Medium Search Write Sign in
Get unlimited access to the best of Medium for less than $5/week. Become a member

Giniborl
Follow
Our thoughts are a collage of angles in business and cognitive transformation. Visit us at www.giniborl.com
Follow conversation

Integrating Generative AI with Spark

Combining Generative AI and Spark for Powerful Insights

By Ravi Kumar · 9 min read · Mar 5, 2025

Photo by Google DeepMind on Unsplash

Nowadays, organizations leveraging advantages of generative AI (GenAI) to unlock insights. The Apache Spark ecosystem, combined with powerful libraries like LangChain helps convert plain English instructions into PySpark objects, such as dataframes. The best part is that it's designed to make Spark more user-friendly and accessible, allowing you to focus your efforts on extracting insights from your data.

In this article, we aim to explore how to leverage the LLM Zoo language model to enhance the data processing and analytics using PySpark. By providing clear instructions and practical examples, the article will guide you through the setup process needed for seamless integration of the Pyspark ecosystem.

Before diving into the practical aspects, let's ensure everything is set up correctly. Here's what you need to do:

Setting Up Spark for AI Workloads

In this section, we will discuss what is required to integrate LLM and Pyspark and how we are going to achieve that.

The following steps are required in order to integrate the LLM and Pyspark.

First, ensure you have a Python JDK. You can use TikZjava, Jupyter Notebook, or PyCharm. Second, install the required packages: langchain and langchain-community. Both packages will help you establish a connection between the language model and Spark.

In order to install the packages, open your terminal or command prompt and run the following command:

```
pip install "pyspark[pandas_on_spark]"
pip install langchain-community
```

After a successful installation of pyspark-ai and langchain-community, now it's time to set up configuration such as LLM API key and LLM Zoo connection with OpenAI LLM. In this article, we'll focus on connecting with OpenAI LLM, but I'll also provide you with code snippets for Google LLM.

The below code snippet sets up the environment for using the LangChain library with OpenAI's chat models. We need to import the necessary modules, specifically the langchain_openai from langchain_community package and os modules. These are used to define two environment variables, OPENAI_API_KEY which should be replaced with your actual OpenAI API key, and OPENAI_API_BASE which specifies the base URL for the API. Here we are going to use gpt-4o model. These setups are essential for authenticating and configuring the chat model for further use in a PySpark application.

```
from langchain_openai.chat_models import ChatOpenAI
import os
os.environ["OPENAI_API_KEY"] = "your-openai-key"
os.environ["OPENAI_API_BASE"] = "https://api.openai.com/v1"

llm = ChatOpenAI(
    model="gpt-4o",
    temperature=0.0,
    max_tokens=1000,
    timeout=30,
    max_retries=2,
    # other parameters...
)
```

The below code snippets import the pyspark_ai module, then from the LangChain library import the ChatOpenAI class and initialize an instance (llm) configured to connect to an Azure deployment of GPT-4 models. The initialize_llm() function accepts parameters stored in be specified to identify the specific Azure deployment and the model to be used. This setup enables the integration of Azure-hosted OpenAI models into applications using the LangChain framework.

```
from langchain_chat_models import AzureChatOpenAI

llm = AzureChatOpenAI(
    deployment_name="...",
    model_name="...",
)
```

But here we are using the OpenAI API to enhance our experience. If you want to use GenAI, that's also okay.

Now, we need to also import the required package, such as pyspark_ai and initialize the SparkAI object. Look at the following snippet of code.

```
from pyspark_ai import SparkAI
from langchain_community.chat_models import ChatOpenAI
llm = ChatOpenAI(verbose=True)
spark_ai = SparkAI(llm=llm)
spark_ai.activate()
```

We've activated our Spark AI session with just a few lines of code. Now we are ready to explore the functionalities of AI powered Pyspark.

Data Ingestion into DataFrame

So far, we have imported all required Python libraries. Now we need to get the data into DataFrame. If you're done set up the Google Python client, you're in luck! You can easily ingest data using the search engine like this:

```
df = spark_ai.create_df("Titanic data by year")
```

But don't worry if you haven't set that up yet! You can also ingest data directly from a URL. Here's how you can grab movie titles from IMDB:

```
# Search and ingest web content into a DataFrame
df_df = spark_ai.create_df("https://www.imdb.com/search/title/?title_type=feature
```

When you execute the above line, SparkAI will generate SQL instructions internally and load the data into a DataFrame. Here is the SQL output that is executed internally.

```
INFO: Starting URL: https://www.imdb.com/search/title/?title_type=feature&sort=r
INFO: URL parsed to: URL(scheme='https', netloc='www.imdb.com', path='/search/ti
INFO: Using selenium for rendering
INFO: Waiting for content to load for https://www.imdb.com/search/title/?title_t
INFO: Content loaded, extracting HTML
INFO: Successfully fetched 25 rows from https://www.imdb.com/search/title/?title
+---------------------------------------------------------------------------------
|                                                                                 |
|                                                                                 |
|                                                                                 |
```

Help Status About Careers Press Blog Privacy Rules Terms Text to speech