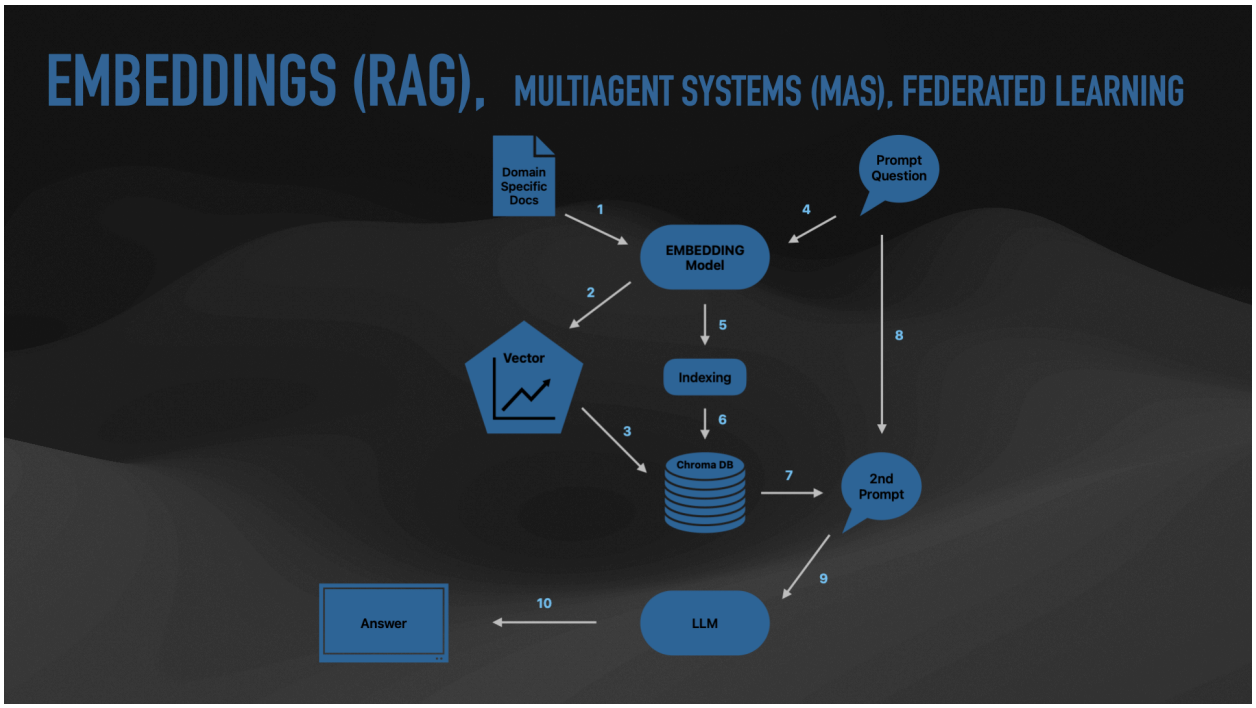
## The Architecture of Simple RAG:



## 1. Distance Function Alternatives in ChromaDB

ChromaDB uses different distance functions for computing similarity between embeddings. Alternatives include:

| Distance Function | Description |
|---|---|
| `cosine` | Measures the cosine of the angle between two vectors. Good for semantic similarity. |
| `l2` (Euclidean) | Measures straight-line distance between two points. Common for numeric data. |
| `ip` (Inner Product) | Measures the dot product. Works well with normalized embeddings. |

**Recommended Choice:**

- **Use `cosine` for text embeddings** (already chosen in your code).
- **Use `ip` for models that output normalized embeddings** (like some SentenceTransformer models).

---

## 2. Embedding Model Alternatives

Instead of `all-MiniLM-L6-v2`, you can use different embedding models:

| Model | Size | Speed | Performance |
|---|---|---|---|
| **`all-MiniLM-L6-v2`** | Small | Fast | Decent performance |
| `all-mpnet-base-v2` | Medium | Slower | Higher accuracy |
| `multi-qa-MiniLM-L6-dot-v1` | Small | Fast | Tuned for QA tasks |
| `e5-large-v2` | Large | Slower | State-of-the-art embeddings |
| `bge-base-en-v1.5` | Medium | Medium | Strong open-source alternative |

| | | | |
|---|---|---|---|
| `intfloat/e5-large` | Large | Slower | Good for multilingual search |

**Recommended Choice:**

- For speed: **`all-MiniLM-L6-v2`** (your current choice).
- For accuracy: **`all-mpnet-base-v2`** or **`e5-large-v2`**.
- For multilingual tasks: **`bge-m3`** or **`intfloat/multilingual-e5-large`**.

---

## 3. Large Language Model (LLM) Alternatives

Instead of `"DiscoResearch/Llama3-German-8B"`, you can use:

| Model | Size | Multilingual? | Pros | Cons |
|---|---|---|---|---|
| **Llama 3 (Meta)** | 8B / 70B | No | Strong performance | Needs fine-tuning |
| **Mistral 7B** | 7B | No | Efficient, fast inference | Not multilingual |
| **Gemma 7B (Google)** | 7B | No | Optimized for chat | Limited open-source support |
| **Mixtral (Mistral MoE)** | 12.9B | No | Very efficient, high quality | More complex setup |

**Recommended Choice:**

- **For speed & small-scale deployment**: **Mistral 7B**.
- **For high performance on a consumer GPU**: **Llama 3 8B**.
- **For multilingual support**: **Gemma 7B** or **bge-m3 for embeddings + Llama 3**.
- **For production-grade models**: **Mixtral (Mistral MoE) or GPT-4-turbo API**.

---

## Final Suggestions for Your Setup

1. Keep `cosine` distance for text embeddings.
2. Upgrade embedding model to `all-mpnet-base-v2` or `e5-large-v2`.
3. Use a stronger model like **Llama 3 8B**, **Mistral 7B**, or **Mixtral** for better responses.

COPY PASTE THIS PART INTO A .PY FILE:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
SIMPLE RAG 4 ACTINIUM

@author: Siawasch.Mirzaee@Gmail.com
"""
#!pip install torch
#!pip install requests
#!pip install langchain_community
#!pip install transformers
#!pip install chromadb

#client.delete_collection(COLLECTION_NAME)
#import os

import pandas as pd
import requests
#import torch
#from dotenv import load_dotenv
from langchain_community.document_loaders import DataFrameLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from transformers import AutoTokenizer, AutoModelForCausalLM
import chromadb
from chromadb.utils import embedding_functions
from chromadb.utils.batch_utils import create_batches
import uuid

# DOMAIN SPECIFIC DOX[1]: Function to fetch text from GitHub
def get_text(chapter: str) -> str:
        github_url =
f"https://raw.githubusercontent.com/keithmcnulty/peopleanalytics-regression-book/master/r/{cha
pter}.Rmd"
        result = requests.get(github_url)
        return result.text

# METADATA: List of chapters
chapter_list = [
        "01-intro", "02-basic_r", "03-primer_stats", "04-linear_regression",
        "05-binomial_logistic_regression", "06-multinomial_regression",
        "07-ordinal_regression", "08-hierarchical_data", "09-survival_analysis",
```

```
        "10-tidy_modeling", "11-power_tests", "12-further",
        "13-solutions", "14-bibliography"
]

# Fetch and store text of all chapters in a DataFrame
book_text = [get_text(chapter) for chapter in chapter_list]
book_data = pd.DataFrame({'chapter': list(range(14)), 'text': book_text})

# Split text into manageable chunks
loader = DataFrameLoader(book_data, page_content_column="text")
data = loader.load()
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=150)
docs = text_splitter.split_documents(data)

# Setup ChromaDB & EMBEDDING MODEL
CHROMA_DATA_PATH = "./chroma_data_regression_book/"
EMBED_MODEL = "all-MiniLM-L6-v2"

#EMBED_MODEL =  "all-mpnet-base-v2"
#EMBED_MODEL = "e5-large-v2"

COLLECTION_NAME = "regression_book_docs"
client = chromadb.PersistentClient(path=CHROMA_DATA_PATH)

# Enable the DB using Cosine Similarity as the DISTANCE FUNCTION
embedding_func =
embedding_functions.SentenceTransformerEmbeddingFunction(model_name=EMBED_MODE
L)
collection = client.create_collection(
        name=COLLECTION_NAME,
        embedding_function=embedding_func,
        metadata={"hnsw:space": "cosine"},
)

# Write text chunks to DB in batches
batches = create_batches(
        api=client,
        ids=[f"{uuid.uuid4()}" for i in range(len(docs))],
        documents=[doc.page_content for doc in docs],
        metadatas=[{'source': './handbook_of_regression_modeling', 'row': k} for k in
range(len(docs))]
)

for batch in batches:
```

```
        print(f"Adding batch of size {len(batch[0])}")
        collection.add(ids=batch[0], documents=batch[3], metadatas=batch[2])

# Load GPT-2 Model or other opensource LLM MODEL
#model_name = "DiscoResearch/Llama3-German-8B"
model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"
#model_name = "microsoft/Phi-3-mini-4k-instruct"

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Add pad token if not already present
if tokenizer.pad_token is None:
        tokenizer.add_special_tokens({'pad_token': tokenizer.eos_token})
        model.resize_token_embeddings(len(tokenizer))

# Function to ask a question (CALCULATING PROMPT)
def ask_question(question: str, model, tokenizer, collection_name=COLLECTION_NAME,
n_docs=3, max_new_tokens=50) -> str:
        # Find close documents in ChromaDB
        collection = client.get_collection(collection_name)
        results = collection.query(query_texts=[question], n_results=n_docs)

        # Collect the results in a context
        context = "\n".join([r for r in results['documents'][0]])

        prompt = f"""
        You are an expert on statistics and its applications to People Analytics.
        Here is a question: {question}
        Answer it with reference to the following information and only using the following
information: {context}.
        """

        # Generate the answer using the LLM
        input_ids = tokenizer.encode(prompt, return_tensors="pt", padding=True,
truncation=True, max_length=512)

        # Generate the response
        outputs = model.generate(input_ids, max_new_tokens=max_new_tokens,
pad_token_id=tokenizer.eos_token_id, num_return_sequences=1)

        # Decode the response
        response = tokenizer.decode(outputs[0], skip_special_tokens=True)
```

```
        return response

## TEST THE RAG:
ask_question("what is the best categorization method?", model, tokenizer)

ask_question("Describe survival analysis?", model, tokenizer)
```