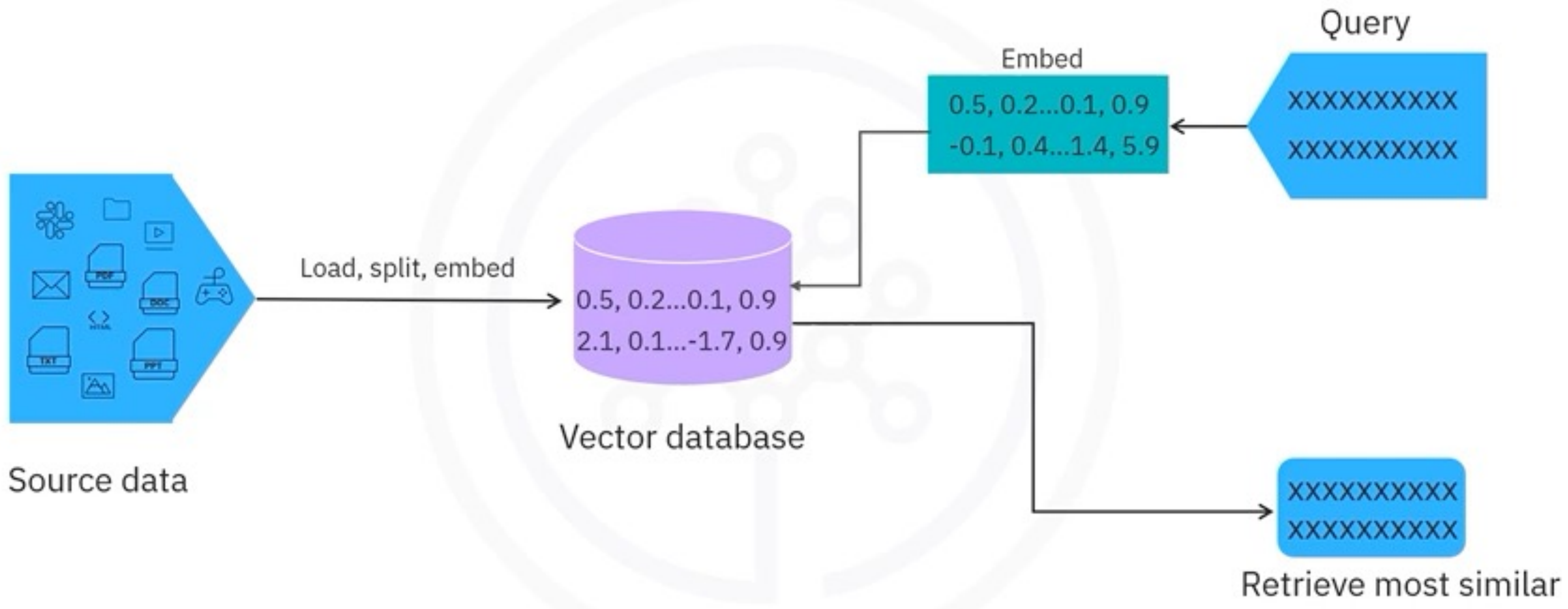
How a vector store works



How a vector store works

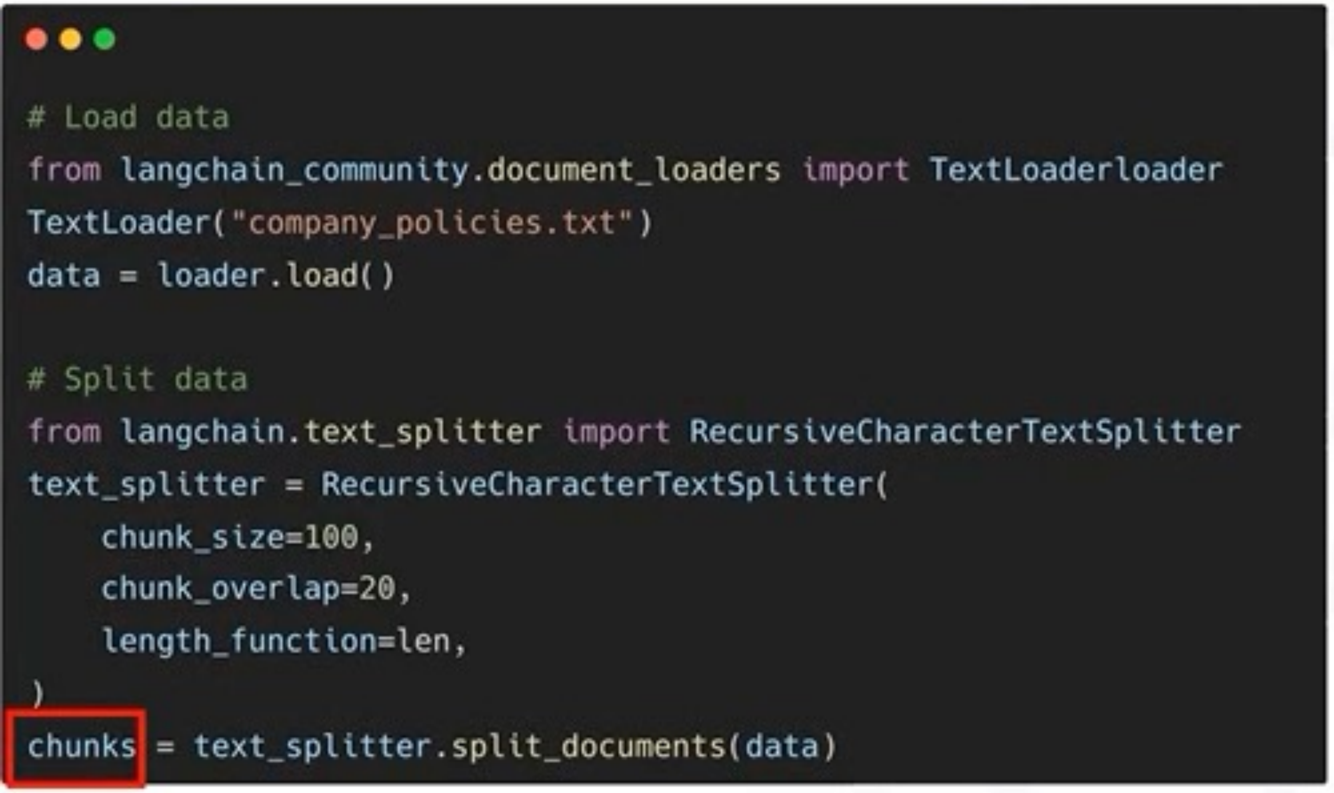
Vector databases

- Embeddings convert data into numerical vector formats within a high-dimensional space
- Can index and search for similar vectors using sophisticated similarity algorithms
- Allow applications to find related vectors, enabling effective information retrieval

Traditional databases

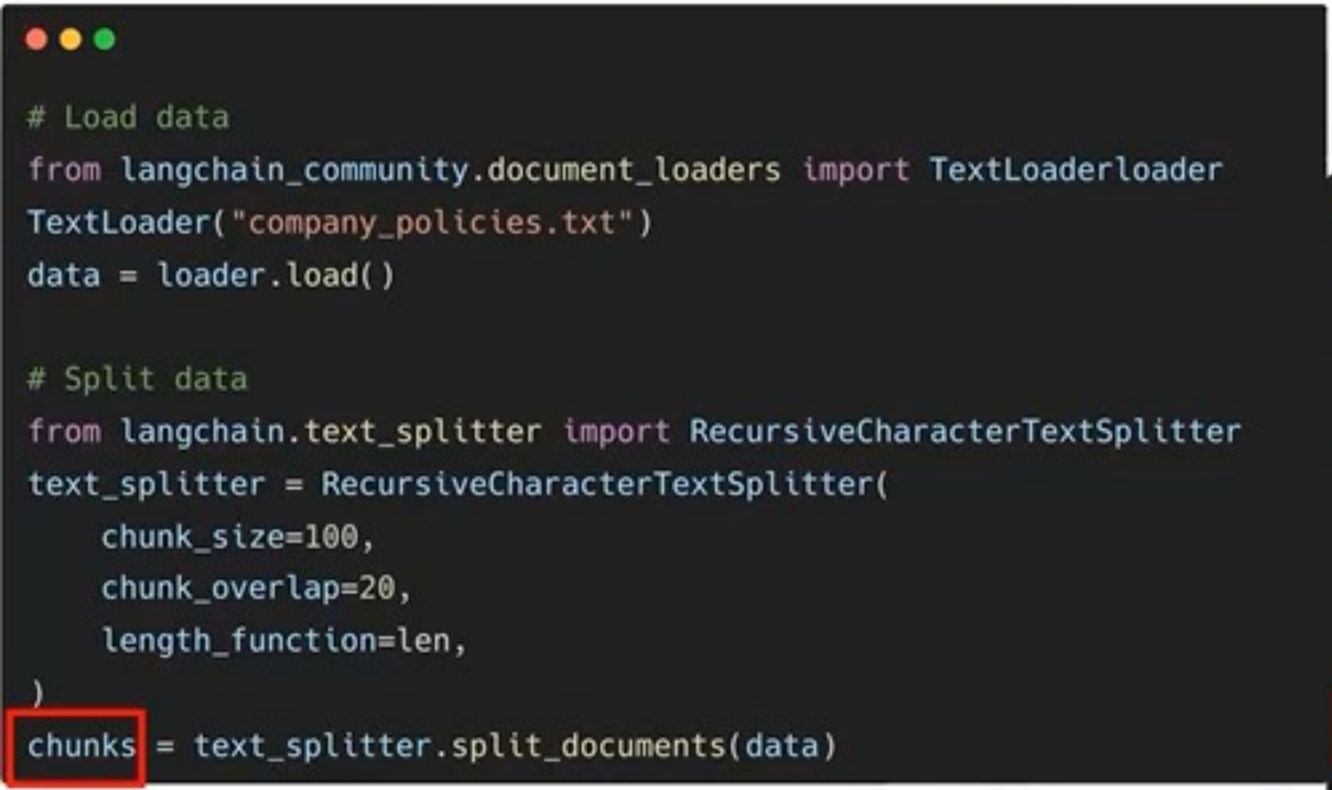
- Not optimized for storing and querying large vector data
- Struggle with storing and searching for vector representations

Chroma DB using LangChain

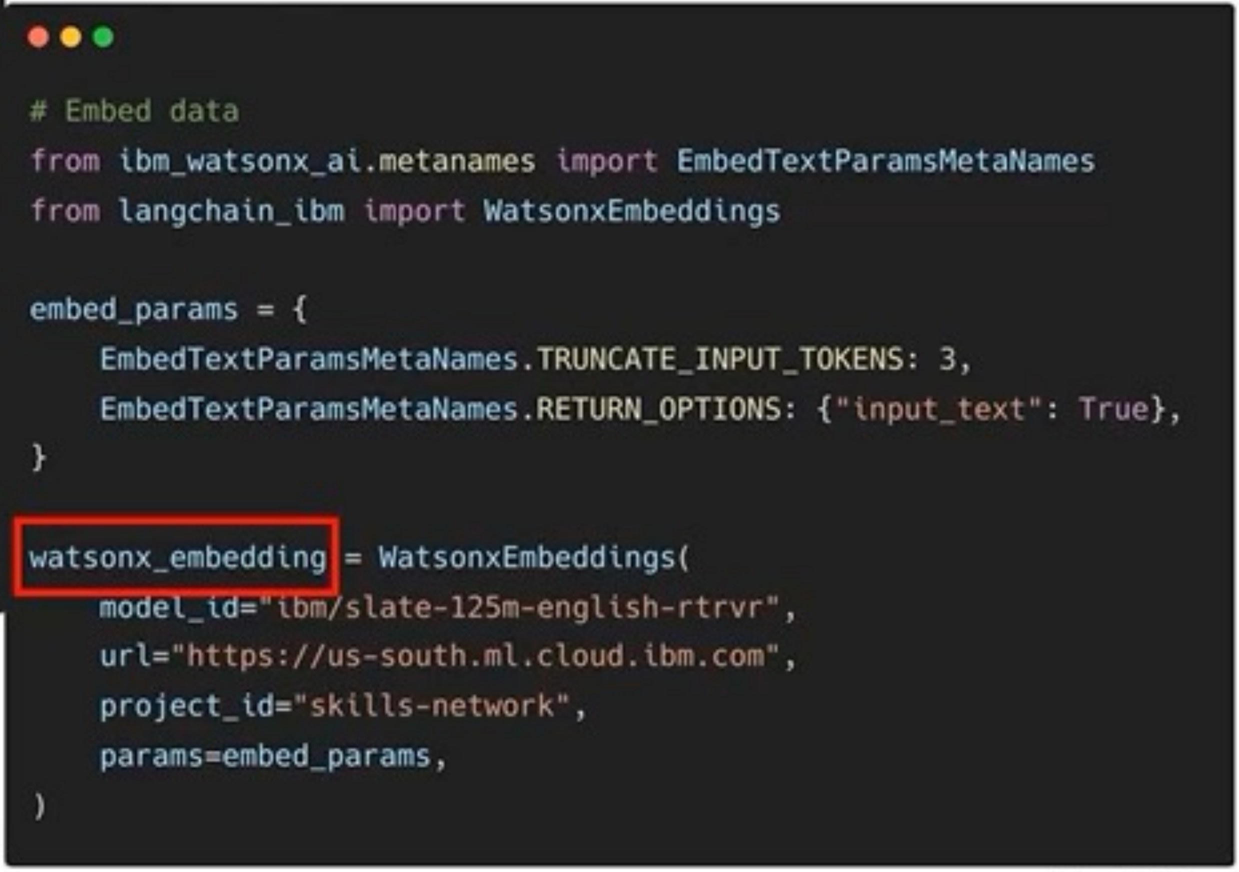


```python
# Load data
from langchain_community.document_loaders import TextLoaderloader
TextLoader("company_policies.txt")
data = loader.load()

# Split data
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=20,
    length_function=len,
)
chunks = text_splitter.split_documents(data)
```

Chroma DB using LangChain

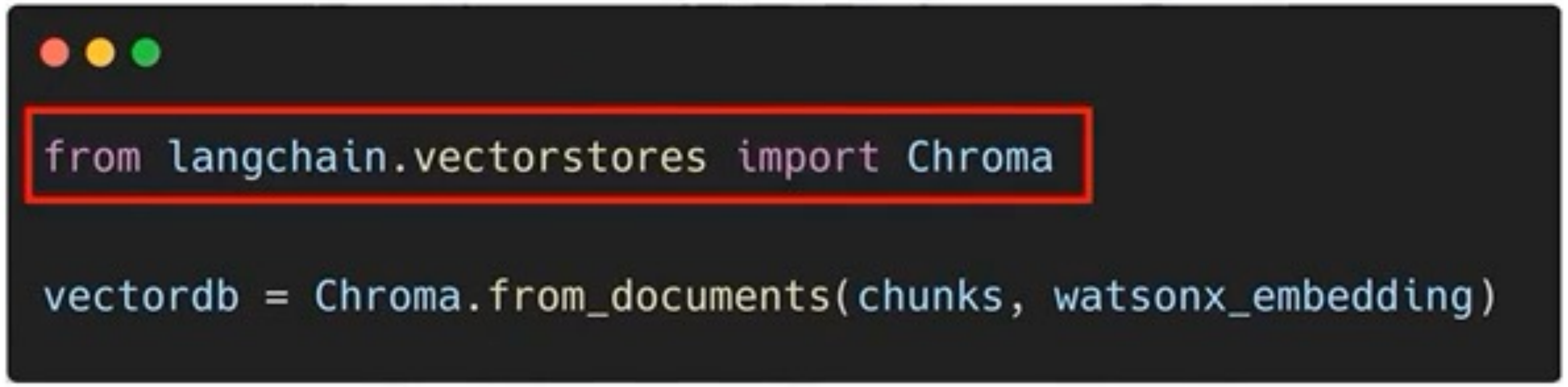


```python
# Load data
from langchain_community.document_loaders import TextLoaderloader
TextLoader("company_policies.txt")
data = loader.load()

# Split data
from langchain.text_splitter import RecursiveCharacterTextSplitter
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=100,
    chunk_overlap=20,
    length_function=len,
)
chunks = text_splitter.split_documents(data)
```



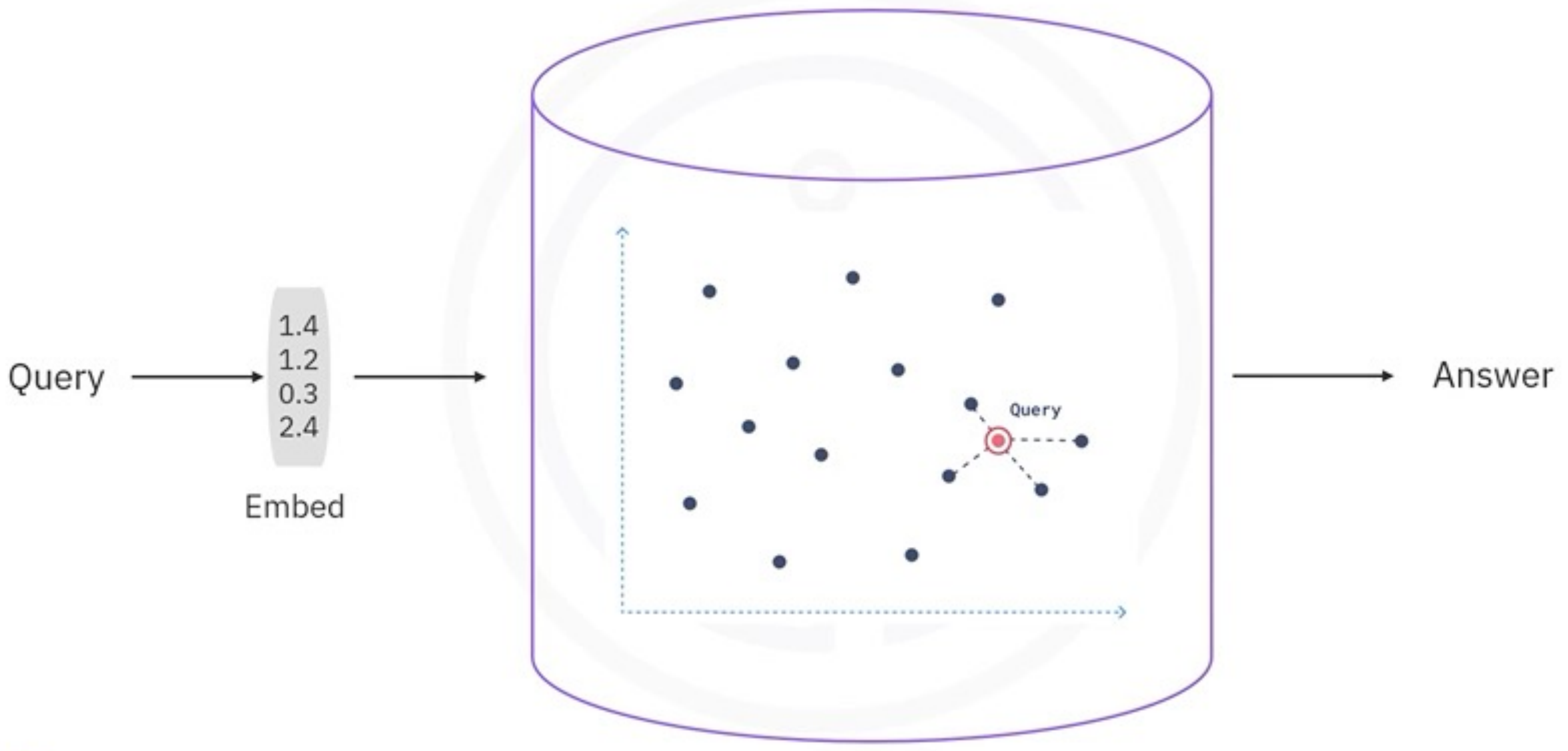
```python
# Embed data
from ibm_watsonx_ai.metanames import EmbedTextParamsMetaNames
from langchain_ibm import WatsonxEmbeddings

embed_params = {
    EmbedTextParamsMetaNames.TRUNCATE_INPUT_TOKENS: 3,
    EmbedTextParamsMetaNames.RETURN_OPTIONS: {"input_text": True},
}

watsonx_embedding = WatsonxEmbeddings(
    model_id="ibm/slate-125m-english-rtrvr",
    url="https://us-south.ml.cloud.ibm.com",
    project_id="skills-network",
    params=embed_params,
)
```

Chroma DB using LangChain



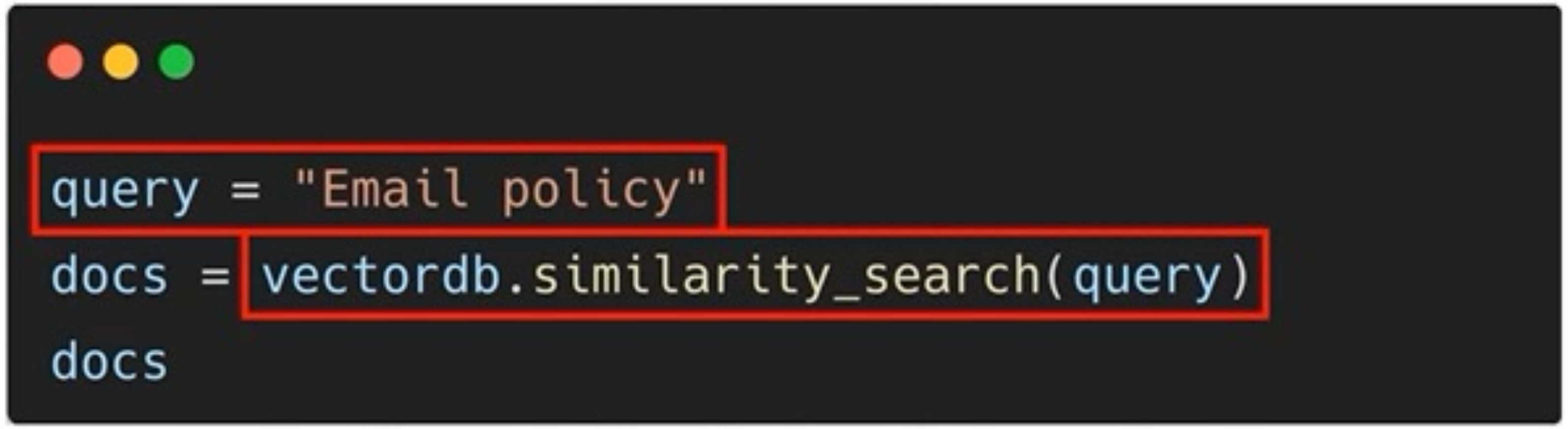
```python
from langchain.vectorstores import Chroma

vectordb = Chroma.from_documents(chunks, watsonx_embedding)
```

Similarity search



Vector database

Retrieval by similarity



```
query = "Email policy"
docs = vectordb.similarity_search(query)
docs
```

```
[Document(metadata={'source': 'company_policies.txt'}, page_content='to this policy. Non-
compliance may lead to appropriate disciplinary action, which could include'),
Document(metadata={'source': 'company_policies.txt'}, page_content='This policy serves as
a framework for handling discipline and termination. The organization'),
Document(metadata={'source': 'company_policies.txt'}, page_content='Policy Purpose: The
Smoking Policy has been established to provide clear guidance and expectations'),
Document(metadata={'source': 'company_policies.txt'}, page_content='Policy Objective: The
Drug and Alcohol Policy is established to establish clear expectations and')]
```

What is a LangChain retriever?

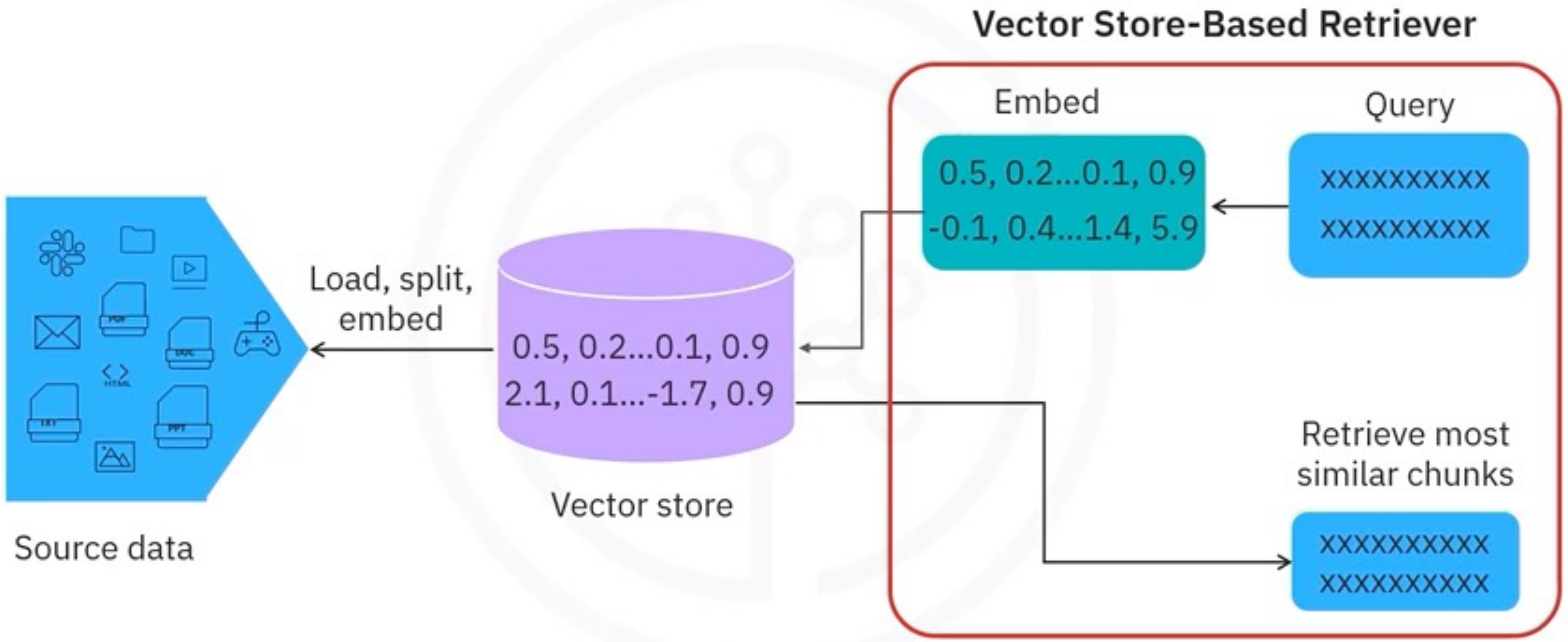
An interface that returns documents based on an unstructured query

More general than a vector store

Retrieves documents or their chunks

Accepts a string query as input and returns a list of documents or chunks as output

Vector Store-Based Retriever



Vector Store-Based Retriever

Simple to understand because it queries an existing vector store

Does not require an LLM to retrieve the most similar chunks

Vector Store-Based Retriever: Maximum marginal relevance (MMR) retrieval

Technique used to balance the relevance and diversity of retrieved results

Selects documents relevant to the query and minimally similar to previous documents

Avoids redundancy and ensures comprehensive coverage of query

Vector Store-Based Retriever: Maximum marginal relevance (MMR) retrieval

```python
retriever = vectordb.as_retriever(search_type="mmr")
docs = retriever.invoke("email policy")
```

```
[Document(metadata={'source': 'companypolicies.txt'}, page_content='to this policy. Non-compliance may lead to appropriate disciplinary action,
which could include'),
 Document(metadata={'source': 'companypolicies.txt'}, page_content='of our employees, customers, and the public. We diligently comply with all r
elevant health and'),
 Document(metadata={'source': 'companypolicies.txt'}, page_content="Data Privacy: We are committed to protecting the privacy of candidates' pers
onal information and"),
 Document(metadata={'source': 'companypolicies.txt'}, page_content='workplace safety measures.')]
```

Multi-Query Retriever

A LangChain retriever returns documents based on unstructured query

A Vector Store-Based Retriever retrieves documents from a vector database

Multi-Query Retriever

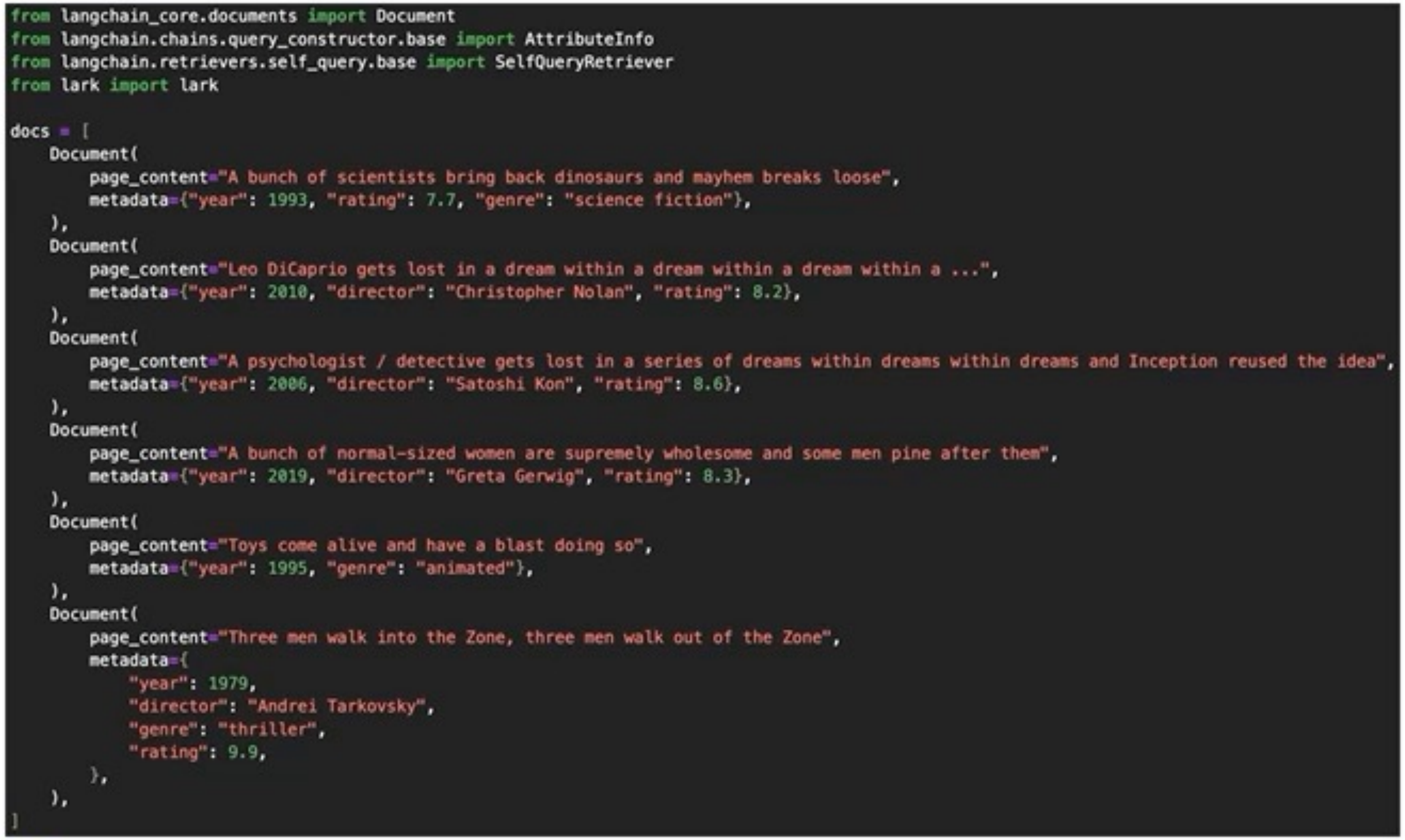
- Uses an LLM to create different versions of the query to generate a richer set of documents
- Overcomes differences in results due to changes in query wording or poor embeddings

Multi-Query Retriever

```python
from ibm_watsonx_ai.foundation_models import ModelInference
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai import Credentials
from ibm_watsonx_ai.foundation_models.extensions.langchain import WatsonxLLM

def llm():
    model_id = 'mistralai/mixtral-8x7b-instruct-v01'
    parameters = {
        GenParams.MAX_NEW_TOKENS: 256,  # this controls the maximum number of tokens in the generated output
        GenParams.TEMPERATURE: 0.5, # this randomness or creativity of the model's responses
    }
    credentials = {
        "url": "https://us-south.ml.cloud.ibm.com"
    }
    project_id = "skills-network"
    model = ModelInference(
        model_id=model_id,
        params=parameters,
        credentials=credentials,
        project_id=project_id
    )
    mixtral_llm = WatsonxLLM(model = model)
    return mixtral_llm
```

```python
from langchain.retrievers.multi_query import MultiQueryRetriever

retriever = MultiQueryRetriever.from_llm(
    retriever=vectordb.as_retriever(), llm=llm()
)

docs = retriever.invoke("email policy")
```

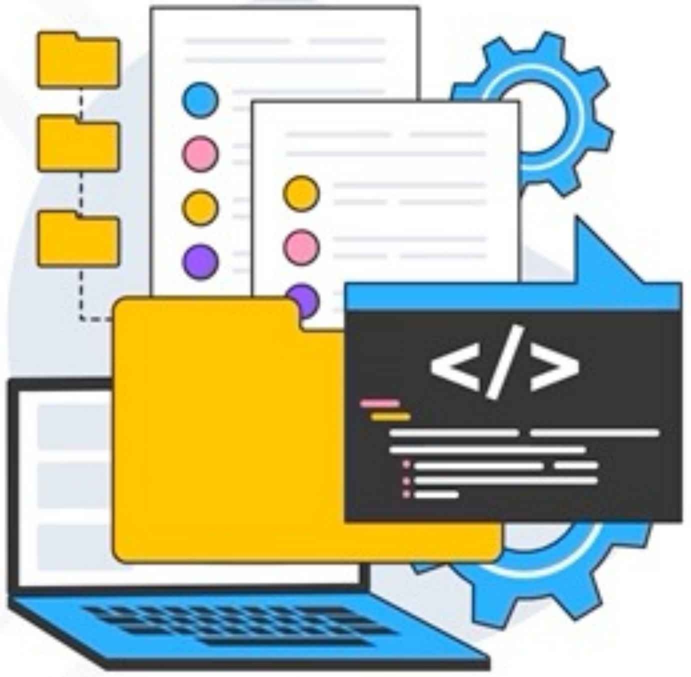
Self-Query Retriever



```python
from langchain_core.documents import Document
from langchain.chains.query_constructor.base import AttributeInfo
from langchain.retrievers.self_query.base import SelfQueryRetriever
from lark import lark

docs = [
    Document(
        page_content="A bunch of scientists bring back dinosaurs and mayhem breaks loose",
        metadata={"year": 1993, "rating": 7.7, "genre": "science fiction"},
    ),
    Document(
        page_content="Leo DiCaprio gets lost in a dream within a dream within a dream within a ...",
        metadata={"year": 2010, "director": "Christopher Nolan", "rating": 8.2},
    ),
    Document(
        page_content="A psychologist / detective gets lost in a series of dreams within dreams within dreams and Inception reused the idea",
        metadata={"year": 2006, "director": "Satoshi Kon", "rating": 8.6},
    ),
    Document(
        page_content="A bunch of normal-sized women are supremely wholesome and some men pine after them",
        metadata={"year": 2019, "director": "Greta Gerwig", "rating": 8.3},
    ),
    Document(
        page_content="Toys come alive and have a blast doing so",
        metadata={"year": 1995, "genre": "animated"},
    ),
    Document(
        page_content="Three men walk into the Zone, three men walk out of the Zone",
        metadata={
            "year": 1979,
            "director": "Andrei Tarkovsky",
            "genre": "thriller",
            "rating": 9.9,
        },
    ),
]
```

Self-Query Retriever

- Converts the query into two components:
  - A string to look up semantically
  - Metadata filter to go along with it



Self-Query Retriever: Setup

```
vectordb = Chroma.from_documents(docs, watsonx_embedding)
```

```
metadata_field_info = [
    AttributeInfo(
        name="genre",
        description="The genre of the movie. One of ['science fiction', 'comedy', 'drama', 'thriller', 'romance', 'action', 'animated']",
        type="string",
    ),
    AttributeInfo(
        name="year",
        description="The year the movie was released",
        type="integer",
    ),
    AttributeInfo(
        name="director",
        description="The name of the movie director",
        type="string",
    ),
    AttributeInfo(
        name="rating", description="A 1-10 rating for the movie", type="float"
    ),
]
```

Self-Query Retriever: Usage

```
document_content_description = "Brief summary of a movie."

retriever = SelfQueryRetriever.from_llm(
    llm(),
    vectordb,
    document_content_description,
    metadata_field_info,
)

retriever.invoke("I want to watch a movie rated higher than 8.5")
```

```
[Document(metadata={'director': 'Andrei Tarkovsky', 'genre': 'thriller', 'rating': 9.9, 'year': 1979}, page_content='Three men walk into the Zon
e, three men walk out of the Zone'),
 Document(metadata={'director': 'Satoshi Kon', 'rating': 8.6, 'year': 2006}, page_content='A psychologist / detective gets lost in a series of d
reams within dreams within dreams and Inception reused the idea')]
```

Parent Document Retriever

- Splitting documents involves conflict between small documents for accuracy, and long for context
- Parent Document Retriever fetches small chunks, looks up their parent IDs, and returns large documents of the small chunks

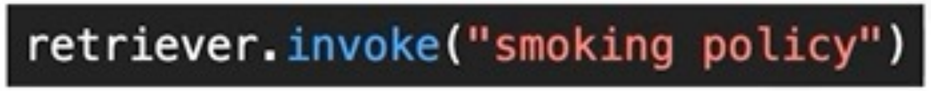
Parent Document Retriever: Setup

```python
from langchain.retrievers import ParentDocumentRetriever
from langchain_text_splitters import CharacterTextSplitter
from langchain.storage import InMemoryStore

# Set two splitters. One is with big chunk size (parent) and one is with small chunk size (child)
parent_splitter = CharacterTextSplitter(chunk_size=2000, chunk_overlap=20, separator='\n')
child_splitter = CharacterTextSplitter(chunk_size=400, chunk_overlap=20, separator='\n')
```

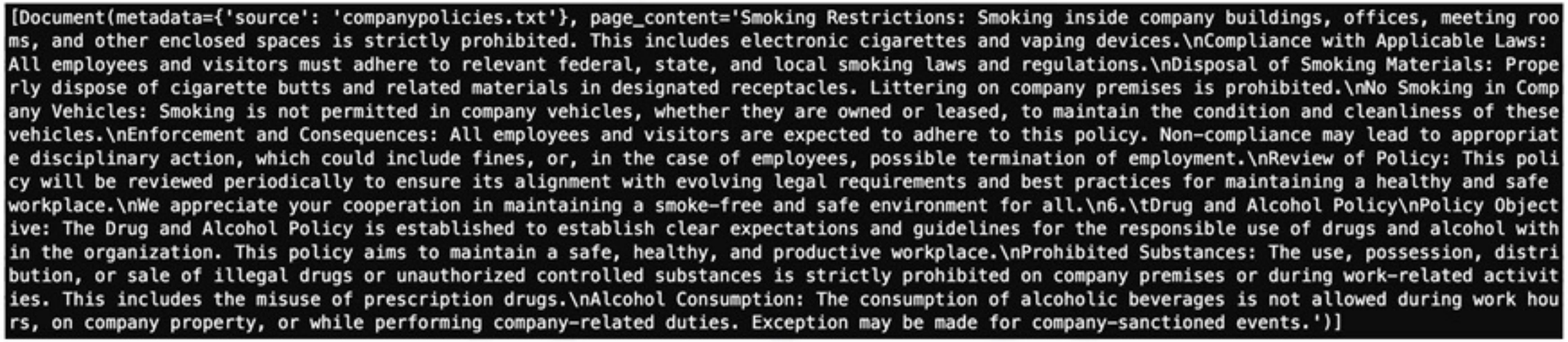
```python
vectordb = Chroma(
    collection_name="split_parents", embedding_function=watsonx_embedding
)

# The storage layer for the parent documents
store = InMemoryStore()
```

```python
retriever = ParentDocumentRetriever(
    vectorstore=vectordb,
    docstore=store,
    child_splitter=child_splitter,
    parent_splitter=parent_splitter,
)
retriever.add_documents(data)
```

Parent Document Retriever: Usage



```python
retriever.invoke("smoking policy")
```



```
[Document(metadata={'source': 'companypolicies.txt'}, page_content='Smoking Restrictions: Smoking inside company buildings, offices, meeting roo
ms, and other enclosed spaces is strictly prohibited. This includes electronic cigarettes and vaping devices.\nCompliance with Applicable Laws: 
All employees and visitors must adhere to relevant federal, state, and local smoking laws and regulations.\nDisposal of Smoking Materials: Prope
rly dispose of cigarette butts and related materials in designated receptacles. Littering on company premises is prohibited.\nNo Smoking in Comp
any Vehicles: Smoking is not permitted in company vehicles, whether they are owned or leased, to maintain the condition and cleanliness of these
vehicles.\nEnforcement and Consequences: All employees and visitors are expected to adhere to this policy. Non-compliance may lead to appropriat
e disciplinary action, which could include fines, or, in the case of employees, possible termination of employment.\nReview of Policy: This poli
cy will be reviewed periodically to ensure its alignment with evolving legal requirements and best practices for maintaining a healthy and safe 
workplace.\nWe appreciate your cooperation in maintaining a smoke-free and safe environment for all.\n6.\tDrug and Alcohol Policy\nPolicy Object
ive: The Drug and Alcohol Policy is established to establish clear expectations and guidelines for the responsible use of drugs and alcohol with
in the organization. This policy aims to maintain a safe, healthy, and productive workplace.\nProhibited Substances: The use, possession, distri
bution, or sale of illegal drugs or unauthorized controlled substances is strictly prohibited on company premises or during work-related activit
ies. This includes the misuse of prescription drugs.\nAlcohol Consumption: The consumption of alcoholic beverages is not allowed during work hou
rs, on company property, or while performing company-related duties. Exception may be made for company-sanctioned events.')]
```

Recap

- The Multi-Query Retriever uses an LLM to create different versions of the query to generate richer documents
- The Self-Query Retriever converts the query into a string and a metadata filter
- The Parent Document Retriever has a parent splitter to split text into large chunks and a child splitter for small chunks