datakami
research & development

Building a personal search engine with llama-index

Judith van Stegeren & Yorick van Pelt

Before this tutorial

1. Download the files from https://github.com/datakami/pydata-llama-index-tutorial/
2. Follow the readme to get the requirements
    a. Local Python environment
    b. Docker
    c. Google Colab

Tutorial (1 hour)

Part 1: searching with llama-index

Notebook 1: retrieval and querying

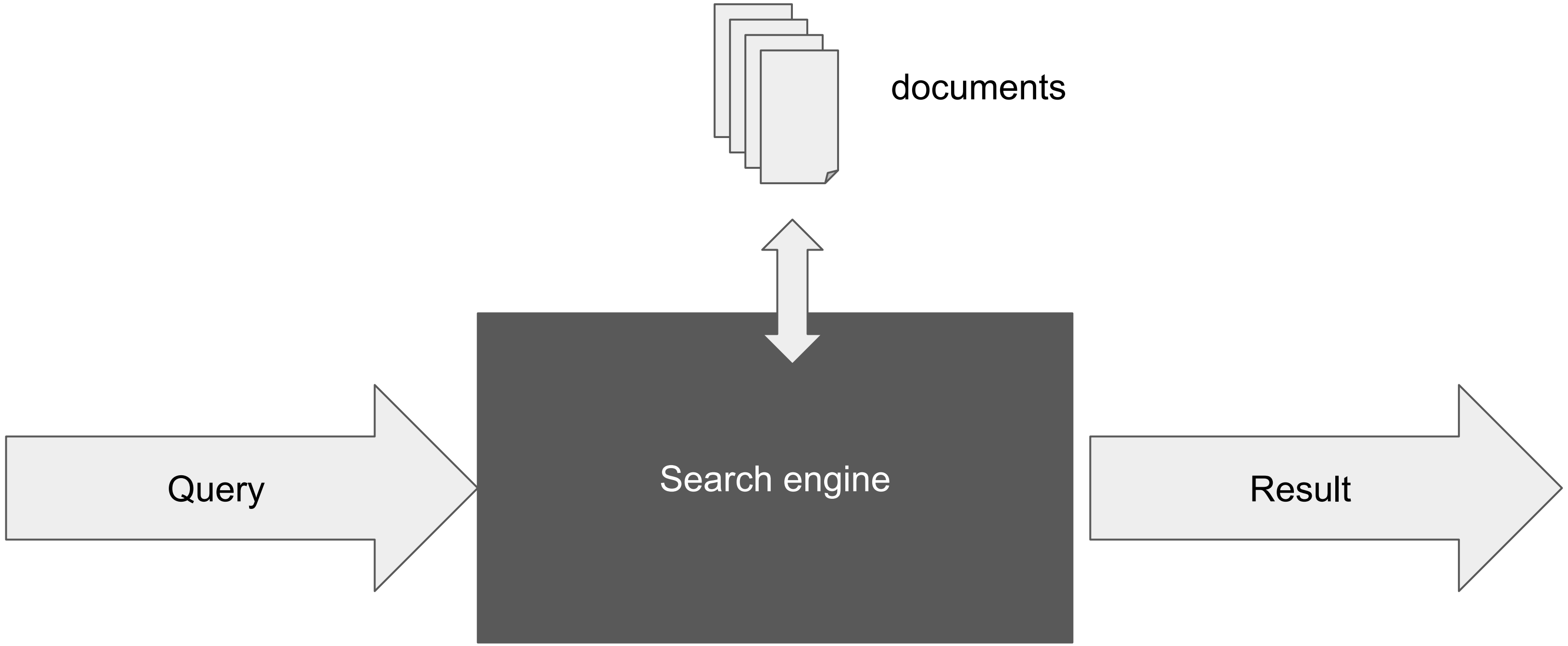
Part 2: building a search engine from your own documents

Notebook 2: building an index from text documents

Wrap-up: things to explore after this tutorial

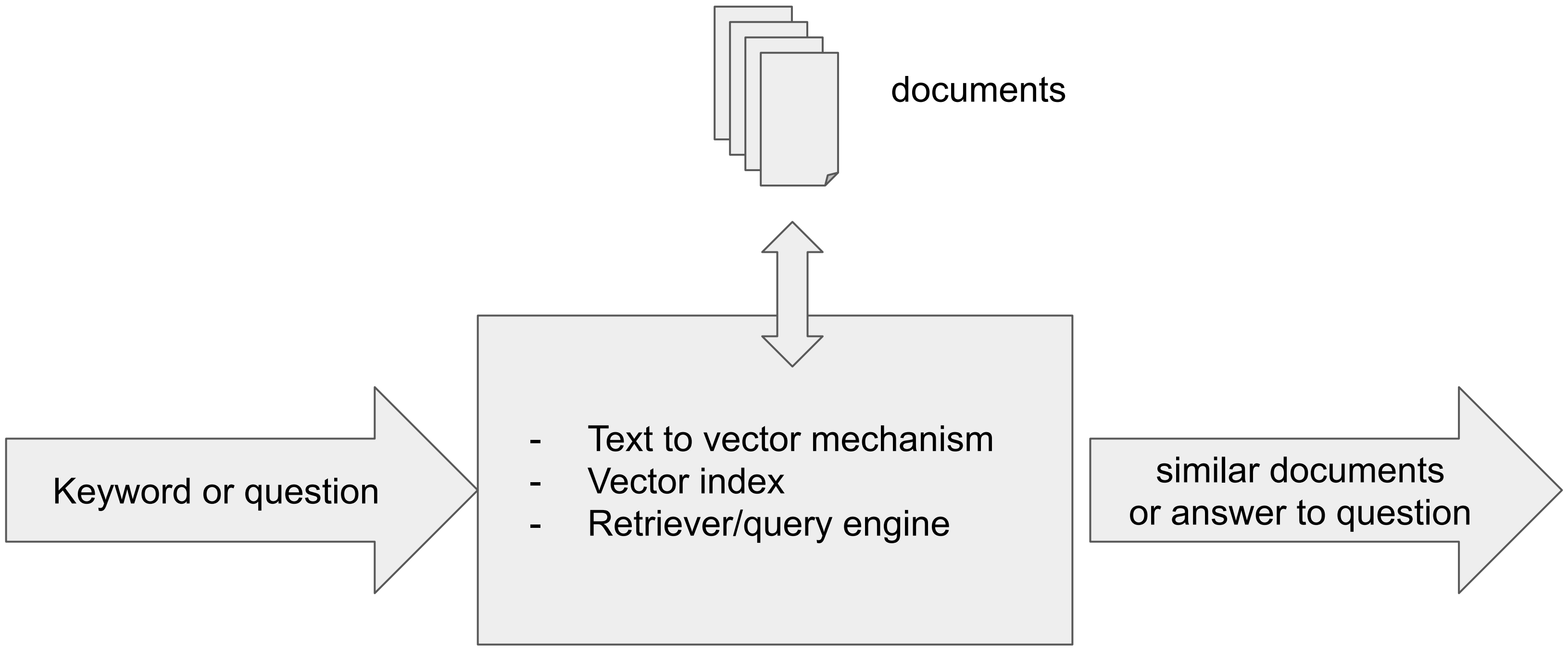
Part 1: searching with llama-index

Personal search engine



import llama_index

Personal search engine



What is this “vector store index” thing?

Vector: row of numbers

Vector store: *thing* that stores vectors

Index: a list of items, each of which identifies a particular record in a computer file or database and contains information about its address

Retrieving vs querying

Retrieval

Ingredients: index, query

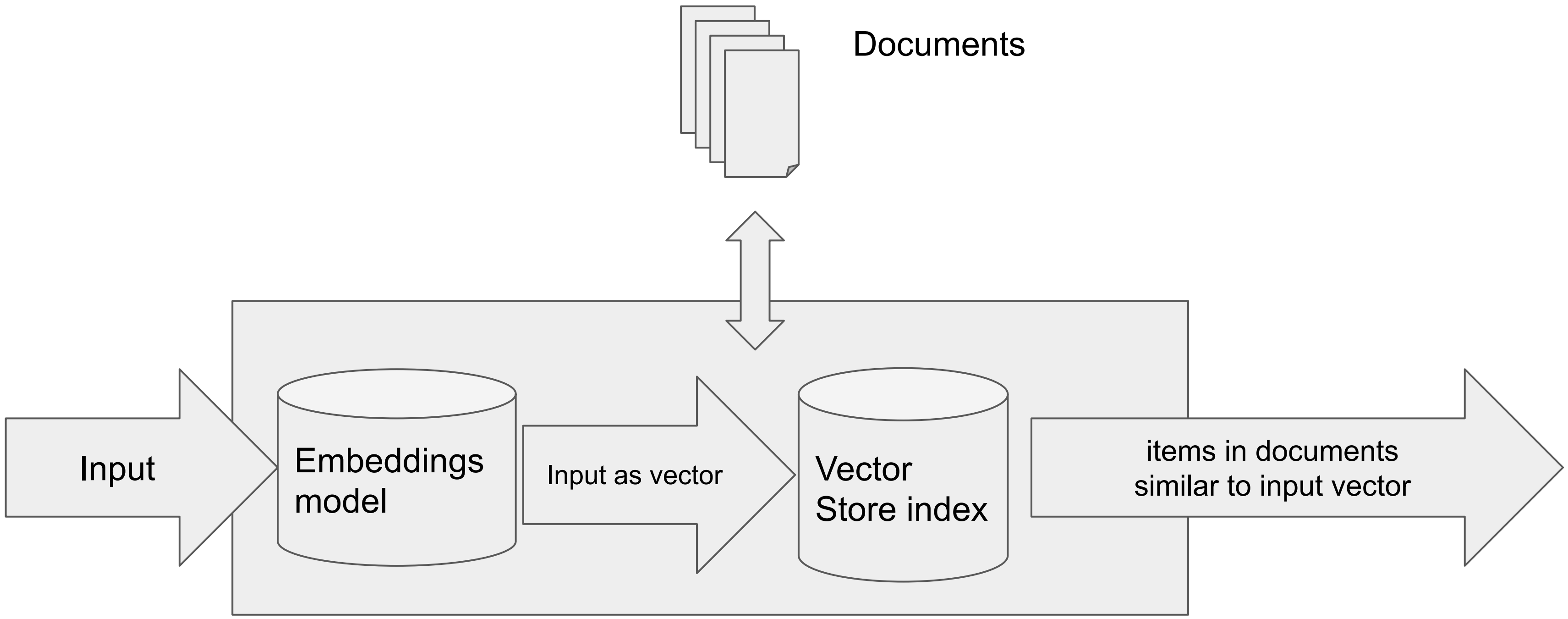
Result: most "similar" things in the index

Querying

Ingredients: index, query, context

Result: LLM uses the context + retrieval result to write an answer to the query

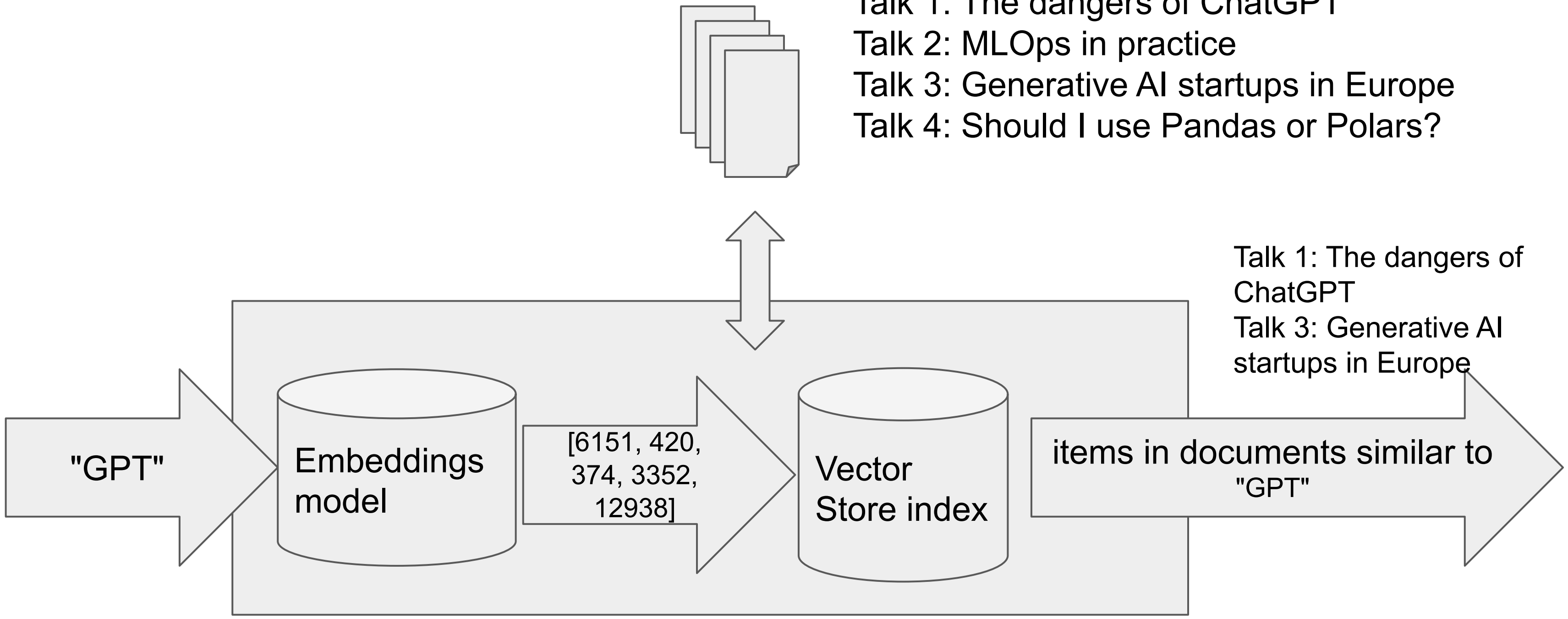
Retrieval


Documents
Input
Embeddings model
Input as vector
Vector Store index
items in documents similar to input vector

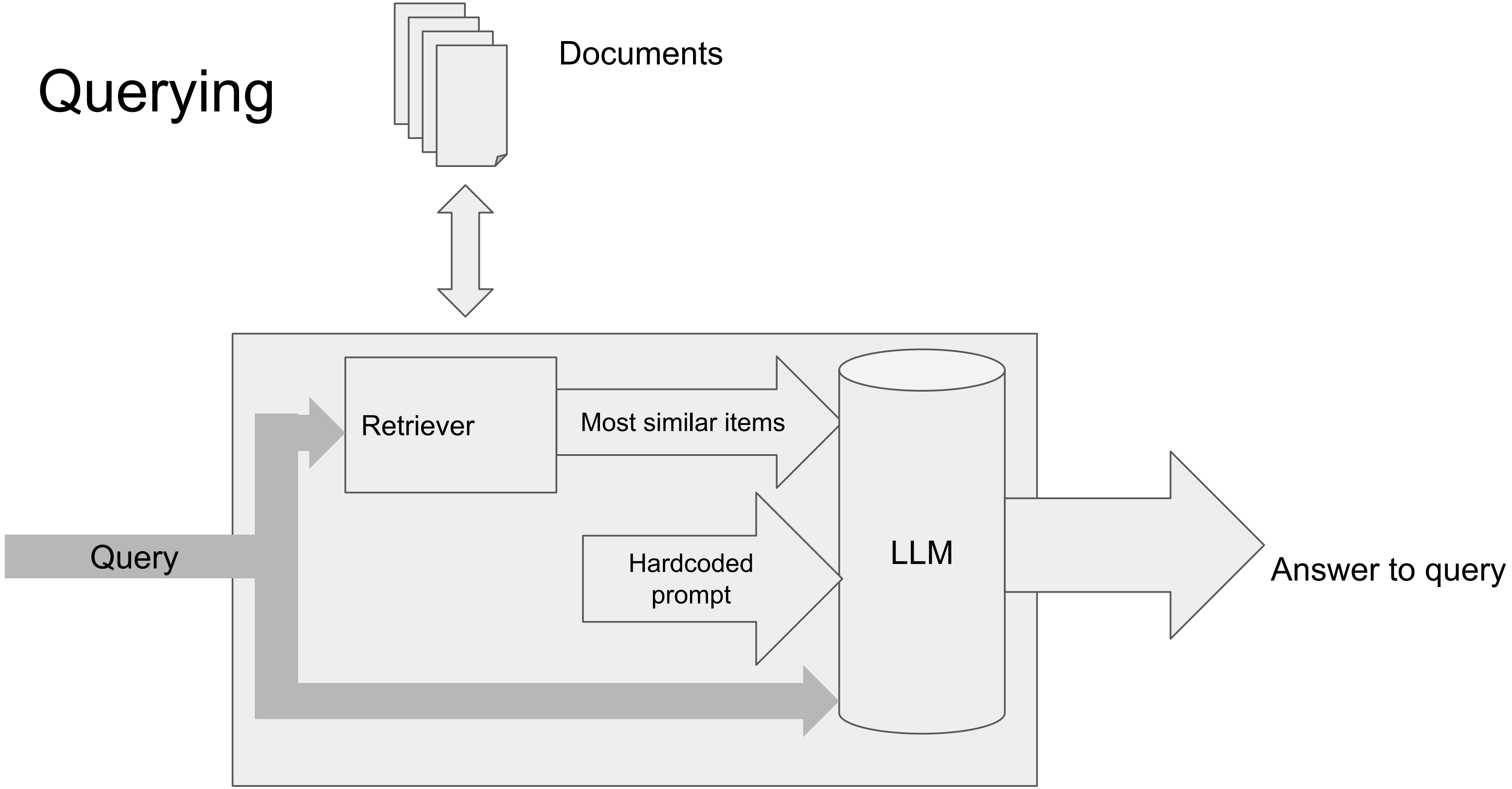
Retrieval

Documents

Talk 1: The dangers of ChatGPT
Talk 2: MLOps in practice
Talk 3: Generative AI startups in Europe
Talk 4: Should I use Pandas or Polars?


"GPT"
Embeddings model
[6151, 420, 374, 3352, 12938]
Vector Store index
items in documents similar to "GPT"
Talk 1: The dangers of ChatGPT
Talk 3: Generative AI startups in Europe

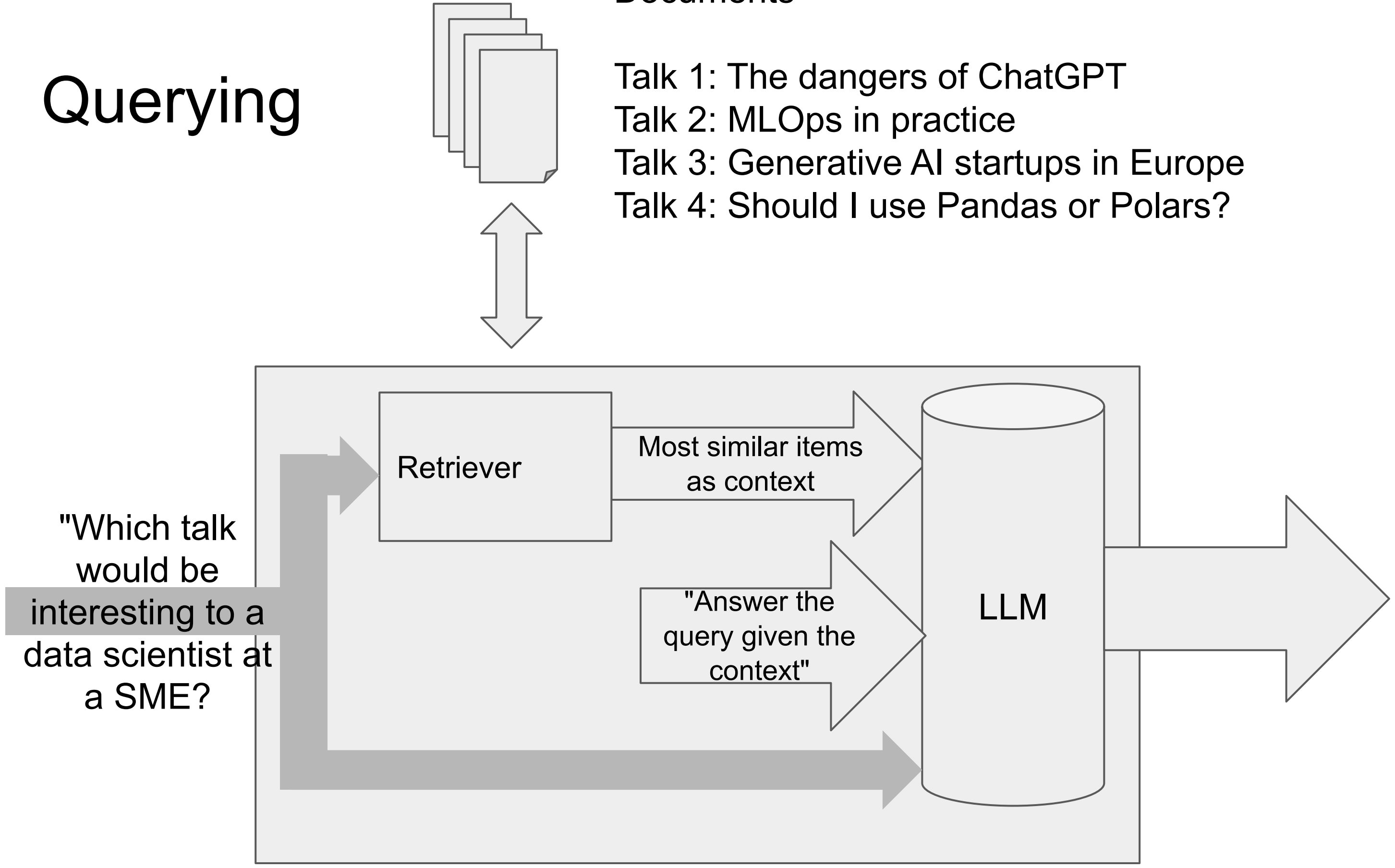
Querying


Documents
Retriever
Most similar items
Hardcoded prompt
LLM
Query
Answer to query

Querying

Documents

Talk 1: The dangers of ChatGPT
Talk 2: MLOps in practice
Talk 3: Generative AI startups in Europe
Talk 4: Should I use Pandas or Polars?



"Which talk would be interesting to a data scientist at a SME?
Retriever
Most similar items as context
"Answer the query given the context"
LLM

The talk titled "Should I use Pandas or Polars" is the most interesting talk for data scientists at a small or medium enterprise, because Pandas is a well-known library for data science applications (...)

Our first index

`index` is a vector store index with PyData Amsterdam 2023 talks!

Index consists of document snippets called `nodes`.

Our first index

A `node` has `text` and `metadata`

```
node.text
```

```
"Building a personal search engine with llama-index\n\n
Wouldn’t it be great to have a Google-like search engine,
but then for your own text files and completely private?
In this tutorial (...)"
```

```
node.metadata
```

```
{'title': 'Building a personal search engine with llama-index',
'speakers': 'Judith van Stegeren, Yorick van Pelt'}
```

Retrieving with llama-index

`index` is a vector store index with the PyData schedule

```
retriever = index.as_retriever()

search_results = retriever.retrieve("your query here")
```

Dealing with search results

`index` is a vector store index with the PyData schedule

`search_results` is a list of retrieval results

A `result` has a `node` and a `score`

```
first_result = search_results[0]

first_result.node

first_result.score
```

Querying with llama-index

`index` is a vector store index with the PyData schedule

```
query_engine = index.as_query_engine()

search_results = query_engine.query("your query here")
```

Time for a notebook: **Exercises-1.ipynb**

`index` is a vector store index with the PyData schedule

```
node.text
node.metadata

retriever = index.as_retriever()
search_results = retriever.retrieve("your query here")

query_engine = index.as_query_engine()
search_results = query_engine.query("your query here")

result.node
result.score
```

Tutorial repository: https://github.com/datakami/pydata-llama-index-tutorial

Part 2: building a vector index

Recap: our first index

`index` is a vector store index with PyData Amsterdam 2023 talks!

Index consists of document snippets called nodes.

A `node` has `text` and `metadata`

Building a new vector store index

An index is created from a set of Documents.
Documents are split into Nodes (we’ve seen these before!)

Four steps:

1. Load data from source (vanilla Python)
2. Transform data to "Document" format
3. Put documents in vector store index
4. Save the vector store index to file

Step 1: loading data

from plaintext file, from json, from the web, etc.

All kinds of custom data loaders on https://llamahub.ai/. (Quality varies)

```
# plaintext
with open("my_research_notes.txt",'r') as infile:
    research_notes = infile.read()

# json
with open("movie_descriptions.json",'r') as infile:
    movies = json.loads(infile.read())

# from web
response = requests.get("httpx://myfavoriteapi.com/endpoint?query=somedata")
api_data = json.loads(response.text)
```

Step 2: creating Documents from data

Transform data to a structure that llama_index understands.

All Document properties are completely optional!

```
document = llama_index.Document() # empty document

Document(
      id_='82c3f6b8-4e64-4d76-9629-f51aaa7446a0',
      embedding=None,
      metadata={},
      excluded_embed_metadata_keys=[],
      excluded_llm_metadata_keys=[],
      relationships={},
      hash='44136fa355b3678a1146ad16f7e8649e94fb4fc21fe77e8310c060f61caaff8a',
      text='',
      start_char_idx=None,
      end_char_idx=None,
      text_template='{metadata_str}\n\n{content}',
      metadata_template='{key}: {value}',
      metadata_seperator='\n'
)
```

Step 2: creating Documents from data

Transform data to a structure that llama_index understands.

All Document properties are completely optional!

```
document = llama_index.Document() # empty document

Document(
      id_='82c3f6b8-4e64-4d76-9629-f51aaa7446a0',
      metadata={},
      text='',
)
```

Step 2: creating Documents from data

Simplest way to create a document: just add text.

```
simple_text = "Hello world"

document = llama_index.Document(text=simple_text)

Document(
      id_='993d3bfd-83df-4fb5-b6f7-5ea688df8032',
      metadata={},
      text='Hello world',
)
```

Step 2: creating Documents from data

You can add custom identifiers to Documents.

This is good for tracking Document sources!

Important: use `id_` instead of `id` (this will fail silently)

```
my_filename = "somefile.txt"
with open(my_filename, "r") as infile:
     my_text = infile.read()

document = llama_index.Document(text=my_text, id_=my_filename)

Document(
     id_="somefile.txt",
     metadata={},
     text="The contents of a file on my computer",
)
```

Step 2: creating Documents from data

Add metadata as dictionary

```
simple_text = "Hello world"
some_metadata = { "author" : "Judith", "created_at": datetime.now() }

document = llama_index.Document(text=simple_text, metadata=some_metadata)


Document(
     id_='70b4c14c-18e9-49c8-9265-01760b8d5c3d',
     metadata={
          'author': 'Judith',
          'created_at': datetime(2023, 9, 8, 11, 7, 55, 281502)
     },
     text='Hello world'
)
```

Step 3: creating vector index from data

```
documents = [Document(text=datapoint) for datapoint in data]

index = VectorStoreIndex.from_documents(documents)
```

And we have an index!

Now we can create a `retriever` or `query_engine` and do everything we saw in part 1.

Step 3: creating vector index from data

We can use a `ServiceContext` to change:

- the size of text in nodes, or chunk size
- the overlap between nodes
- the embeddings model for transforming text to numbers
- the language model used for querying

And lots of other things as well!

Step 3: creating vector index from data

```
my_service_context = ServiceContext.from_defaults()
my_documents = [Document(text=datapoint) for datapoint in data]

index = VectorStoreIndex.from_documents(documents=my_documents,
service_context=my_service_context)
```

In part 1 of this tutorial, we used a ServiceContext to

- use the smallest sentence-transformers embeddings model
- remove the default LLM so llama-index won’t call OpenAI’s GPT

Step 3: creating vector index from data

```
service_context = ServiceContext.from_defaults()

service_context = ServiceContext.from_defaults(llm=None)

# llama-index will download the model for us
my_model = "local:sentence-transformers/all-minilm-l6-v2"
service_context = ServiceContext.from_defaults(embed_model=my_model)

service_context = ServiceContext.from_defaults(chunk_size=10)
```

Step 4: saving the index to file

`StorageContext` is a utility class for storing indices, vectors, and nodes.

`index` is a VectorStoreIndex

```
index_path = 'indices/name_of_my_index'
index.storage_context.persist(index_path)
```

Step 4: saving the index to file

We can load an index from disk by recreating its StorageContext and ServiceContext.

```
index_path = 'indices/my_old_index'

storage_context = StorageContext.from_defaults(persist_dir=index_path)
service_context = ServiceContext.from_defaults(llm=my_llm)
my_old_index = load_index_from_storage(storage_context,
service_context=service_context)
```

Time for a notebook: **Exercises-2.ipynb**

`index` is a vector store index

```
node.text
node.metadata

index.as_retriever()
retriever.retrieve("your query here")

index.as_query_engine()
query_engine.query("your query here")

result.node
Result.score

Document(
      text="hello",
      id_="hello.txt",
      metadata={"author":"Judith"}
)

from llama_index.llms import OpenAI
my_llm = OpenAI(model="gpt-4")
```

```
my_embed_model =
"local:sentence-transformers/all-minilm-l6-v2"

ServiceContext.from_defaults(
      llm=my_llm, embed_model=my_embed_model,
      chunk_size=my_chunk_size
)

VectorStoreIndex.from_documents(
      documents=documents,
      service_context=my_service_context
)

index.storage_context.persist('my_path')

StorageContext.from_defaults(persist_dir=index_path)

load_index_from_storage(
      storage_context,
      service_context=service_context
)
```

Tutorial repository: https://github.com/datakami/pydata-llama-index-tutorial

What’s next?

datakami
research & development

Experiment with your own data and favorite models
Other stuff to change or optimize: chunking, other models, meta-data, hardcoded prompts, …

Use cases: research notes, diaries, CRM, improve other LLM pipelines, …

Please don’t use this for production
Defaults are slowly getting more sane over time

Want to understand *embeddings* better? https://vickiboykis.com/what_are_embeddings/

Tutorial repository: https://github.com/datakami/pydata-llama-index-tutorial