

December 15, 2023

SHOWCASE • LLM • DATA-PIPELINE



Run in browser



Get from Github

Always up-to-date data indexing pipeline

This showcase shows how to use Pathway to deploy a live data indexing pipeline, which can be queried like a typical vector store. However, under the hood, Pathway updates the index on each data change, always giving up-to-date answers.



Pathway Vectorstore enables building a document index on top of your documents without the complexity of ETL pipelines, managing different containers for storing, embedding, and serving. It allows for easy to manage, always up-to-date, LLM pipelines accessible using a RESTful API and with integrations to popular LLM toolkits such as Langchain and Llama-index.

In this article, we will use a simple document processing pipeline that:

1. Monitors several data sources (files, S3 folders, cloud storages) for data changes.
2. Parses, splits and embeds the documents.
3. Builds a vector index for the data.

We will connect to the index using a `VectorStore` client, which allows retrieval of semantically similar documents.

Prerequisites

Install the `pathway` package. You can also install the `unstructured` package to use the most powerful `unstructured.io` -based parser. Finally, install `pathway llm-app` to use the LLM model wrappers.

```
# !pip install pathway llm-app
```

Building the data pipeline

First, make sure you have an API key with an LLM provider such as OpenAI.

```
import getpass
import os

if "OPENAI_API_KEY" not in os.environ:
    os.environ["OPENAI_API_KEY"] = getpass.getpass("OpenAI API Key: ")
```

We will now assemble the data vectorization pipeline, using a simple UTF8 file parser, and a character splitter and embedder from the [llm-app](#).

First, we define the data sources. We use the files-based one for simplicity, but any supported `pathway connector`, such as [s3](#) or [Google Drive](#) will also work.

Then, we define the embedder and splitter.

Last, we assemble the data pipeline. We will start it running in a background thread to be able to query it immediately from the demonstration. Please note that in a production deployment, the server will run in another process, possibly on another machine. For the quick-start, we keep the server and client as different threads of the same Python process.

```

import pathway as pw
from pathway.xpacks.llm.embedders import OpenAIEmbedder
from pathway.xpacks.llm.splitters import TokenCountSplitter
from pathway.xpacks.llm.vector_store import VectorStoreClient, VectorStoreServer

# This creates a connector that reads a single file.
data_sources = []
data_sources.append(
    pw.io.fs.read(
        "./sample_documents",
        format="binary",
        mode="streaming",
        with_metadata=True,
    )
)

# This creates a connector that tracks files in Google Drive.
# Please follow the instructions at https://pathway.com/developers/user-guide/c
# data_sources.append(
#     pw.io.gdrive.read(object_id="17H4YpBOAKQzEJ93xmC2z170l0bP2npMy", service_

# Choose document transformers
text_splitter = TokenCountSplitter()
embedder = OpenAIEmbedder(api_key=os.environ["OPENAI_API_KEY"])

# The `PathwayVectorServer` is a wrapper over `pathway.xpacks.llm.vector_store`
# Fell free to fork it to develop bespoke document processing pipelines.
vector_server = VectorStoreServer(
    *data_sources,
    embedder=embedder,
    splitter=text_splitter,
)
vector_server.run_server(host="127.0.0.1", port="8765", threaded=True, with_cac

```

We now instantiate and configure the client

```
client = VectorStoreClient(  
    host="127.0.0.1",  
    port="8765",  
)
```

And we can start asking queries

```
query = "What is Pathway?"  
docs = client(query)
```

Your turn! Now make a change to the source documents or make a fresh one and retry the query!

Integrations

Langchain

This currently is submitted to Langchain in [a PR](#)

```
# !pip install langchain
```

```
from langchain.vectorstores import PathwayVectorClient
```

```
# PathwayVectorClient implements regular VectorStore API of Langchain
```

```
client = PathwayVectorClient(host="127.0.0.1", port="8765")
```

```
docs = client.similarity_search("What is Pathway?")
```

LLama-index

See Pathway Retriever cookbook [here](#) Also see Pathway Reader cookbook [here](#)

```
# !pip install llama-index
```

```
from llama_index.retrievers import PathwayRetriever

# PathwayRetriever implements the Retriever interface
pr = PathwayRetriever(host="127.0.0.1", port="8765")
pr.retrieve(str_or_query_bundle="something")
```

Advanced topics

Getting information on indexed files

`PathwayVectorClient.get_vectorstore_statistics()` gives essential statistics on the state of the vector store, like the number of indexed files and the timestamp of the last updated one. You can use it in your chains to tell the user how fresh your knowledge base is.

```
client.get_vectorstore_statistics()
```

Filtering based on file metadata

We support document filtering using [jmespath](#) expressions, for instance:

```
# take into account only sources modified later than unix timestamp
docs = client(query, metadata_filter="modified_at >= `1702672093`")


# take into account only sources modified later than unix timestamp
docs = client(query, metadata_filter="owner == `james`")

# take into account only sources with path containing 'repo_readme'
docs = client(query, metadata_filter="contains(path, 'repo_readme')")

# and of two conditions
docs = client(query, metadata_filter="owner == `james` && modified_at >= `1702672093`")

# or of two conditions
docs = client(query, metadata_filter="owner == `james` || modified_at >= `1702672093`")
```

Configuring the parser

The vectorization pipeline supports pluggable parsers. If not provided, defaults to UTF-8 parser. You can find available parsers [here](#) . An example parser that can read PDFs, Word documents and other formats is provided with `parsers.ParseUnstructured` :

```
# !pip install unstructured[all-docs] # if you will need to parse complex documents
```

```
from pathway.xpacks.llm import parsers

vector_server = VectorStoreServer(
    *data_sources,
    parser=parsers.ParseUnstructured(),
    embedder=embeddings_model,
    splitter=text_splitter,
)
```

Configuring the cache

The Pathway vectorizing pipeline comes with an embeddings cache:

```
vector_server.run_server(..., with_cache=True)
```

The default cache configuration is the locally hosted disk cache, stored in the `./Cache` directory. However, it can be customized by explicitly specifying the caching backend chosen among several persistent backend [options](#).

Running in production

A production deployment will typically run the server in a separate process. We provide a template application under [templates](#) . We recommend running the Pathway data indexing pipeline in a container-based deployment environment like Docker or Kubernetes. For more info, see [Pathway's deployment guide](#).

Related articles



Realtime Classification with Nearest...

Olivier Ruas

2022-10-25

.pathway

Realtime Classification with Nearest...

Olivier Ruas

2022-10-26

B
M



Pathway Team

#LLM #RAG #GPT #OpenAI #Google Docs #KNN #Vector store #langchain
#llama-index #vectordb #vectore store langchain #retriever

F

S

Share this article

C



C

E

Showcases

← Use LLMs to Ingest Raw Text into ...

Showcases

Launching Pathway + Llamaln... →

Documentation

Tutorials

Showcases

About

Legal & GDPR

Equal opportunity employer

Privacy policy

Licensing

Media kit

Glossary

Contact

Let's talk

Chat with us on Discord

Pathway

96bis Boulevard Raspail

Agoranov

75006 Paris, France

contact@pathway.com

© 2021-2023 Pathway