July 20, 2023 SHOWCASE • LLM

Get from Github

Build an LLM App with Pathway

In this blog series, learn how to construct a dynamic, real-time LLM App using Pathway. Explore key features like real-time document indexing from S3 storage and other solutions, adaptive learning from updated documentation, and managing user sessions. Dive into this exciting combination of technologies that brings a responsive, knowledge-growing application to life.

Pathway makes handling realtime data easy. In this showcase we are going to demonstrate how Pathway can be used to build a chatbot answering questions about the Pathway documentation. This interactive application will exhibit dynamic adaptability to changing data sources:

1. User queries, to which responses must be generated in realtime,
2. Documentation entries, which should be incrementally re-indexed after each change.

The app in action can be seen in the video below:

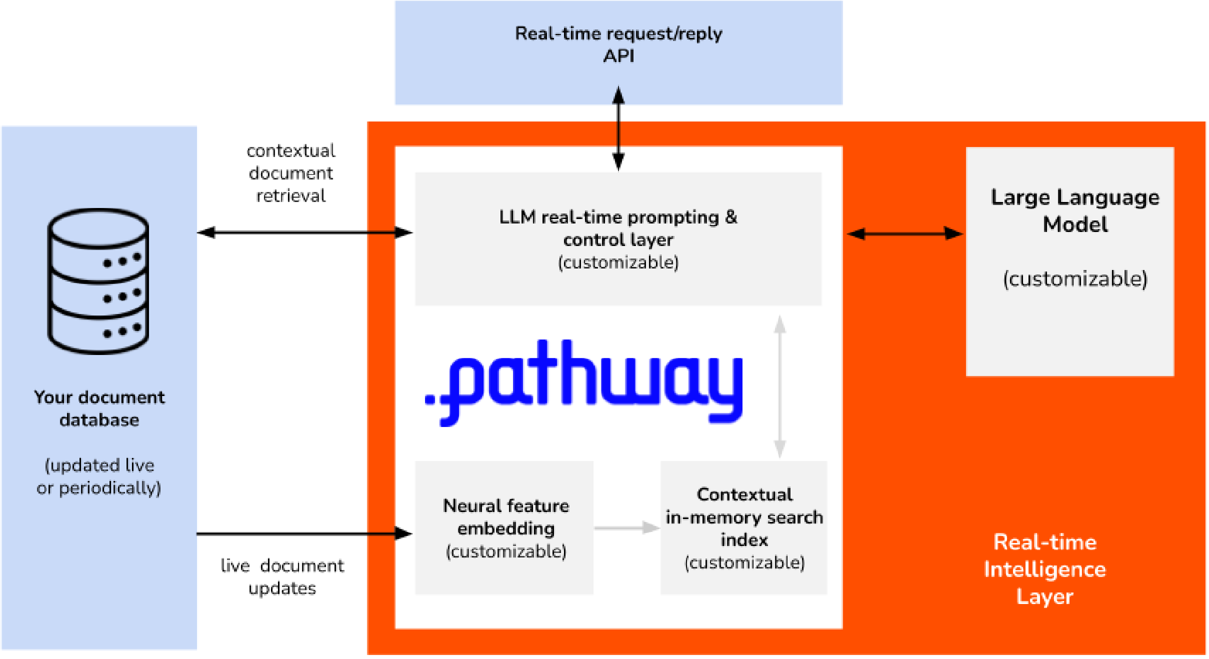
Build an LLM App in 30 lines of code without a vector database.

Jan Chorowski
CTO, Pathway

Speaker bio: Jan has developed attention models at Google Brain and MILA. 10k+ citations, co-author of Geoff Hinton and Yoshua Bengio.

pathwaycom/**llm-app**

The LLM (Large Language Model) App first reads a corpus of documents stored in S3. It preprocesses them and builds a vector index. It then listens to user queries coming as HTTP REST requests. Each query uses the index to retrieve relevant documentation snippets and uses the OpenAI API to provide a response in natural language. The bot is reactive to changes to the corpus of documents: once new snippets are provided, it reindexes them and starts to use the new knowledge to answer subsequent queries.


Real-time request/reply API
contextual document retrieval
Your document database
(updated live or periodically)
live document updates
LLM real-time prompting & control layer (customizable)
.pathway
Neural feature embedding (customizable)
Contextual in-memory search index (customizable)
Large Language Model
(customizable)
Real-time Intelligence Layer

In this series of posts we will explain below how Pathway can be used to:

- Implement a microservice that responds in realtime to HTTP REST queries
- Implement a realtime document indexing pipeline directly reading data from S3-compatible storage, without having to query a vector document database
- Extend the query building process to handle user sessions and beta tests for new models
- Reuse exactly the same code for offline evaluation of the system.

For an in-depth exploration of our app's underlying code, visit our GitHub repository at llm-app. Ensure you install the necessary dependencies with `poetry` by following the steps on the README file before diving in. We value transparency and collaborative learning, and invite you to explore and contribute to this open-source platform.

Warmup: answering user queries without context

The simplest way to get started with a conversational AI model using Pathway is to create an application that answers user queries without any context. This application will leverage a RESTful API and apply a Large Language Model.

Key Insights from This Section

- How to use a REST connector.
- Apply an LLM or any custom model on a user query.

The corresponding code can be located in the `examples/pipelines` directory. Now, we'll proceed with importing `Pathway` .

contextless/app.py

```
import os
import pathway as pw
from llm_app.model_wrappers import OpenAIChatGPTModel, OpenAIEmbeddingModel
```

contextless/app.py

```
#  REST Connector config.
HTTP_HOST = os.environ.get("PATHWAY_REST_CONNECTOR_HOST", "127.0.0.1")
HTTP_PORT = os.environ.get("PATHWAY_REST_CONNECTOR_PORT", "8080")

#  LLM model parameters
#  For OPENAI API
API_KEY = os.environ["OPENAI_API_KEY"]
#  Specific model from OpenAI. You can also use gpt-3.5-turbo for faster respor
MODEL_LOCATOR = "gpt-4"
# Controls the stochasticity of the openai model output.
TEMPERATURE = 0.0
# Max completion tokens
MAX_TOKENS = 50
```

Firstly, we define the input schema for our application. This is done using pw.Schema, which helps to enforce the structure of the data being processed by Pathway. Our schema, QueryInputSchema, expects a query (the question or prompt from the user) and a user (the identifier for the user). Then, we establish a RESTful connection using `pw.io.http.rest_connector` .

contextless/app.py

```
class QueryInputSchema(pw.Schema):
    query: str
    user: str



query, response_writer = pw.io.http.rest_connector(
    host=HTTP_HOST,
    port=int(HTTP_PORT),
    schema=QueryInputSchema,
    autocommit_duration_ms=50,
)
```

Here, `query` will be a Pathway stream that receives input from HTTP requests. `response_writer` is a function that we can use to write responses back to the HTTP client. We can now construct the main query pipeline for our application. The model to use here is GPT4 from OpenAI API.

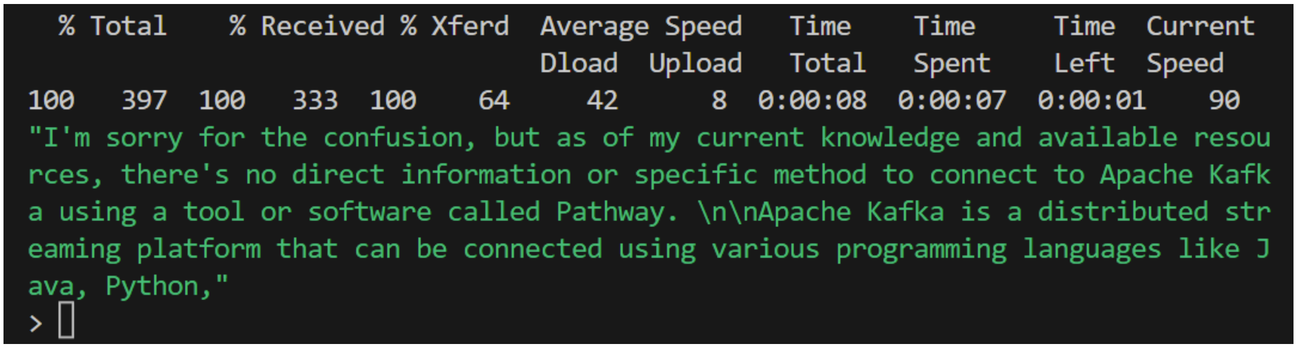
contextless/app.py

```
model = OpenAIChatGPTModel(api_key=API_KEY)

response = query.select(
    query_id=pw.this.id,
    result=model.apply(
        pw.this.query,
        locator=MODEL_LOCATOR,
        temperature=TEMPERATURE
        max_tokens=MAX_TOKENS
    ),
)

response_writer(response)
pw.run()
```

```
poetry run ./run_examples.py contextless
```

On a different terminal:

```
curl --data '{"user": "user", "query": "How to connect to Kafka in Pathway?"}'
```



```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   397  100   333  100    64     42      8  0:00:08  0:00:07  0:00:01    90
"I'm sorry for the confusion, but as of my current knowledge and available resou
rces, there's no direct information or specific method to connect to Apache Kafk
a using a tool or software called Pathway. \n\nApache Kafka is a distributed str
eaming platform that can be connected using various programming languages like J
ava, Python,"
> 
```

Context Enhancement for Better Responses

Despite `GPT-4` 's extensive training, it may not recognize certain context-specific elements, such as Pathway documentation. The solution lies in adding pertinent documents to the context. This is where the role of a vector database becomes crucial.

Essential Learnings from This Section

- Use an S3 input connector in Pathway.
- Generating vector embeddings using an LLM.
- Creating a k-Nearest Neighbors (k-NN) powered Index.
- Expanding the bot's capabilities to respond to user queries.

In our illustration, we'll consider a scenario where documents are stored in JSON Lines files within an AWS S3 bucket, though it could equally apply to a local directory using `jsonlines` reader. Each document is represented as a separate line within these files. The JSON Lines format is particularly advantageous for managing large data sets that cannot fit into memory all at once. Each line in a JSON Lines file contains a separate, independent JSON object. This makes the format especially suitable for handling and streaming large data, as it doesn't require loading the entire files into memory.

For each document and each query, we calculate embeddings using a pre-trained language model. These embeddings are numerical representations of the documents and they are used to find the documents that are most relevant to each query. Pathway offers API integration with premier LLM service providers, including but not limited to OpenAI and HuggingFace. You can import the model interface for the provider of your choice, specify the api key and the model id to call. By default the embedder is `text-embedding-ada-002` from OpenAI which returns vectors of dimension `1536` . Please check out **openai-model-endpoint-compatibility** for more information on the available models.

contextful/app.py

```
from pathway.stdlib.ml.index import KNNIndex


EMBEDDER_LOCATOR = "text-embedding-ada-002"
EMBEDDING_DIMENSION = 1536

embedder = OpenAIEmbeddingModel(api_key=API_KEY)
```

```python
class DocumentInputSchema(pw.Schema):
    doc: str

documents = pw.io.s3.read(
    "llm_demo/data/",
    aws_s3_settings=pw.io.s3.AwsS3Settings(
        bucket_name="pathway-examples",
        region="eu-central-1",
    ),
    format="json",
    schema=DocumentInputSchema,
    mode="streaming"
)



enriched_documents = documents + documents.select(
    data=embedder.apply(text=pw.this.doc, locator=EMBEDDER_LOCATOR)
)

query += query.select(
        data=embedder.apply(text=pw.this.query, locator=EMBEDDER_LOCATOR),
    )
```

```
           | query                           | data
^X1MXHYY... | How to connect to Kafka in Pathway? | [-0.00027798660448752344, 0
```

To achieve efficient retrieval of relevant documents, we leverage the power of **KNN (K-Nearest Neighbors)** indexing. By constructing an index using the generated embeddings, the KNN model allows us to quickly identify the documents that bear the most similarity to a given query. This technique is significantly faster and more efficient than conducting individual comparisons between the query and every document.

contextful/app.py

```python
index = KNNIndex(enriched_documents, d=EMBEDDING_DIMENSION)
```

```
query_context = query + index.get_nearest_items(
        query.data, k=3, collapse_rows=True
    ).select(documents_list=pw.this.doc)
```

```
            | query                              | documents_list
^X1MXHYY... | How to connect to Kafka in Pathway? | ('The documentation describ
```

By implementing the `build_prompt` function, we consolidate the query and associated documents into one coherent string, allowing the model to use the given documents for contextual understanding when generating its response. This procedure also provides an opportunity to include specific directives and guidelines for the Large Language Model (LLM) to adhere to.

contextful/app.py

```
@pw.udf
def build_prompt(documents, query) -> str:
    docs_str = "\n".join(documents)
    prompt = (
        f"Given the following documents : \n {docs_str} \nanswer this query: {q
    )
    return prompt


prompt = query_context.select(
    prompt=build_prompt(pw.this.documents_list, pw.this.query)
)
```

```
            | prompt
^X1MXHYY... | Given the following documents...
```

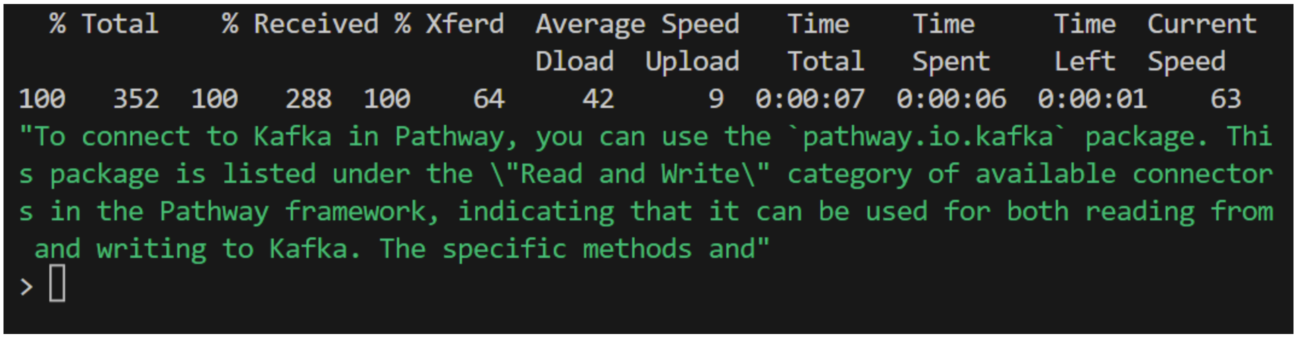
Ultimately, we invoke the `GPT-4` model with these thoughtfully crafted prompts and observe the sophistication of its generated responses.

contextful/app.py

```
response = prompt.select(
    query_id=pw.this.id,
    result=model.apply(
        pw.this.prompt,
        locator=MODEL_LOCATOR,
        temperature=TEMPERATURE,
        max_tokens=MAX_TOKENS,
    ),
)

response_writer(response)
pw.run()
```

```
poetry run ./run_examples.py contextful_s3
```

```
curl --data '{"user": "user", "query": "How to connect to Kafka in Pathway?"}'
```



```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   352  100   288  100    64     42      9  0:00:07  0:00:06  0:00:01    63
"To connect to Kafka in Pathway, you can use the `pathway.io.kafka` package. Thi
s package is listed under the \"Read and Write\" category of available connector
s in the Pathway framework, indicating that it can be used for both reading from
 and writing to Kafka. The specific methods and"
> 
```

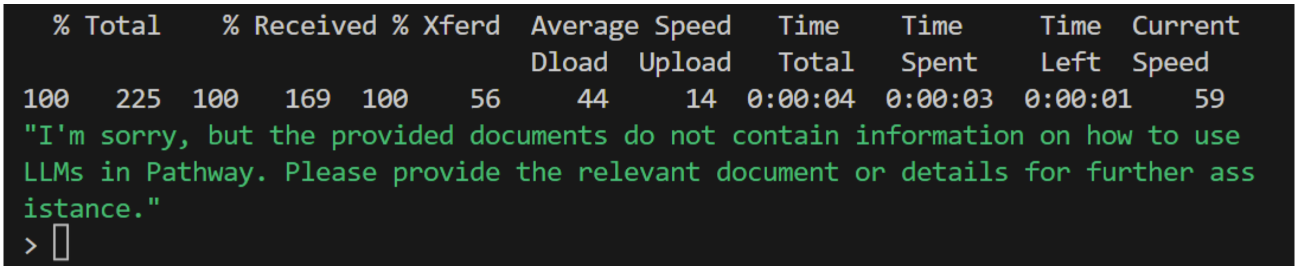
Real-time Adaptability: Automatic Updates with Pathway

A remarkable feature of Pathway is its automatic adaptability to changes. This feature makes Pathway an effective and efficient tool for real-time document indexing and query answering.

Once you have preprocessed your corpus and created the index, Pathway automatically detects any changes in the document directory and updates the vector index accordingly. This real-time reactivity ensures that app's responses are always based on the most recent and relevant information available.

Let's put this feature to the test. Consider a scenario where you initially query the system with "How to run large language models with Pathway?". Since the bot doesn't have any context about LLMs in Pathway, it wouldn't provide a satisfactory response at this point.

```
curl --data '{"user": "user", "query": "How to use LLMs in Pathway?"}' http://l
```



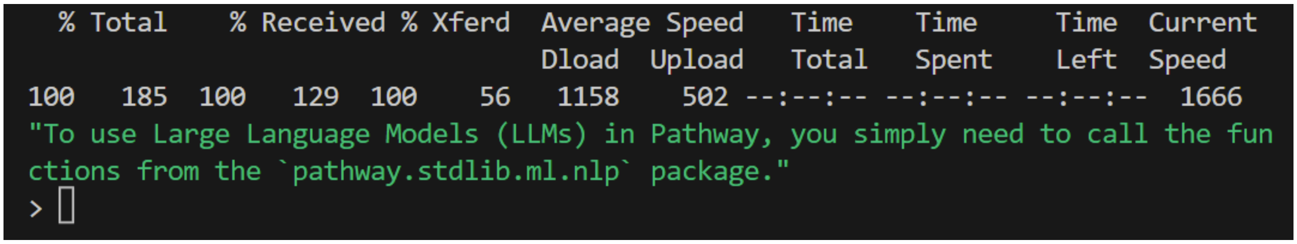
```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   225  100   169  100    56     44     14  0:00:04  0:00:03  0:00:01    59
"I'm sorry, but the provided documents do not contain information on how to use
LLMs in Pathway. Please provide the relevant document or details for further ass
istance."
> 
```

Next, we add some additional documents which provide context about Pathway to our S3 bucket.

```
aws s3 cp documents_extra.jsonl s3://pathway-examples/llm_demo/data/
```

Now, when you query the system with the same question again, Pathway automatically detects the newly added documents, updates the vector index, and the bot can provide a more appropriate response.

```
curl --data '{"user": "user", "query": "How to use LLMs in Pathway?"}' http://l
```



```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   185  100   129  100    56   1158    502 --:--:-- --:--:-- --:--:--  1666
"To use Large Language Models (LLMs) in Pathway, you simply need to call the fun
ctions from the `pathway.stdlib.ml.nlp` package."
> 
```

This real-time adaptability of Pathway is truly a game-changer when it comes to keeping your AI models updated with the latest data.

At this point, you should have a complete pipeline that not only sifts continuously through your document database to find the most relevant documents for a given query but also calls upon a Generative AI model to generate a detailed and coherent response based on these relevant documents.

The power of Pathway lies in its flexibility and robustness - you can tweak this pipeline to suit a variety of other applications, from customer support to medical literature review. The possibilities are truly endless.

Roadmap: What's Next in Our Series

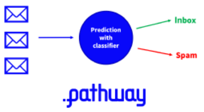
To stay updated with the latest content, we strongly encourage you to subscribe to our newsletter and give a star to the llm-app repository. Your engagement fuels our drive to continue delivering valuable resources and innovative use-cases.

We are excited to reveal what's coming up in the subsequent sections of this series:

- Efficient Index Building: We'll be moving the indexing process to a separate operation, essentially boosting your delta lake with vector indexing capabilities without necessitating the use of additional tools. This segment will also include an incremental rerun demonstration.
- Managing Complex Query-Response Generation Algorithms: This section delves into handling sophisticated algorithms for generating query responses, furthering the capabilities of your system.
- LLM App Evaluation: Lastly, we will elucidate how to critically assess the performance of your intelligent query response system. This part is essential to ensure your system's continuous improvement and optimal functioning.

Stay tuned for these exciting updates and more. Your journey towards mastering the application of Pathway in real-world use-cases is just beginning!

Related articles



.pathway

Mohamed Malhou

R&D Engineer in LLMs

#LLM #RAG #AWS S3 #Slack #GPT #OpenAI #KNN #HTTP connector

Share this article

Showcases

← Launching Pathway + LlamaIndex

Pathway

Features →

C

F

S

C

C

E

D

D

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