Agenda

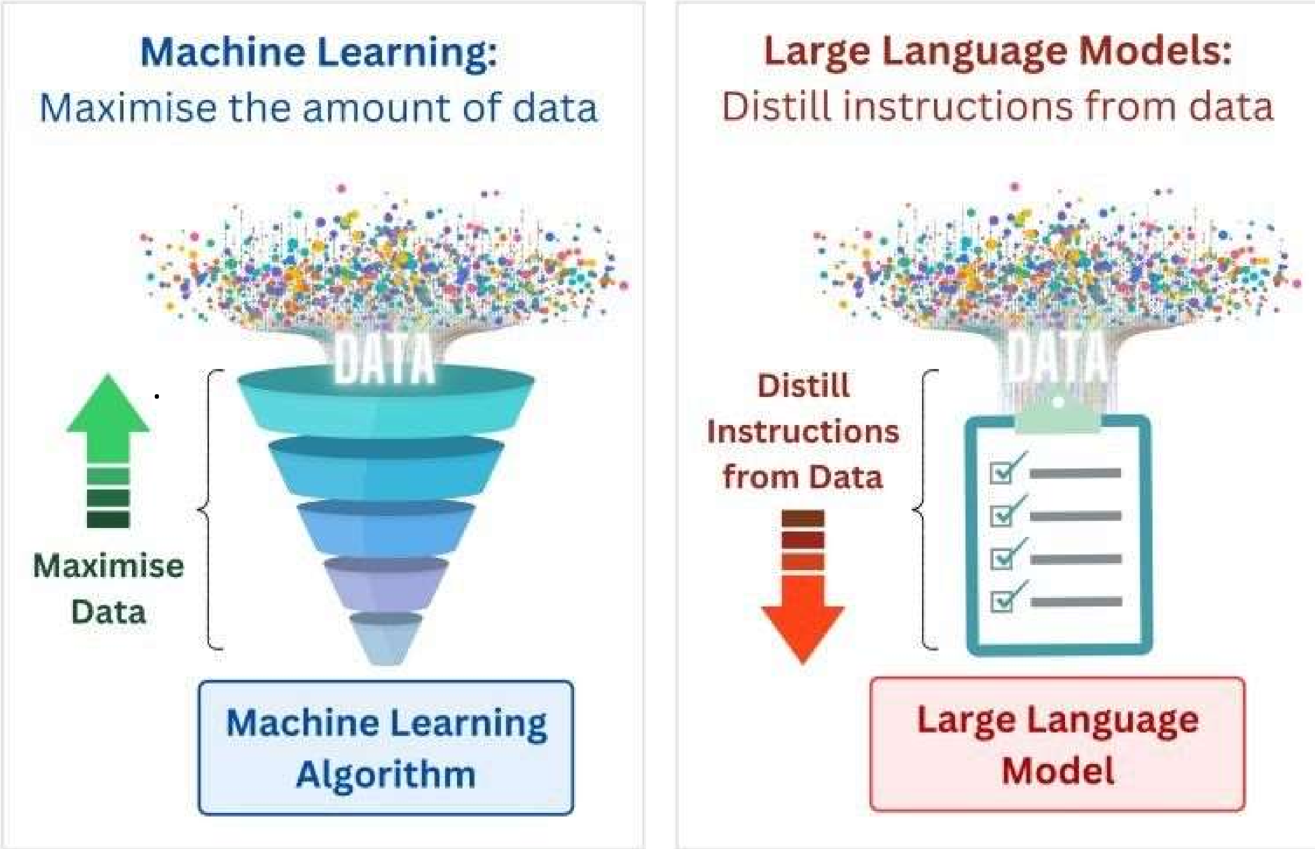
UNIDADE 7: Treinando Modelos LLMs

7.1 Retrieval Augmented Generation
7.2 Framework LangChain
7.3 Llama 3

Garbage in garbage out!

With LLMs we don't need as much data as before

LLMs cannot easily deal with a lot of data. Instead, we need to write data-informed instructions in a highly condensed way



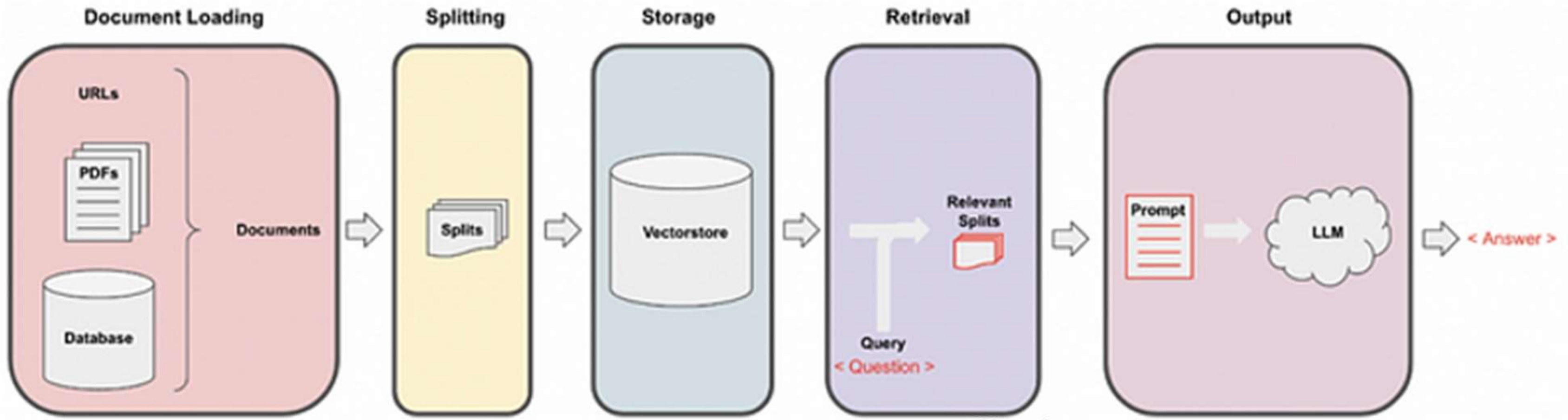
Source: Peter Gostev (https://www.linkedin.com/in/peter-gostev/)

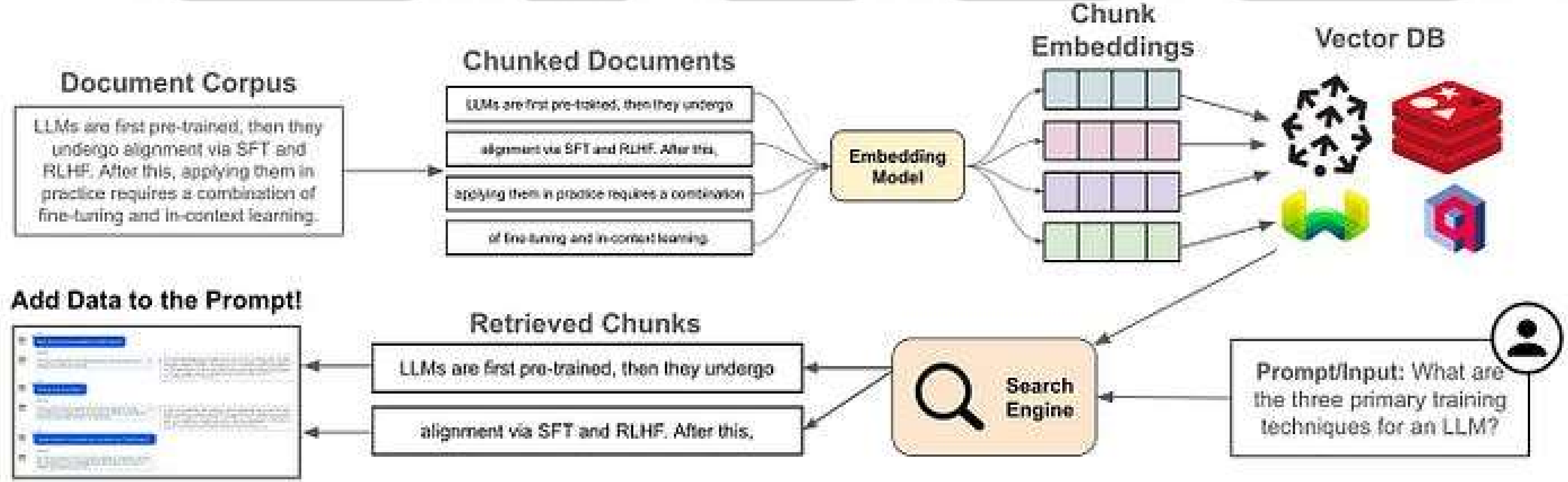
Retrieval-Augmented Generation (RAG)

- **Modelos como GPT, LLama e o Gemini** do Google são muito bons em entender e trabalhar com a linguagem humana. Mas, às vezes, não se saem bem com tópicos especiais ou novas informações.
- Para melhorar isso, os especialistas criaram um método chamado ***Retrieval-Augmented Generation, ou RAG***.
- O RAG ajuda esses modelos, permitindo que eles busquem informações em outras bases.

https://levelup.gitconnected.com/unlocking-llms-potential-with-rag-a-complete-guide-from-basics-to-advanced-techniques-b4557f268134

Retrieval-Augmented Generation (RAG)

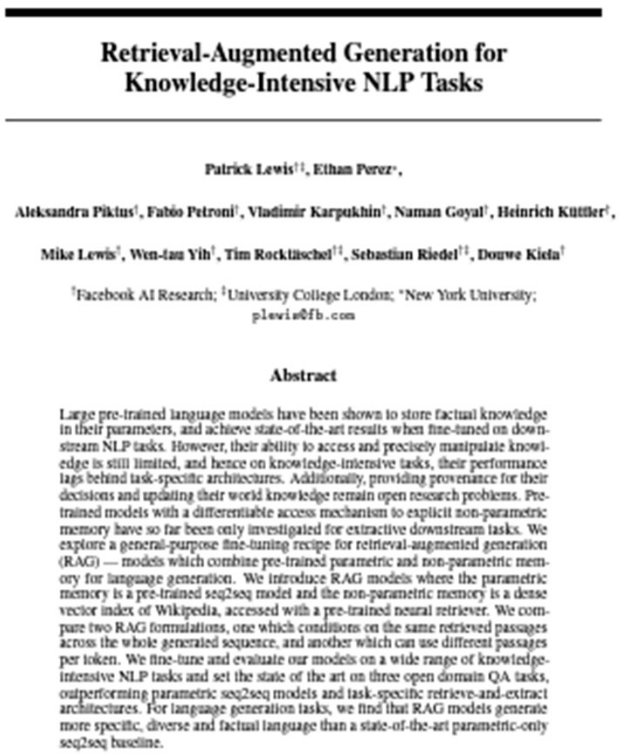




https://towardsdatascience.com/a-practitioners-guide-to-retrieval-augmented-generation-rag-36fd38786a84

Fonte: https://jorgepit-14189.medium.com/get-started-with-chroma-db-and-retrieval-models-using-langchain-87784ffaa918

Retrieval-Augmented Generation (RAG)



Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], Ethan Perez[⋆],

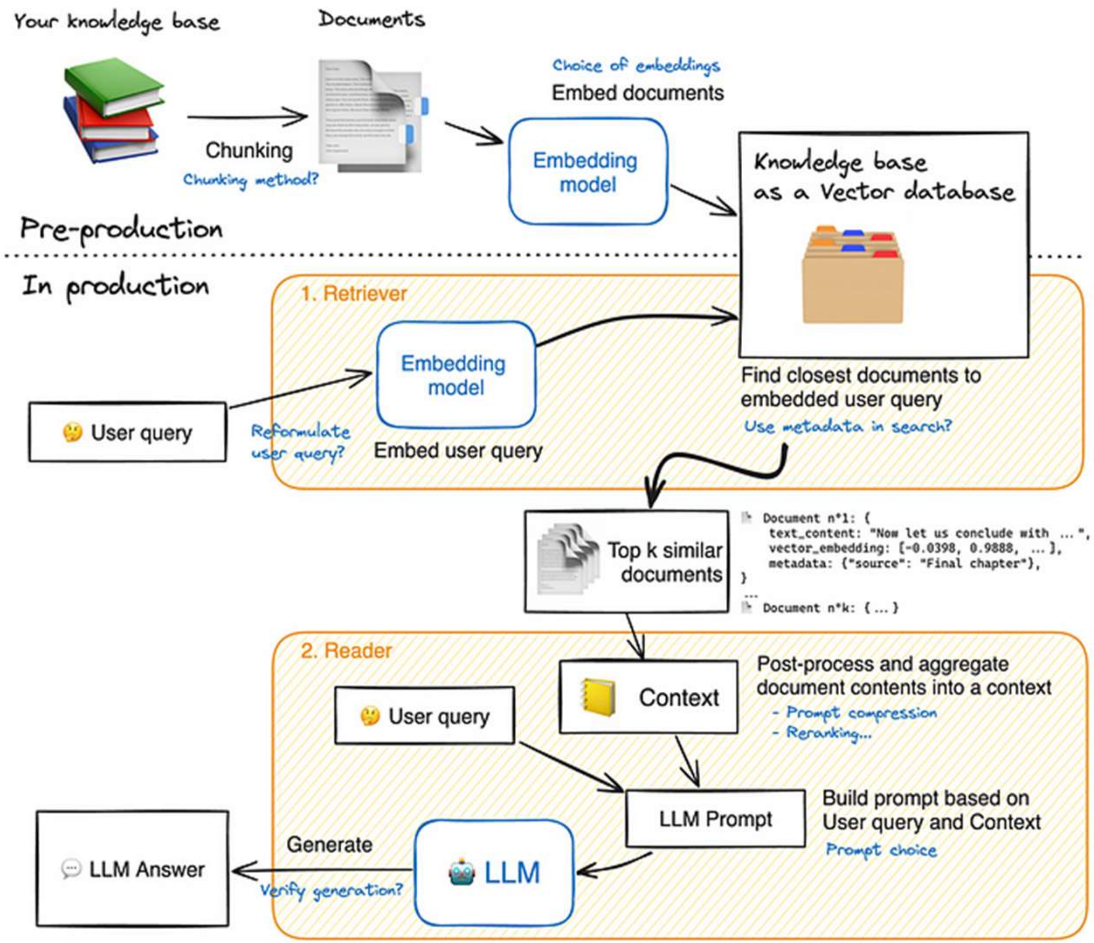
Aleksandra Piktus[†], Fabio Petroni[†], Vladimir Karpukhin[†], Naman Goyal[†], Heinrich Küttler[†],

Mike Lewis[†], Wen-tau Yih[†], Tim Rocktäschel[†‡], Sebastian Riedel[†‡], Douwe Kiela[†]

[†]Facebook AI Research; [‡]University College London; [⋆]New York University;
plewis@fb.com

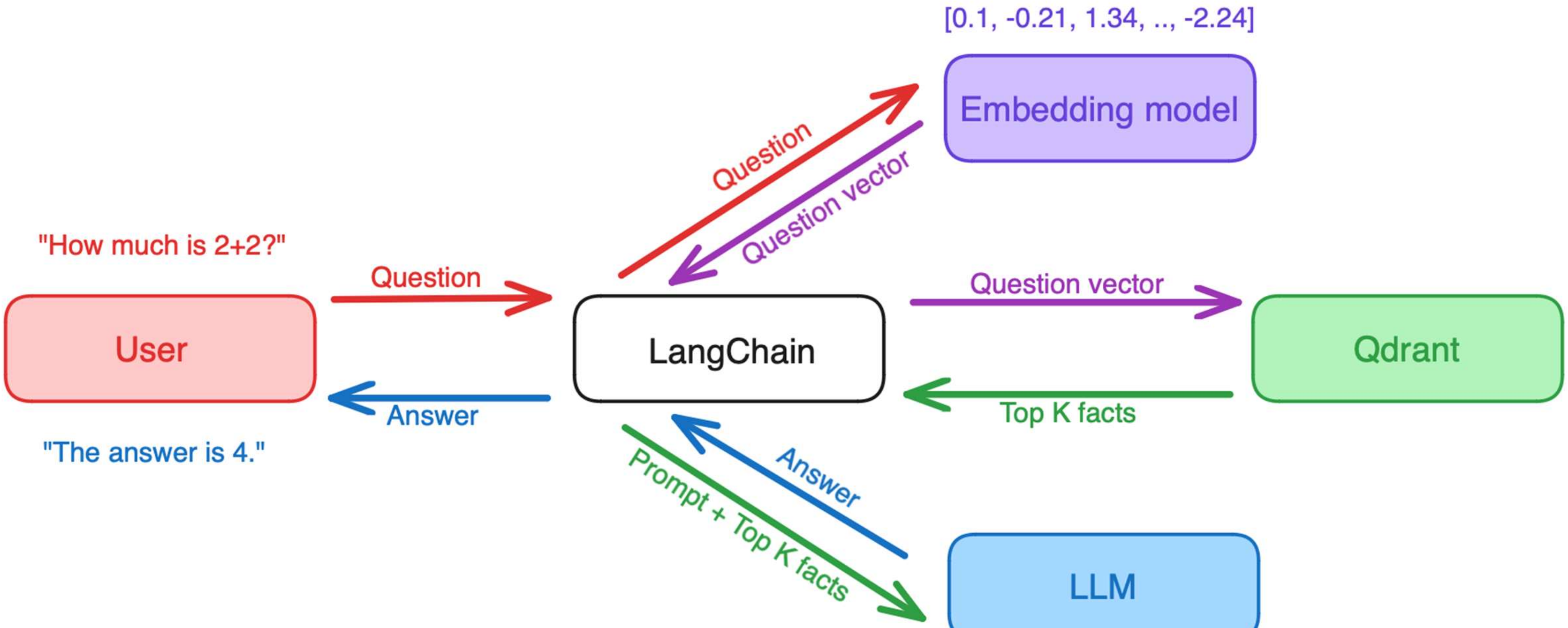
Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.



https://christiangrech.medium.com/evaluating-rag-performance-a-comprehensive-guide-b1d8f903b7ad

LangChain



https://levelup.gitconnected.com/unlocking-llms-potential-with-rag-a-complete-guide-from-basics-to-advanced-techniques-b4557f268134

Imports

```
from langchain.document_loaders import TextLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.chat_models import ChatOpenAI
from langchain.chains import RetrievalQA
from langchain.prompts import PromptTemplate
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
```

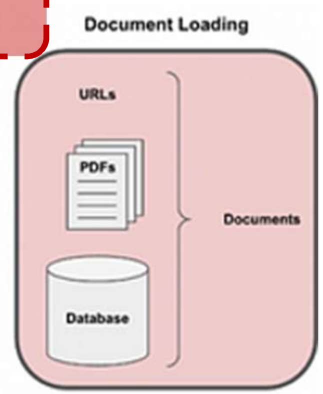
Framework — LangChain: Uma biblioteca que facilita a construção de aplicações que combinam modelos de linguagem com conhecimento externo.

Fonte: https://jorgepit-14189.medium.com/get-started-with-chroma-db-and-retrieval-models-using-langchain-87784ffaa918

Carregando os documentos

```
def mount_google_drive():
        """Mounts Google Drive for accessing files."""
        drive.mount('/content/drive’)
```

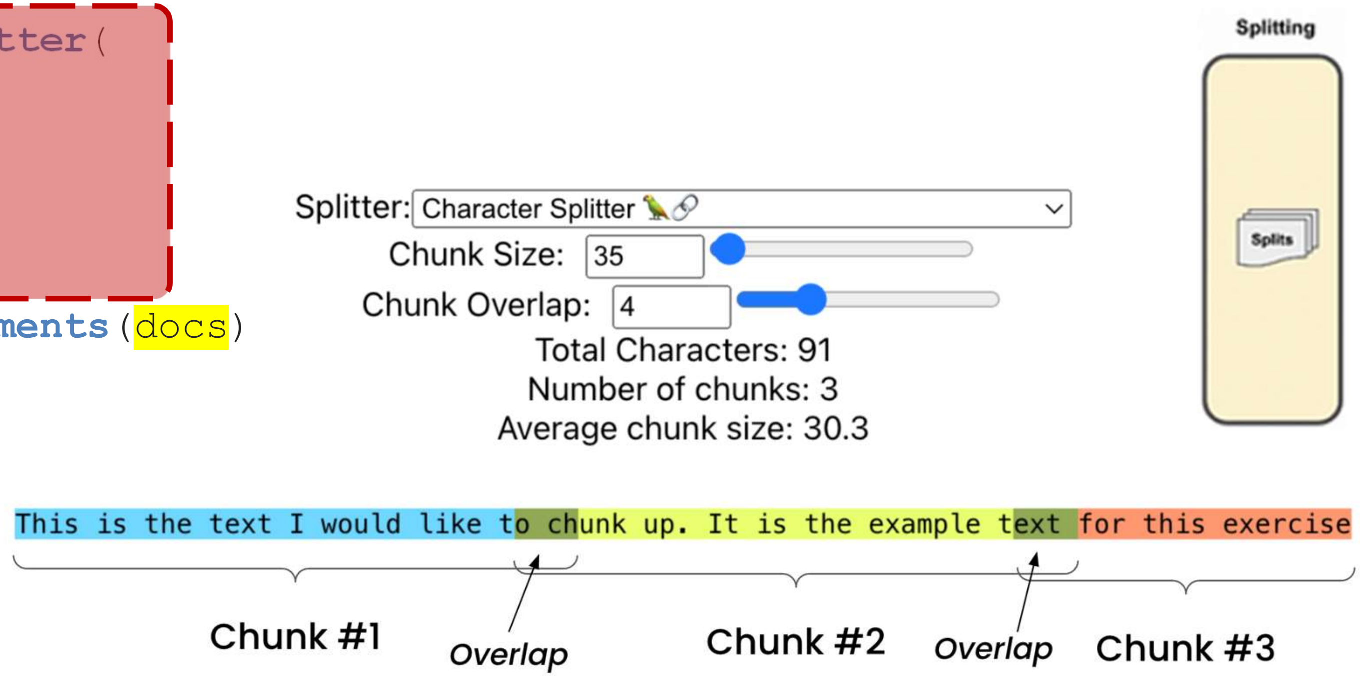
```
loader = TextLoader("/content/drive/MyDrive/LLM/docs/txt_documents/facts.txt")
docs = loader.load()
```



Document Loader — PyPDFLoader: ela carrega documentos PDF e os prepara para processamento.

Divide o documento em chunks

```
text_splitter = CharacterTextSplitter(
    separator = "\n",
    chunk_size = 200,
    chunk_overlap = 0

)
splits = text_splitter.split_documents(docs)
```



ChunkViz.com

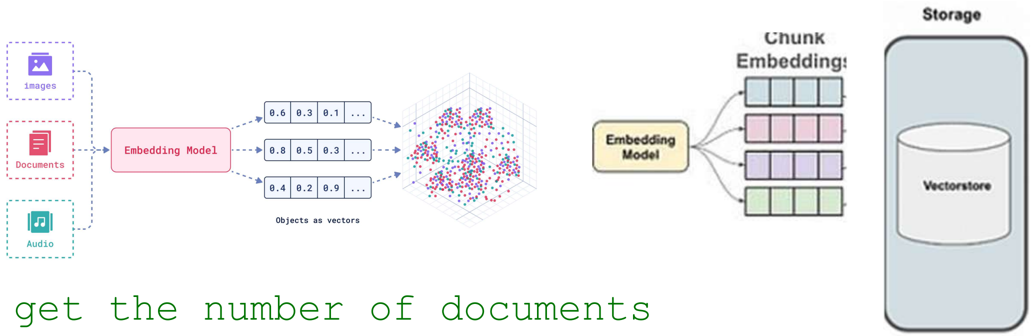
Text Splitter — RecursiveCharacterTextSplitter: usada para dividir documentos longos em pedaços menores para facilitar o processamento.

Cria os embeddings armazena no VectorStore

```
embedding = OpenAIEmbeddings()
persist_directory = '/content/chroma/'

if not os.path.exists(persist_ directory):
        os.makedirs(persist_ directory)

vectordb = Chroma.from_documents(
    documents=splits,
    embedding=embedding,
    persist_directory=persist_directory
)
# save the database so we can use it later
vectordb.persist()

# check that the database have been created and get the number of documents
print(vectordb._collection.count())
```

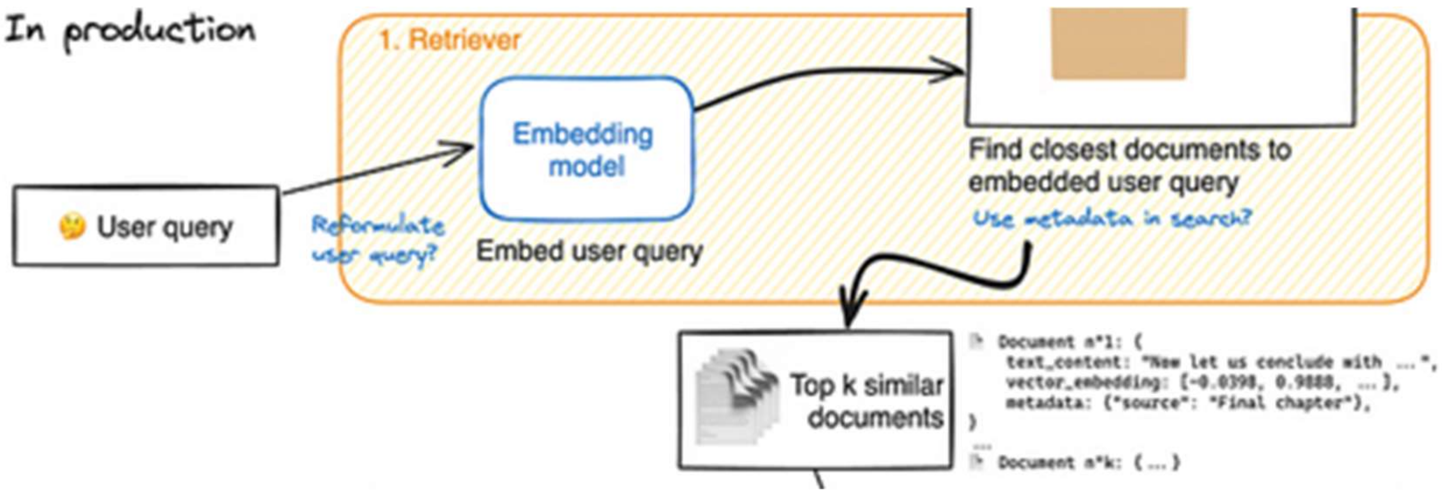
Embeddings Model — OpenAI
Embeddings: Converte texto em embeddings, uma representação numérica que as máquinas podem entender e processar.

Vector Database — ChromaDB: Usado para criar um banco de dados vetorial que armazena embeddings de documentos para a recuperação eficiente de informações.



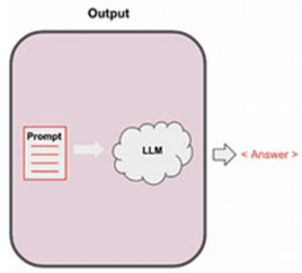
Carregue o VectorStore

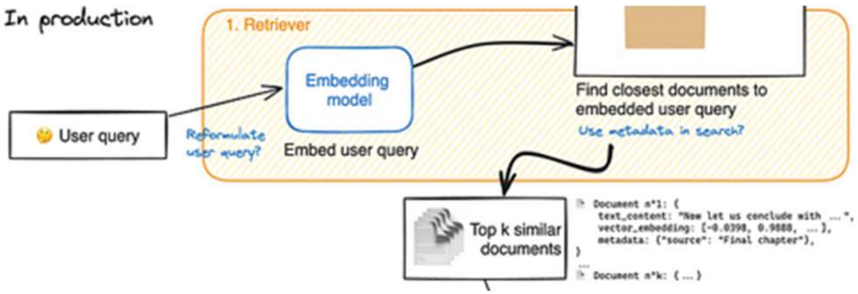
```python
persist_directory = '/content/chroma/'

# load again the db
vectordb = Chroma(
    persist_directory=persist_directory,
    embedding_function=embedding
)
```



Recuperando documento similares (teste)

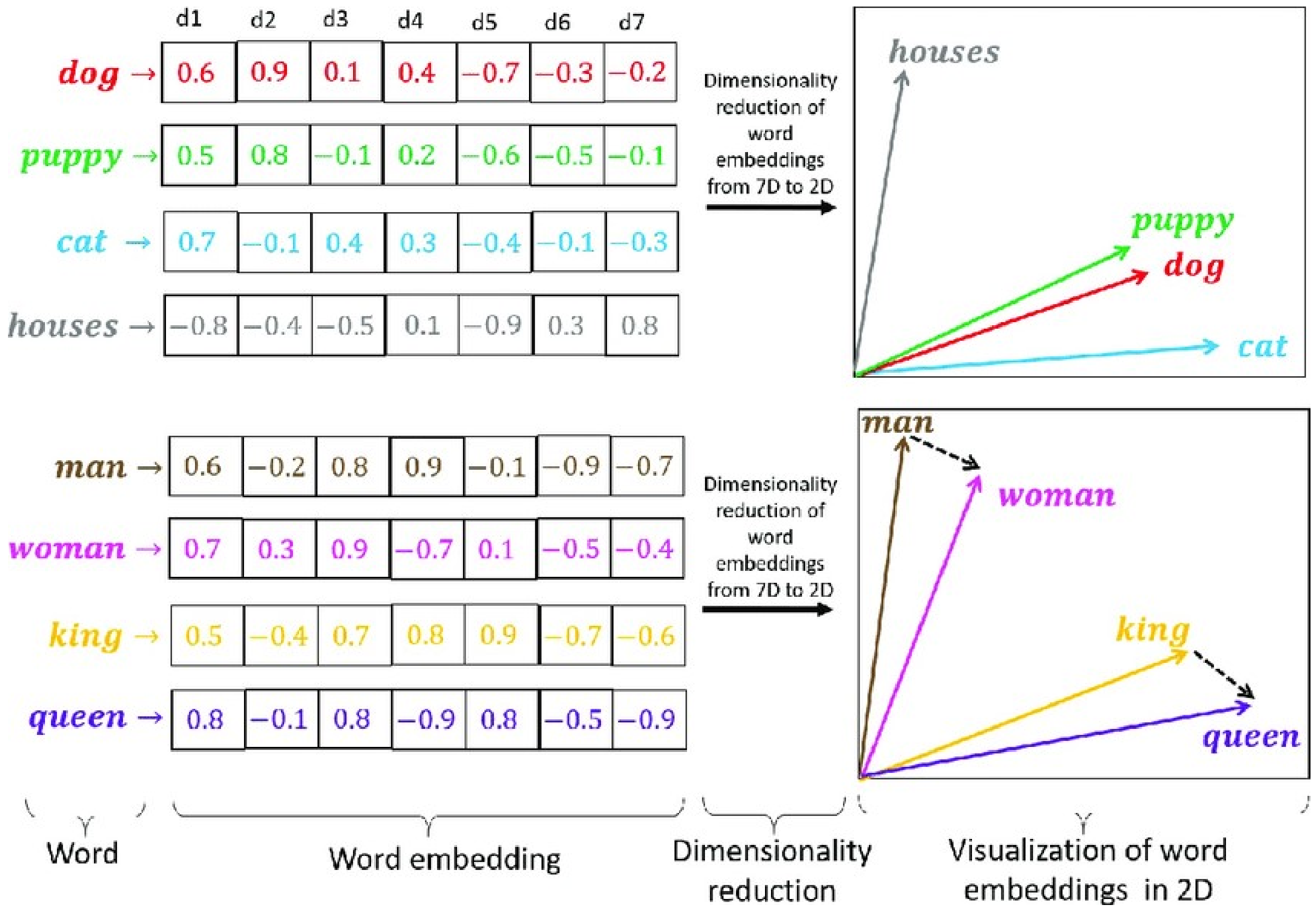
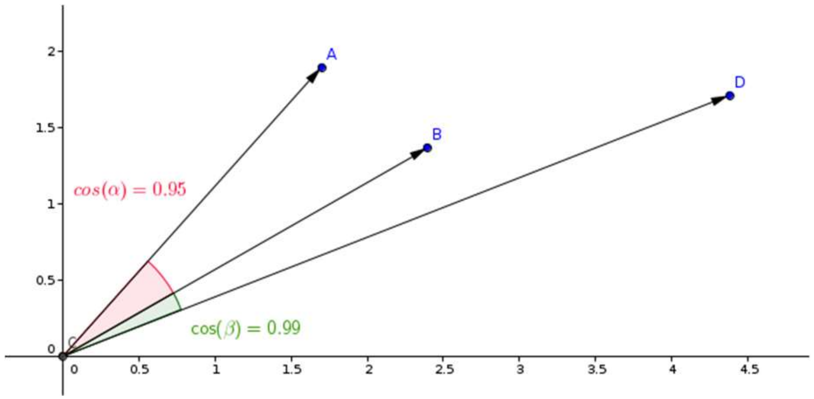
```python
# similarity search
query = "tell me a fact about ostriches"

docs = vectordb.similarity_search_with_score(query,k=3)

for result in docs:
       print("\n")
       print(result[1])
       print(result[0].page_content)
```





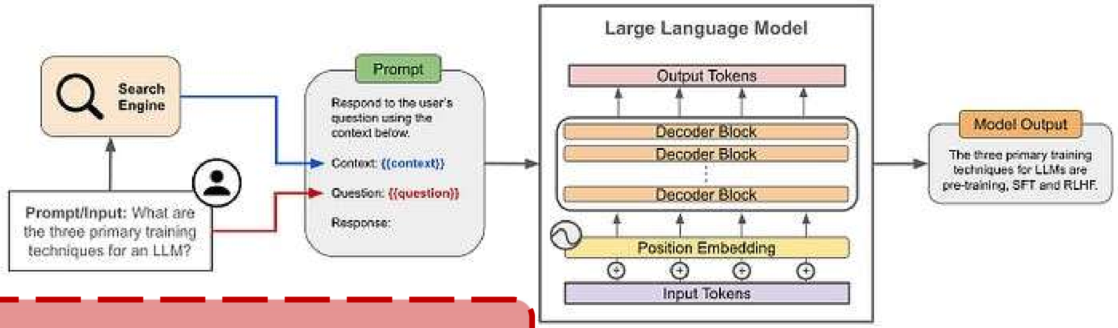

```
0.3597276056751038
101. Avocado has more protein than any other fruit.
102. Ostriches can run faster than horses.
103. The Golden Poison Dart Frog's skin has enough toxins to kill 100 people.


0.3783363542359751
1. "Dreamt" is the only English word that ends with the letters "mt."
2. An ostrich's eye is bigger than its brain.
3. Honey is the only natural food that is made without destroying any kind of life


0.4047550216944485
54. The kangaroo and the emu are featured on the Australian coat of arms because ne
```

Para cada documento, também retornaremos a distância. Quanto menor o número, mais relevante é o documento.


d1 d2 d3 d4 d5 d6 d7
dog → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
puppy → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
cat → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
houses
puppy
dog
cat
man
woman
king
queen
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

A
B
D
$cos(\alpha) = 0.95$
$\cos(\beta) = 0.99$

Monte o prompt

```
# Build prompt
template = """Use the following pieces of context to answer the question at
the end. If you don't know the answer, just say that you don't know, don't
try to make up an answer.
{context}
Question: {question}
Helpful Answer:"""

QA_CHAIN_PROMPT = PromptTemplate.from_template(template)
```



Uma vez que recuperamos fragmentos textuais relevantes, a etapa final do RAG é inserir esses fragmentos no prompt de um modelo de linguagem e gerar uma saída;

Carregue a LLM pré treinada

```
# Q&A
llm = ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0)
```

Fonte: https://jorgepit-14189.medium.com/get-started-with-chroma-db-and-retrieval-models-using-langchain-87784ffaa918

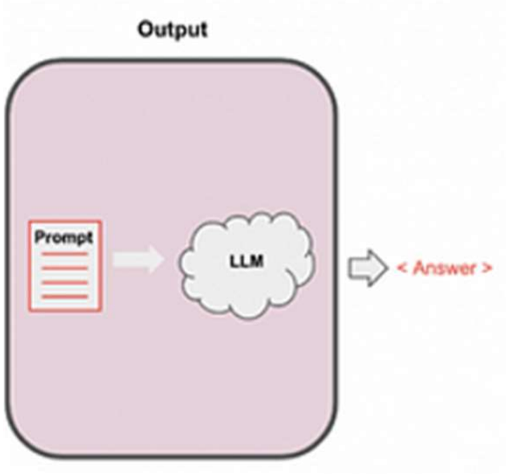
Monta o chain de execução (retrieval)

```
qa_chain = RetrievalQA.from_chain_type(
    llm,
    retriever=vectordb.as_retriever(),
    return_source_documents=True,
    chain_type_kwargs={"prompt": QA_CHAIN_PROMPT}
)
```

Retrieval QA Chain — ConversationalRetrievalChain: Integra o sistema de recuperação com o modelo de chat, formando uma cadeia que pode responder perguntas usando tanto as informações recuperadas quanto as capacidades do LLM.

Execute o LLM com os documentos

```
question = "tell me a fact about ostriches"
result = qa_chain({"question": question})

print(result["result"])
print(f"Question: {question}\nAnswer: {result["result"]}\n")
```



Mantenha a conversação na memória

```
memory = ConversationBufferMemory(
          memory_key="chat_history",
          k = 5,
          return_messages=True
)

qa = ConversationalRetrievalChain.from_llm(
          llm,
          retriever=vectordb.as_retriever(),
          memory=memory
)

question = "tell me a fact about ostriches"
result = qa({"question": question})
print(result['answer'])

question = "what is the maximum speed that they can reach?"
result = qa({"question": question})
print(result['answer'])
```

Conversational Memory — ConversationBufferWindowMemory: Mantém um buffer de memória do histórico da conversa para melhorar o contexto e a relevância das respostas.

Retrieval QA Chain — ConversationalRetrievalChain: Integra o sistema de recuperação com o modelo de chat, formando uma cadeia que pode responder perguntas usando tanto as informações recuperadas quanto as capacidades do LLM.