Databricks

The Databricks Lakehouse Platform unifies data, analytics, and AI on one platform.

This example notebook shows how to wrap Databricks endpoints as LLMs in LangChain. It supports two endpoint types:

- Serving endpoint, recommended for production and development,
- Cluster driver proxy app, recommended for iteractive development.

```
from langchain.llms import Databricks
```

Wrapping a serving endpoint

Prerequisites:

- An LLM was registered and deployed to a Databricks serving endpoint.
- You have "Can Query" permission to the endpoint.

The expected MLflow model signature is:

- inputs: `[{"name": "prompt", "type": "string"}, {"name": "stop", "type": "list[string]"}]`
- outputs: `[{"type": "string"}]`

If the model signature is incompatible or you want to insert extra configs, you can set `transform_input_fn` and `transform_output_fn` accordingly.

```
# If running a Databricks notebook attached to an interactive cluster in "single user"
# or "no isolation shared" mode, you only need to specify the endpoint name to create
# a `Databricks` instance to query a serving endpoint in the same workspace.
llm = Databricks(endpoint_name="dolly")

llm("How are you?")
```

```
    'I am happy to hear that you are in good health and as always, you are appreciated.'
```

```
llm("How are you?", stop=["."])
```

```
    'Good'
```

```
# Otherwise, you can manually specify the Databricks workspace hostname and personal access token
# or set `DATABRICKS_HOST` and `DATABRICKS_TOKEN` environment variables, respectively.
# See https://docs.databricks.com/dev-tools/auth.html#databricks-personal-access-tokens
# We strongly recommend not exposing the API token explicitly inside a notebook.
# You can use Databricks secret manager to store your API token securely.
# See https://docs.databricks.com/dev-tools/databricks-utils.html#secrets-utility-dbutilssecrets

import os

os.environ["DATABRICKS_TOKEN"] = dbutils.secrets.get("myworkspace", "api_token")
```

```
llm = Databricks(host="myworkspace.cloud.databricks.com", endpoint_name="dolly")

llm("How are you?")
```

```
    'I am fine. Thank you!'
```

```
# If the serving endpoint accepts extra parameters like `temperature`,
# you can set them in `model_kwargs`.
llm = Databricks(endpoint_name="dolly", model_kwargs={"temperature": 0.1})

llm("How are you?")
```

```
    'I am fine.'
```

```
# Use `transform_input_fn` and `transform_output_fn` if the serving endpoint
# expects a different input schema and does not return a JSON string,
# respectively, or you want to apply a prompt template on top.


def transform_input(**request):
    full_prompt = f"""{request["prompt"]}
    Be Concise.
    """
    request["prompt"] = full_prompt
    return request


llm = Databricks(endpoint_name="dolly", transform_input_fn=transform_input)
```

```
llm("How are you?")
```

```
    'I’m Excellent. You?'
```

Wrapping a cluster driver proxy app

Prerequisites:

- An LLM loaded on a Databricks interactive cluster in "single user" or "no isolation shared" mode.
- A local HTTP server running on the driver node to serve the model at `"/"` using HTTP POST with JSON input/output.
- It uses a port number between `[3000, 8000]` and listens to the driver IP address or simply `0.0.0.0` instead of localhost only.
- You have "Can Attach To" permission to the cluster.

The expected server schema (using JSON schema) is:

- inputs:

```
{"type": "object",
 "properties": {
    "prompt": {"type": "string"},
     "stop": {"type": "array", "items": {"type": "string"}}},
  "required": ["prompt"]}
```

- outputs: `{"type": "string"}`

If the server schema is incompatible or you want to insert extra configs, you can use `transform_input_fn` and `transform_output_fn` accordingly.

The following is a minimal example for running a driver proxy app to serve an LLM:

```
from flask import Flask, request, jsonify
import torch
from transformers import pipeline, AutoTokenizer, StoppingCriteria

model = "databricks/dolly-v2-3b"
tokenizer = AutoTokenizer.from_pretrained(model, padding_side="left")
dolly = pipeline(model=model, tokenizer=tokenizer, trust_remote_code=True, device_map="auto")
device = dolly.device

class CheckStop(StoppingCriteria):
    def __init__(self, stop=None):
        super().__init__()
        self.stop = stop or []
        self.matched = ""
        self.stop_ids = [tokenizer.encode(s, return_tensors='pt').to(device) for s in self.stop]
    def __call__(self, input_ids: torch.LongTensor, scores: torch.FloatTensor, **kwargs):
        for i, s in enumerate(self.stop_ids):
            if torch.all((s == input_ids[0][-s.shape[1]:])).item():
                self.matched = self.stop[i]
                return True
        return False

def llm(prompt, stop=None, **kwargs):
  check_stop = CheckStop(stop)
  result = dolly(prompt, stopping_criteria=[check_stop], **kwargs)
  return result[0]["generated_text"].rstrip(check_stop.matched)
```

```
app = Flask("dolly")

@app.route('/', methods=['POST'])
def serve_llm():
  resp = llm(**request.json)
  return jsonify(resp)

app.run(host="0.0.0.0", port="7777")
```

Once the server is running, you can create a `Databricks` instance to wrap it as an LLM.

```
# If running a Databricks notebook attached to the same cluster that runs the app,
# you only need to specify the driver port to create a `Databricks` instance.
llm = Databricks(cluster_driver_port="7777")

llm("How are you?")
```

```
    'Hello, thank you for asking. It is wonderful to hear that you are well.'
```

```
# Otherwise, you can manually specify the cluster ID to use,
# as well as Databricks workspace hostname and personal access token.

llm = Databricks(cluster_id="0000-000000-xxxxxxxx", cluster_driver_port="7777")

llm("How are you?")
```

```
    'I am well. You?'
```

```
# If the app accepts extra parameters like `temperature`,
# you can set them in `model_kwargs`.
llm = Databricks(cluster_driver_port="7777", model_kwargs={"temperature": 0.1})

llm("How are you?")
```

```
    'I am very well. It is a pleasure to meet you.'
```

```
# Use `transform_input_fn` and `transform_output_fn` if the app
# expects a different input schema and does not return a JSON string,
# respectively, or you want to apply a prompt template on top.


def transform_input(**request):
    full_prompt = f"""{request["prompt"]}
    Be Concise.
    """
    request["prompt"] = full_prompt
    return request


def transform_output(response):
    return response.upper()


llm = Databricks(
    cluster_driver_port="7777",
    transform_input_fn=transform_input,
    transform_output_fn=transform_output,
)
```

```
llm("How are you?")
```

```
'I AM DOING GREAT THANK YOU.'
```