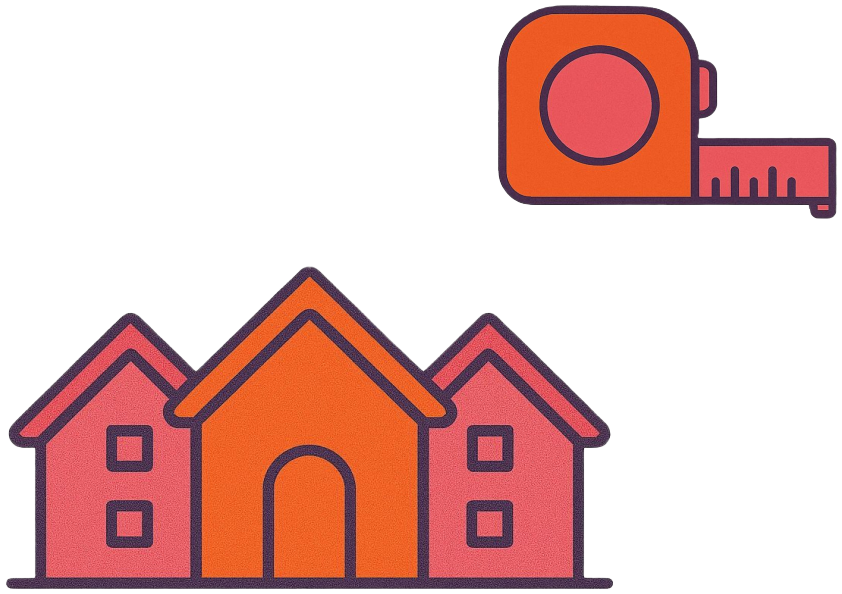
Technical support

Enterprise knowledge management

Legal research

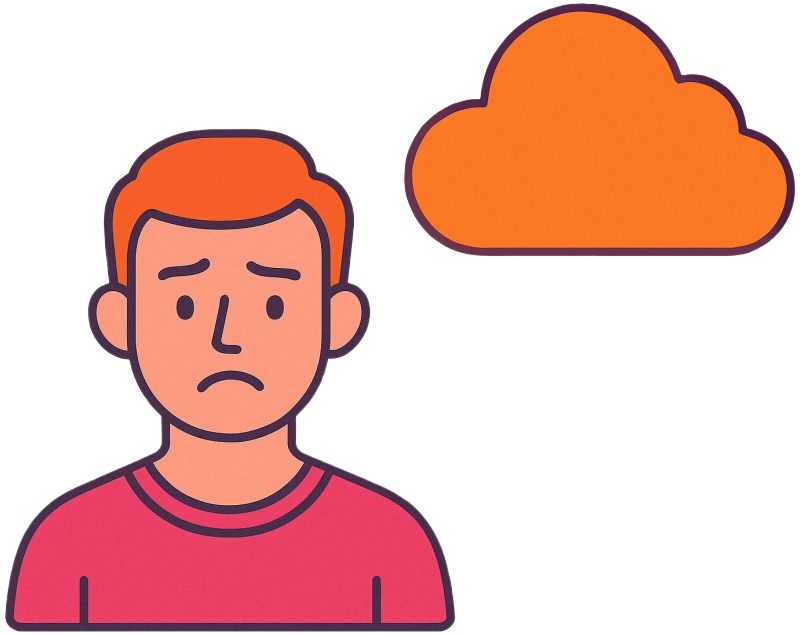
Linus is a software developer

He works on a legal research team



Linus’s team is working on a housing project

Security, accuracy, currency, and control are priorities



Chat GPT and other web-based LLMs

won’t meet Linus’s needs

Traditional solution? Wait for Chat GPT to update

Linus’s solution?

Build a localized retrieval augmented generation system

Demo

Access the code via Github :

github.com/alexanderusaf/local_rag_python

Follow the setup instructions

```python
import ollama, chromadb

documents = [
    "City of San Antonio Ordinance : PMC 404.4.1 Room area. Every living room shall contain at least 120 square
    feet (11.2m2) and every bedroom shall contain at least 70 square feet (6.5m2)"
]

client = chromadb.Client()
collection = client.create_collection(name="docs")

# store each document in a vector embedding database
for i, d in enumerate(documents):
  response = ollama.embed(model="mxbai-embed-large", input=d)
  embeddings = response["embeddings"]
  collection.add(
    ids=[str(i)],
    embeddings=embeddings,
    documents=[d]
  )
```

```python
import ollama, chromadb

documents = [
    "City of San Antonio Ordinance : PMC 404.4.1 Room area. Every living room shall contain at least 120 square feet (11.2m2) and every bedroom shall contain at least 70 square feet (6.5m2)"
]

client = chromadb.Client()
collection = client.create_collection(name="docs")

# store each document in a vector embedding database
for i, d in enumerate(documents):
  response = ollama.embed(model="mxbai-embed-large", input=d)
  embeddings = response["embeddings"]
  collection.add(
    ids=[str(i)],
    embeddings=embeddings,
    documents=[d]
  )
```

```python
import ollama, chromadb

documents = [
    "City of San Antonio Ordinance : PMC 404.4.1 Room area. Every living room shall contain at least 120 square
    feet (11.2m2) and every bedroom shall contain at least 70 square feet (6.5m2)"
]

client = chromadb.Client()
collection = client.create_collection(name="docs")

# store each document in a vector embedding database
for i, d in enumerate(documents):
  response = ollama.embed(model="mxbai-embed-large", input=d)
  embeddings = response["embeddings"]
  collection.add(
    ids=[str(i)],
    embeddings=embeddings,
    documents=[d]
  )
```

```python
import ollama, chromadb

documents = [
    "City of San Antonio Ordinance : PMC 404.4.1 Room area. Every living room shall contain at least 120 square
    feet (11.2m2) and every bedroom shall contain at least 70 square feet (6.5m2)"
]

client = chromadb.Client()
collection = client.create_collection(name="docs")

# store each document in a vector embedding database
for i, d in enumerate(documents):
  response = ollama.embed(model="mxbai-embed-large", input=d)
  embeddings = response["embeddings"]
  collection.add(
    ids=[str(i)],
    embeddings=embeddings,
    documents=[d]
  )
```

```python
# an example prompt
input = "My client is developing a low income housing project in the city of San Antonio. What are the size requirements for living rooms in the city of San Antonio?"

# generate an embedding for the input and retrieve the most relevant doc
response = ollama.embed(
  model="mxbai-embed-large",
  input=input
)

results = collection.query(
  query_embeddings=[response["embeddings"][0]],
  n_results=1
)
data = results['documents'][0][0]

# # generate a response combining the prompt and data we retrieved in step 2
output = ollama.generate(
  model="qwen3:8b",
  prompt=f"Using this data: {data}. Respond to this prompt: {input}"
)

print(output['response'])
```

```python
# an example prompt
input = "My client is developing a low income housing project in the city of San Antonio. What are the size requirements for living rooms in the city of San Antonio?"

# generate an embedding for the input and retrieve the most relevant doc
response = ollama.embed(
  model="mxbai-embed-large",
  input=input
)

results = collection.query(
  query_embeddings=[response["embeddings"][0]],
  n_results=1
)
data = results['documents'][0][0]

# # generate a response combining the prompt and data we retrieved in step 2
output = ollama.generate(
  model="qwen3:8b",
  prompt=f"Using this data: {data}. Respond to this prompt: {input}"
)

print(output['response'])
```

```python
# an example prompt
input = "My client is developing a low income housing project in the city of San Antonio. What are the size requirements for living rooms in the city of San Antonio?"

# generate an embedding for the input and retrieve the most relevant doc
response = ollama.embed(
  model="mxbai-embed-large",
  input=input
)

results = collection.query(
  query_embeddings=[response["embeddings"][0]],
  n_results=1
)
data = results['documents'][0][0]

# # generate a response combining the prompt and data we retrieved in step 2
output = ollama.generate(
  model="qwen3:8b",
  prompt=f"Using this data: {data}. Respond to this prompt: {input}"
)

print(output['response'])
```

```python
# an example prompt
input = "My client is developing a low income housing project in the city of San Antonio. What are the size requirements for living rooms in the city of San Antonio?"

# generate an embedding for the input and retrieve the most relevant doc
response = ollama.embed(
  model="mxbai-embed-large",
  input=input
)

results = collection.query(
  query_embeddings=[response["embeddings"][0]],
  n_results=1
)
data = results['documents'][0][0]

# # generate a response combining the prompt and data we retrieved in step 2
output = ollama.generate(
  model="qwen3:8b",
  prompt=f"Using this data: {data}. Respond to this prompt: {input}"
)

print(output['response'])
```

Response without RAG

According to the Texas Building Code… living room dimensions are typically:

- Clear width: 80 inches
- Depth: Approximately 7 feet

Response with RAG

…City of San Antonio, the size requirements for living rooms are specified in **Ordinance PMC 404.4.1**...

- **Every living room must contain at least 120 square feet (11.2 square meters)**.

This minimum size applies to all living rooms in residential unit…

Summary

Core Concepts

Why use RAG? WIIFM?

Strengths and Weaknesses of RAG

Real world use cases

Implementation of RAG