Home > Tutorials > Artificial Intelligence

DuckDB Tutorial: Building AI Projects

This tutorial guides you through DuckDB's key features and practical applications, including building tables, performing data analysis, building an RAG application, and using an SQL query engine with LLM.

Jul 7, 2024 · 13 min read

Abid Ali Awan

Certified data scientist, passionate about building ML apps, blogging on data science, and editing.

TOPICS

Artificial Intelligence

Recently, DuckDB came out of beta and released its stable version, gaining popularity rapidly as various data frameworks integrate it into their ecosystems. This makes it a prime time to learn DuckDB so you can keep up with the ever-changing world of data and AI.

In this tutorial, we will learn about DuckDB and its key features with code examples. Our primary focus will be on how we can integrate it with current AI frameworks. For that, we will work on two projects. First, we'll build a Retrieval-Augmented Generation (RAG) application using DuckDB as a vector database. Then, we'll use DuckDB as an AI query engine to analyze data using natural language instead of SQL.

What is DuckDB?

DuckDB is a modern, high-performance, in-memory analytical database management system (DBMS) designed to support complex analytical queries. It is a relational (table-oriented) DBMS that supports the Structured Query Language (SQL).

DuckDB combines the simplicity and ease of use of SQLite with the high-performance capabilities required for analytical workloads, making it an excellent choice for data scientists and analysts.

Key features

1. **Simple operation:** DuckDB is serverless, has no external dependencies, and is embedded within a host process. This makes it easy to install and deploy, requiring only a C++11 compiler for building.
2. **Feature-rich:** It supports extensive SQL data management features. DuckDB also offers deep integration with Python and R, making it suitable for data science and interactive data analysis.
3. **Fast analytical queries:** DuckDB uses a columnar-vectorized query execution engine optimized for analytics, enabling parallel query processing and efficient handling of large datasets.
4. **Free and open source:** It is released under the permissive MIT License, making it free to use and open-source.
5. **Portability:** With no external dependencies, DuckDB is highly portable and can run on various operating systems (Linux, macOS, Windows) and CPU architectures (x86, ARM). It can even run in web browsers using DuckDB-Wasm.
6. **Extensibility:** DuckDB supports a flexible extension mechanism, allowing the addition of new data types, functions, file formats, and SQL syntax.
7. **Thorough testing:** It undergoes intensive testing using Continuous Integration, with a test suite containing millions of queries. This ensures stability and reliability across different platforms and compilers.

Getting Started with DuckDB

In this section, we will learn to set up DuckDB, load CSV files, perform data analysis, and learn about relations and query functions.

We will start by installing the DuckDB Python package.

```
pip install duckdb --upgrade
```

Hide code explanation

POWERED BY datalab

- This line of code is a command used in the terminal or command prompt, not in a Python script.
- `pip` is a package installer for Python. It's used to install and manage software packages/libraries.
- `install` is a command that tells pip to install a package.
- `duckdb` is the name of the package that pip is being told to install.
- `--upgrade` is an optional argument that tells pip to upgrade the package if it's already installed.

Was this helpful? ✓ Yes ✕ No

Creating the DuckDB database

To create the persistent database, you just have to use the `connect` function and provide it with the database name.

```
import duckdb
con = duckdb.connect("datacamp.duckdb")
```

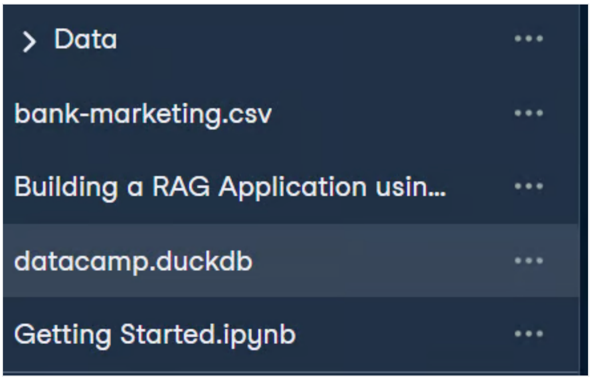
Hide code explanation

POWERED BY datalab

- The code starts by importing the `duckdb` module, which is a fast analytical database written in C++.
- Then. it establishes a connection to a DuckDB database named "datacamp.duckdb" using the `connect` method.

Was this helpful? ✓ Yes ✕ No

It will create a database base file in your local directory.



> Data
bank-marketing.csv
Building a RAG Application usin...
datacamp.duckdb
Getting Started.ipynb

We will load a CSV file and create a "bank" table. The dataset we are using is available on DataLab and is called [Bank Marketing](). It consists of direct marketing campaigns by a Portuguese banking institution using phone calls.

To load the CSV file, you have to create a Table first using SQL and then use the `read_csv()` function within the SQL script to load the file. It is that simple.

We will then validate our table by executing the SQL script that shows all of the tables within the database and using the `fetchdf` function to display the result as a `pandas` DataFrame.

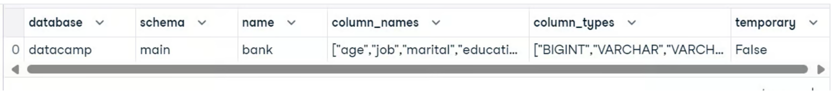
Note: We are using DataCamp's DataLab as a code editor. DataLab is a cloud Jupyter Notebook that you can access for free if you have a DataCamp account.

```
con.execute("""
    CREATE TABLE IF NOT EXISTS bank AS
    SELECT * FROM read_csv('bank-marketing.csv')
""")
con.execute("SHOW ALL TABLES").fetchdf()
```

Hide code explanation POWERED BY datalab

- The `con.execute()` function is used to execute SQL commands.
- The first `con.execute()` runs a `CREATE TABLE` SQL command.
- `IF NOT EXISTS` checks if the table 'bank' already exists, and if it does, the command is ignored.
- `AS SELECT * FROM read_csv('bank-marketing.csv')` creates the 'bank' table using data from the CSV file.
- The `read_csv()` function reads the 'bank-marketing.csv' file.
- The second `con.execute()` runs a `SHOW ALL TABLES` SQL command.
- `fetchdf()` fetches the result of the `SHOW ALL TABLES` command as a DataFrame.

Was this helpful? ✓ Yes ✕ No



	database	schema	name	column_names	column_types	temporary
0	datacamp	main	bank	["age","job","marital","educati...	["BIGINT","VARCHAR","VARCH...	False

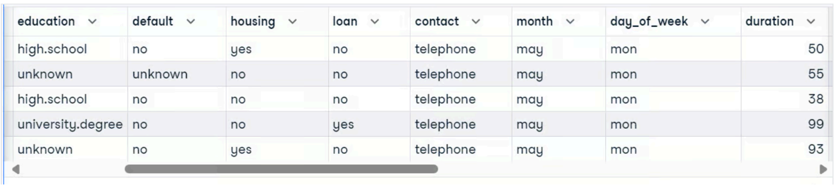
Now that we have successfully created our first table, we will run a beginner-level query to analyze the data and display the result as a DataFrame.

```
con.execute("SELECT * FROM bank WHERE duration < 100 LIMIT 5").fetchdf()
```

Hide code explanation POWERED BY datalab

- `con.execute()` is a method that runs the SQL query enclosed in the parentheses.
- `"SELECT * FROM bank WHERE duration &lt; 100 LIMIT 5"` is the SQL query being executed.
- `SELECT * FROM bank` selects all columns from the 'bank' table.
- `WHERE duration &lt; 100` filters the data to only include rows where the 'duration' is less than 100.
- `LIMIT 5` restricts the output to the first 5 rows that match the condition.
- `.fetchdf()` is a method that fetches the result of the query and returns it as a DataFrame.

Was this helpful? ✓ Yes ✕ No



education	default	housing	loan	contact	month	day_of_week	duration
high.school	no	yes	no	telephone	may	mon	50
unknown	unknown	no	no	telephone	may	mon	55
high.school	no	no	no	telephone	may	mon	38
university.degree	no	no	yes	telephone	may	mon	99
unknown	no	yes	no	telephone	may	mon	93

DuckDB is natively integrated into the new DataLab by DataCamp. Learn more about it by reading the blog "DuckDB Makes SQL a First-Class Citizen on DataLab" and using the interactive SQL cell to analyze data.

DuckDB Relations

DuckDB relations are essentially tables that can be queried using the Relational API. This API allows for the chaining of various query operations on data sources like Pandas DataFrames. Instead of using SQL queries, you will by chaining together various Python functions to analyze the data.

For example, we will load a CSV file to create the DuckDB relation. To analyze the table, you can chain the filter and limit functions.

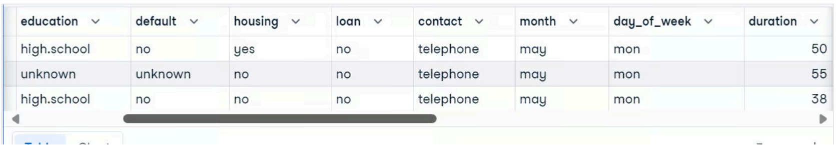
```
bank_duck = duckdb.read_csv("bank-marketing.csv",sep=";")
bank_duck.filter("duration < 100").limit(3).df()
```

Hide code explanation

POWERED BY datalab

- The code uses the `duckdb` library, which is a high-performance analytical database system.
- `duckdb.read_csv("bank-marketing.csv",sep=";")` reads a CSV file named "bank-marketing.csv" using a semicolon as a separator.
- The read CSV file is stored in the `bank_duck` variable.
- `bank_duck.filter("duration &lt; 100")` filters the data to only include rows where the "duration" column is less than 100.
- `.limit(3)` further limits the output to the first 3 rows of the filtered data.
- `.df()` converts the result into a pandas DataFrame, which is a two-dimensional, size-mutable, and heterogeneous tabular data structure.

Was this helpful? ✓ Yes ✕ No



education	default	housing	loan	contact	month	day_of_week	duration
high.school	no	yes	no	telephone	may	mon	50
unknown	unknown	no	no	telephone	may	mon	55
high.school	no	no	no	telephone	may	mon	38

We can also create relations by loading the table from the DuckDB database.

```
rel = con.table("bank")
```

```
rel.columns
```

✨ Hide code explanation POWERED BY datalab

- The code is using a database connection object `con` to access a table named "bank".
- `rel = con.table("bank")` is accessing the "bank" table from the database and storing it in `rel`.
- `rel.columns` is then used to display the column names of the "bank" table.

Was this helpful? ✓ Yes ✕ No

```
['age',
 'job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
 'nr.employed',
 'y']
```

✨ Hide code explanation POWERED BY datalab

The text above is a code output or a data entry that complements the tutorial.

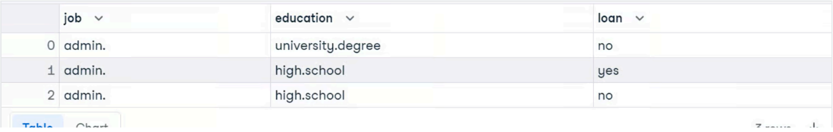
Was this helpful? ✓ Yes ✕ No

Let’s write a relation that uses multiple functions to analyze the data.

```
rel.filter("duration < 100").project("job,education,loan").order("job").lim  ).
```

✨ Explain code POWERED BY datalab

We have three rows and columns sorted by job and filtered by duration column.



	job	education	loan
0	admin.	university.degree	no
1	admin.	high.school	yes
2	admin.	high.school	no

DuckDB Query Function

The DuckDB query function allows SQL queries to be executed within the database, returning results that can be converted into various formats for further analysis.

In the code example, we are running the SQL query to find out the job titles of clients over the age of 30, count the number of clients contacted for each job, and calculate the average duration of the campaign.

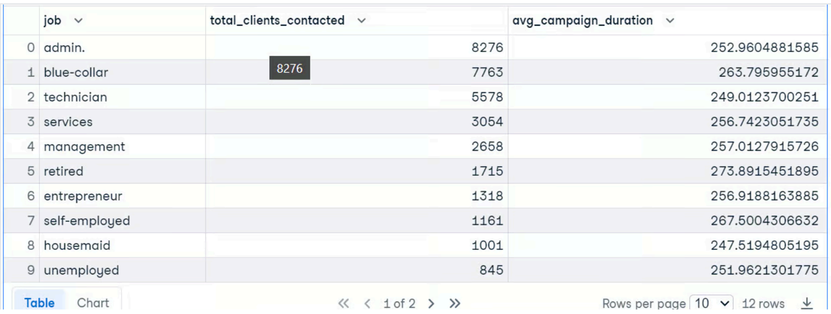
Take the SQL Fundamentals skill track to learn how to manage a relational database and execute queries for simple data analysis.

```
res = duckdb.query("""SELECT
                            job,
                            COUNT(*) AS total_clients_contacted,
                            AVG(duration) AS avg_campaign_duration,
                        FROM
                            'bank-marketing.csv'
                        WHERE
                            age > 30
                        GROUP BY
                            job
                        ORDER BY
                            total_clients_contacted DESC;""")
res.df()
```

Hide code explanation POWERED BY datalab

- The code is written in SQL and is executed using the DuckDB Python library.
- `res = duckdb.query("""...""")` is running an SQL query and storing the result in the variable `res`.
- `SELECT job, COUNT(*) AS total_clients_contacted, AVG(duration) AS avg_campaign_duration` is selecting the 'job' column, the count of rows (renamed as 'total_clients_contacted'), and the average of the 'duration' column (renamed as 'avg_campaign_duration').
- `FROM 'bank-marketing.csv'` is specifying the data source, a CSV file named 'bank-marketing.csv'.
- `WHERE age &gt; 30` is filtering the data to only include rows where the 'age' column is greater than 30.
- `GROUP BY job` is grouping the selected data by the 'job' column.
- `ORDER BY total_clients_contacted DESC` is ordering the result in descending order based on the 'total_clients_contacted' column.
- `res.df()` is converting the result into a pandas DataFrame for easier manipulation and analysis.

Was this helpful? ✓ Yes ✕ No



	job	total_clients_contacted	avg_campaign_duration
0	admin.	8276	252.9604881585
1	blue-collar	7763	263.795955172
2	technician	5578	249.0123700251
3	services	3054	256.7423051735
4	management	2658	257.0127915726
5	retired	1715	273.8915451895
6	entrepreneur	1318	256.9188163885
7	self-employed	1161	267.5004306632
8	housemaid	1001	247.5194805195
9	unemployed	845	251.9621301775

We will now close the connection to the database and release any resources associated with that connection, preventing potential memory and file handle leaks.

```
con.close()
```

Hide code explanation POWERED BY datalab

- This is a single line of code that closes the connection to a database.
- `con` is the variable representing the database connection.
- `.close()` is a method used to terminate the connection.

Was this helpful? ✓ Yes ✕ No

If you are facing issues running the above code, please have a look at the Getting Started with DuckDB workspace.

Building a RAG Application with DuckDB

In the first project, we will learn to build an RAG application with LlamaIndex and use DuckDB as a Vector database and retriever.

Setting up

Install all the necessary Python packages that will be used to create and retrieve the index.

```
%%capture
%pip install duckdb
%pip install llama-index
%pip install llama-index-vector-stores-duckdb
```

Hide code explanation

POWERED BY datalab

- The `%%capture` command is a Jupyter notebook magic command that suppresses the output of the cell.
- The `%pip install duckdb` command installs the DuckDB library, an in-memory analytical database.
- The `%pip install llama-index` command installs the Llama Index library, a Python library for indexing.
- The `%pip install llama-index-vector-stores-duckdb` command installs a specific component of the Llama Index library that integrates with DuckDB.

Was this helpful? ✓ Yes ✗ No

Import the necessary Python package with the functions.

```
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.vector_stores.duckdb import DuckDBVectorStore
from llama_index.core import StorageContext

from IPython.display import Markdown, display
```

Hide code explanation

POWERED BY datalab

- The code imports the `VectorStoreIndex` and `SimpleDirectoryReader` classes from the `llama_index.core` module.
- It then imports the `DuckDBVectorStore` class from the `llama_index.vector_stores.duckdb` module.
- The `StorageContext` class is also imported from the `llama_index.core` module.
- Finally, the `Markdown` and `display` functions are imported from the `IPython.display` module.

Was this helpful? ✓ Yes ✗ No

Setting up GPT-4o and Embedding Model

For a language model, we will use the latest GPT4o model and the OpenAI API. To create the large language model (LLM) client, you just have to provide a model name and API key.

```
import os
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-4o",api_key=os.environ["OPENAI_API_KEY"])
```

Hide code explanation

POWERED BY datalab

- The code begins by importing the 'os' module, which provides functions for interacting with the operating system.
- Then, it imports the 'OpenAI' class from the 'llama_index.llms.openai' module.
- It creates an instance 'llm' of the 'OpenAI' class.
- The 'OpenAI' class is initialized with two parameters: 'model' and 'api_key'.
- The 'model' parameter is set to "gpt-4o", which is the name of the model to be used.
- The 'api_key' parameter is retrieved from the environment variables using 'os.environ["OPENAI_API_KEY"]'.

Was this helpful? ✓ Yes ✕ No

Then, we will create the embed model client using the OpenAI text-embedding-3-small model.

Note: Providing an OpenAI API key is optional if the environment variable is set with the name “OPENAI_API_KEY” on your development environment.

```
from llama_index.embeddings.openai import OpenAIEmbedding
embed_model = OpenAIEmbedding(
    model="text-embedding-3-small",
)
```

Hide code explanation POWERED BY datalab

- The code starts with importing the `OpenAIEmbedding` class from the `llama_index.embeddings.openai` module.
- Then, it creates an instance of the `OpenAIEmbedding` class named `embed_model`.
- While creating the instance, it passes `"text-embedding-3-small"` as an argument to the `model` parameter of the `OpenAIEmbedding` class.
- The `embed_model` instance can now be used to generate embeddings for text data using the specified OpenAI model.

Was this helpful? ✓ Yes ✕ No

We will make OpenAI LLM and Embedding models global for all LlamaIndex functions to use. In short, these models will be set as default.

```
from llama_index.core import Settings

Settings.llm = llm
Settings.embed_model = embed_model
```

Hide code explanation POWERED BY datalab

- The code begins with importing the `Settings` class from the `llama_index.core` module.
- The `Settings` class is likely a configuration class for the `llama_index` package.
- The `llm` object is assigned to the `llm` attribute of the `Settings` class.
- The `embed_model` object is assigned to the `embed_model` attribute of the `Settings` class.
- These assignments are probably used to configure the behavior of the `llama_index` package.

Was this helpful? ✓ Yes ✕ No

Using DuckDB as a vector database

For our project, we will load the PDF files from the data folder. These PDF files are tutorials from DataCamp that are saved as PDF files using the browser’s print function.

Provide the folder directory to the SimpleDirectoryReader function and load the data.

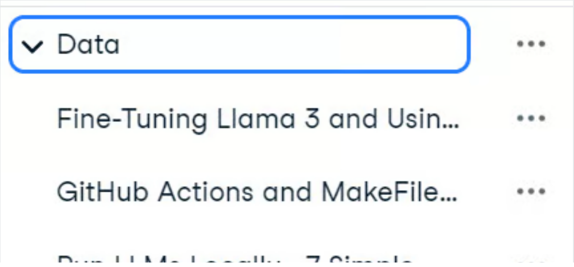
```
documents = SimpleDirectoryReader("Data").load_data()
```

Hide code explanation

POWERED BY datalab

- The code is written in Python and it's using a class named `SimpleDirectoryReader`.
- `SimpleDirectoryReader("Data")` creates an instance of the `SimpleDirectoryReader` class.
- The string `"Data"` is passed as an argument, which is likely the name of the directory to be read.
- The method `load_data()` is called on the instance, which presumably loads the data from the specified directory.
- The loaded data is then stored in the variable `documents`.

Was this helpful? ✓ Yes ✕ No


Data
Fine-Tuning Llama 3 and Usin...
GitHub Actions and MakeFile...

Then, create the vector store called “blog” using an existing database called “datacamp.duckdb.” After that, convert the PDF's data into embeddings and store them in the vector store.

```
vector_store = DuckDBVectorStore(database_name = "datacamp.duckdb",table_nar  "
storage_context = StorageContext.from_defaults(vector_store=vector_store)

index = VectorStoreIndex.from_documents(
    documents, storage_context=storage_context
)
```

Hide code explanation

POWERED BY datalab

- The first line creates an instance of `DuckDBVectorStore` with specified database and table names, a directory for persistence, and embedding dimension.
- The second line creates a `StorageContext` using the default settings and the previously created `vector_store`.
- The final line creates a `VectorStoreIndex` from a list of documents, using the previously created `storage_context`.

Was this helpful? ✓ Yes ✕ No

To check if our vector store was successfully created, we will connect the database using the DuckDB Python API and run the SQL query to display all the tables in the database.

```
import duckdb
con = duckdb.connect("datacamp.duckdb")

con.execute("SHOW ALL TABLES").fetchdf()
```

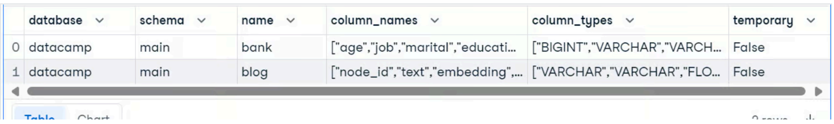
Hide code explanation

POWERED BY datalab

- The code starts by importing the `duckdb` module, which is a high-performance analytical database system.
- The `duckdb.connect("datacamp.duckdb")` line establishes a connection to a DuckDB database file named "datacamp.duckdb".
- `con.execute("SHOW ALL TABLES")` sends a SQL command to the database to retrieve all table names.
- The `.fetchdf()` method is then used to fetch the result of the SQL command as a pandas DataFrame.

Was this helpful? ✓ Yes ✕ No

We have two tables: a “bank” promotional table and a “blog” table, which is a vector store. The “blog” table has an “embedding” column where all the embeddings are stored.



	database	schema	name	column_names	column_types	temporary
0	datacamp	main	bank	["age","job","marital","educati...	["BIGINT","VARCHAR","VARCH...	False
1	datacamp	main	blog	["node_id","text","embedding",...	["VARCHAR","VARCHAR","FLO...	False

Creating a simple RAG application

Convert the index into the query engine, which will automatically first search the vector database for similar documents and use the additional context to generate the response.

To test the RAG query engine, we will ask the question about the tutorial.

```
query_engine = index.as_query_engine()
response = query_engine.query("Who wrote 'GitHub Actions and MakeFile: A Hands-on
display(Markdown(f"<b>{response}</b>"))
```

Hide code explanation POWERED BY datalab

- The first line creates a query engine from an index object using the `as_query_engine()` method.
- The second line uses the `query()` method of the query engine to search for the author of a specific article.
- The third line uses the `Markdown` function to format the response in bold, and `display` function to show it.

Was this helpful? ✓ Yes ✕ No

And the answer is correct.

```
The author of "GitHub Actions and MakeFile: A Hands-on Introduction" is Abir i
```

Hide code explanation POWERED BY datalab

The text above is a code output or a data entry that complements the tutorial.

Was this helpful? ✓ Yes ✕ No

Creating a RAG chatbot with memory

Now, let’s create an advanced RAG application that uses the conversation history to generate the response. For that, we have to create a chat memory buffer and then a chat engine with memory, LLM, and vector store retriever.

```
from llama_index.core.memory import ChatMemoryBuffer
from llama_index.core.chat_engine import CondensePlusContextChatEngine

memory = ChatMemoryBuffer.from_defaults(token_limit=3900)
```

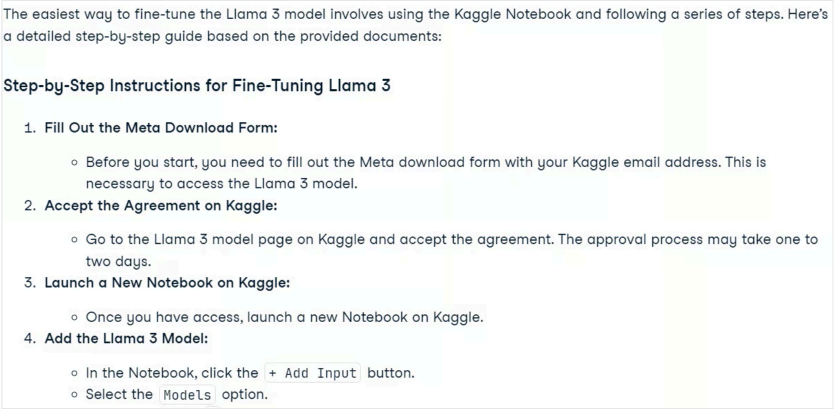
```
chat_engine = CondensePlusContextChatEngine.from_defaults(
    index.as_retriever(),
    memory=memory,
    llm=llm
)

response = chat_engine.chat(
    "What is the easiest way of finetuning the Llama 3 model? Please provide step
)

display(Markdown(response.response))
```

Hide code explanation POWERED BY datalab

- The code starts by importing `ChatMemoryBuffer` from `llama_index.core.memory`.
- It also imports `CondensePlusContextChatEngine` from `llama_index.core.chat_engine`.
- A `ChatMemoryBuffer` object is created with a token limit of 3900.
- A `CondensePlusContextChatEngine` object is created using default settings, the previously created memory buffer, and `llm`.
- The `chat` method of `chat_engine` is called with a question about fine-tuning the Llama 3 model.
- The response from the chat engine is displayed using the `Markdown` function.

Was this helpful? ✓ Yes ✕ No

We asked the chat engine how to fine-tune the Llama 3 model, and it used the vector store to give a highly accurate answer.



The easiest way to fine-tune the Llama 3 model involves using the Kaggle Notebook and following a series of steps. Here's a detailed step-by-step guide based on the provided documents:

Step-by-Step Instructions for Fine-Tuning Llama 3

1. **Fill Out the Meta Download Form:**
   - Before you start, you need to fill out the Meta download form with your Kaggle email address. This is necessary to access the Llama 3 model.
2. **Accept the Agreement on Kaggle:**
   - Go to the Llama 3 model page on Kaggle and accept the agreement. The approval process may take one to two days.
3. **Launch a New Notebook on Kaggle:**
   - Once you have access, launch a new Notebook on Kaggle.
4. **Add the Llama 3 Model:**
   - In the Notebook, click the `+ Add Input` button.
   - Select the `Models` option.

To check if the memory buffer is working correctly, we will ask a follow-up question.

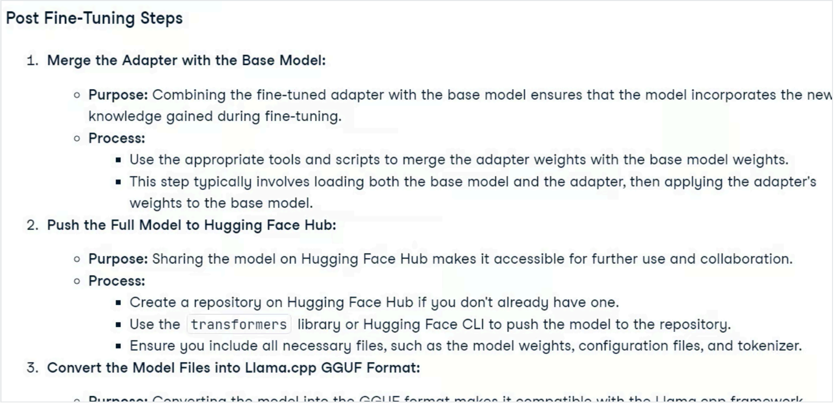
```
response = chat_engine.chat(
    "Could you please provide more details about the Post Fine-Tuning Steps?
)
display(Markdown(response.response))
```

Hide code explanation POWERED BY datalab

- The code is using a chat engine, which is likely an AI model, to generate a response to a given input.
- The method `chat_engine.chat()` is called with a string argument, which is the question to be asked.
- The response from the chat engine is stored in the variable `response`.
- `Markdown(response.response)` is used to format the response text in Markdown, a lightweight markup language.
- The `display()` function is then used to display the formatted text in the output.

Was this helpful? ✓ Yes ✕ No

The chat engine remembered the previous conversation and responded accordingly.



Post Fine-Tuning Steps

1. **Merge the Adapter with the Base Model:**
   - **Purpose:** Combining the fine-tuned adapter with the base model ensures that the model incorporates the new knowledge gained during fine-tuning.
   - **Process:**
     - Use the appropriate tools and scripts to merge the adapter weights with the base model weights.
     - This step typically involves loading both the base model and the adapter, then applying the adapter's weights to the base model.
2. **Push the Full Model to Hugging Face Hub:**
   - **Purpose:** Sharing the model on Hugging Face Hub makes it accessible for further use and collaboration.
   - **Process:**
     - Create a repository on Hugging Face Hub if you don't already have one.
     - Use the `transformers` library or Hugging Face CLI to push the model to the repository.
     - Ensure you include all necessary files, such as the model weights, configuration files, and tokenizer.
3. **Convert the Model Files into Llama.cpp GGUF Format:**
   - **Purpose:** Converting the model into the GGUF format makes it compatible with the Llama.cpp framework

If you are facing issues running the above code, please have a look at the [Building a RAG application with DuckDB]() workspace.

Building a DuckDB SQL Query Engine Using an LLM

In the second project, we will use DuckDB as an SQL query engine. This involves integrating the database engine with the GPT-4o model to generate natural language responses to questions about the database.

Install duckdb-engine to create a database engine using SQLAlchemy.

```
%pip install duckdb-engine -q
```

Hide code explanation

POWERED BY datalab

- The code is using the `%pip` command, which is a magic command in Jupyter notebooks.
- The `install` argument tells pip to install a package.
- `duckdb-engine` is the name of the package that is being installed.
- The `-q` flag is used to run the command in quiet mode, reducing the output displayed.

Was this helpful? ✓ Yes ✕ No

Loading the DuckDB database

We will load the DuckDB database using the create_engine function and then write a simple SQL query to check whether it is successfully loaded.

```
from sqlalchemy import create_engine

engine = create_engine("duckdb:///datacamp.duckdb")
with engine.connect() as connection:
    cursor = connection.exec_driver_sql("SELECT * FROM bank LIMIT 3")
    print(cursor.fetchall())
```

Hide code explanation

POWERED BY datalab

- The code begins by importing the `create_engine` function from the `sqlalchemy` module.
- `create_engine` is then used to create an engine that connects to a DuckDB database file named `datacamp.duckdb`.
- `with engine.connect()` is used to establish a connection to the database.
- Inside the `with` block, `connection.exec_driver_sql("SELECT * FROM bank LIMIT 3")` is executed.
- This SQL query selects all columns from the first three rows of the 'bank' table in the database.
- The results of the query are fetched with `cursor.fetchall()` and then printed to the console.

Was this helpful? ✓ Yes ✕ No

Prefect. Our DuckDB database engine is ready to be used.

```
[(56, 'housemaid', 'married', 'basic.4y', 'no', 'no', 'no', 'telephone', 'ma  '
```

Hide code explanation POWERED BY datalab

- The provided Python code is a list of tuples.
- Each tuple in the list contains 21 elements.
- These elements could represent different attributes of a dataset, such as age, job, marital status, education, etc.
- The data types of the elements in the tuples vary, including integers, strings, and floating-point numbers.
- The data seems to be structured and could be used for further analysis or processing in a program.

Was this helpful? ✓ Yes ✕ No

Now, we have to create a database Tool using the SQLDatabase function. Provide it with an engine object and table name.

```
from llama_index.core import SQLDatabase
sql_database = SQLDatabase(engine, include_tables=["bank"])
```

Hide code explanation POWERED BY datalab

- The code begins by importing the `SQLDatabase` class from the `llama_index.core` module.
- Then, an instance of the `SQLDatabase` class is created, named `sql_database`.
- The `SQLDatabase` instance is initialized with two arguments: `engine` and `include_tables`.
- `engine` is a previously defined variable that represents the database engine to be used.
- `include_tables` is a list of tables to be included in the database, in this case, only the "bank" table.

Was this helpful? ✓ Yes ✕ No

Building the SQL query engine

Create the SQL query engine using the NLSQLTableQueryEngine function by providing it with the LlamaIndex SQL database object.

```
from llama_index.core.query_engine import NLSQLTableQueryEngine

query_engine = NLSQLTableQueryEngine(sql_database)
```

Hide code explanation POWERED BY datalab

- The code begins by importing the `NLSQLTableQueryEngine` class from the `llama_index.core.query_engine` module.
- Then, an instance of the `NLSQLTableQueryEngine` class is created, named `query_engine`.
- This instance is initialized with the `sql_database` parameter, which presumably represents a SQL database.

Was this helpful? ✓ Yes ✕ No

Ask the question from the query engine about the “bank” table in the natural language.

```
response = query_engine.query("Which is the longest running campaign?")


print(response.response)
```

Explain code POWERED BY datalab

In response, we will get the answer to your query in natural languages. This is awesome, don't you think?

```
The longest running campaign in the database has a duration of 4918 days.
```

Hide code explanation POWERED BY datalab

The text above is a code output or a data entry that complements the tutorial.

Was this helpful? ✓ Yes ✕ No

Let's ask a complex question.

```
response = query_engine.query("Which type of job has the most housing loan?"
print(response.response)
```

Explain code POWERED BY datalab

The answer is precise, with additional information.

```
The job type with the most housing loans is 'admin.' with 5559 housing loans Thi
```

Hide code explanation POWERED BY datalab

The text above is a code output or a data entry that complements the tutorial.

Was this helpful? ✓ Yes ✕ No

To check what is going on on the back end, we will print the metadata.

```
print(response.metadata)
```

Hide code explanation POWERED BY datalab

- The code is written in Python, a high-level, interpreted programming language.
- The `print()` function is a built-in Python function used to output data to the console.
- `response` is an object that presumably holds some data, including a `metadata` attribute.
- `response.metadata` is accessing the `metadata` attribute of the `response` object.
- The code will print the value of `response.metadata` to the console.

Was this helpful? ✓ Yes ✕ No

As we can see, GPT-4o first generates the SQL query, runs the query to get the result, and uses the result to generate the response. This multi-step process is achieved through two lines of code.

```
{'d4ddf03c-337e-4ee6-957a-5fd2cfaa4b1c': {}, 'sql_query': "SELECT job, COUNT us
```

Hide code explanation

POWERED BY datalab

- The code is a SQL query embedded in a Python dictionary.
- The key 'sql_query' contains the actual SQL command.
- "SELECT job, COUNT(housing) AS housing_loan_count" selects the 'job' column and counts the 'housing' column.
- "FROM bank" specifies the table 'bank' from which the data is selected.
- "WHERE housing = 'yes'" filters the data to only include rows where 'housing' is 'yes'.
- "GROUP BY job" groups the data by the 'job' column.
- "ORDER BY housing_loan_count DESC" sorts the data in descending order by the count of 'housing'.
- The 'result' key in the dictionary stores the result of the SQL query.
- The 'col_keys' key in the dictionary stores the column names of the result.

Was this helpful? ✓ Yes ✕ No

Close the engine when you are done with the project.

```
engine.close()
```

Hide code explanation

POWERED BY datalab

- This is a single line of Python code that calls the `close()` method on an object named `engine`.
- The `close()` method is commonly used to close a connection or a file.
- In this context, it's likely that `engine` is a database engine or a connection to a database.
- By calling `engine.close()`, the code is closing the connection to the database to free up resources.

Was this helpful? ✓ Yes ✕ No

If you are facing issues running the above code, please have a look at the DuckDB SQL Query Engine workspace.

Conclusion

DuckDB is fast, easy to use, and integrates seamlessly with numerous data and AI frameworks. As a data scientist, you will find that it takes only a few minutes to get accustomed to its API and start using it like any other Python package. One of the best features of DuckDB is that it has no dependencies, meaning you can use it virtually anywhere without worrying about hosting or additional setup.

In this tutorial, we have learned about DuckDB and its key features. We have also explored the DuckDB Python API, using it to create a table and perform simple data analysis. The second half of the tutorial covered two projects: one involving a Retrieval-Augmented Generation (RAG) application with DuckDB as a vector database and the other demonstrating DuckDB as an SQL query engine.

Before jumping into using a SQL query engine or integrating a database with AI, you need a basic understanding of SQL and data analysis. You can write the query, but how would you know what question to ask? This is where a basic knowledge of data analysis and SQL comes in. You can gain this knowledge by completing the Associate Data Analyst in SQL career track.

AUTHOR

Abid Ali Awan

As a certified data scientist, I am passionate about leveraging cutting-edge technology to create innovative machine learning applications. With a strong background in speech recognition, data analysis and reporting, MLOps, conversational AI, and NLP, I have honed my skills in developing intelligent systems that can make a real impact. In addition to my technical expertise, I am also a skilled communicator with a talent for distilling complex concepts into clear and concise language. As a result, I have become a sought-after blogger on data science, sharing my insights and experiences with a growing community of fellow data professionals. Currently, I am focusing on content creation and editing, working with large language models to develop powerful and engaging content that can help businesses and individuals alike make the most of their data.

TOPICS

Artificial Intelligence

Top DataCamp Courses

COURSE

Vector Databases for Embeddings with Pinecone

3 hr 464

Discover how the Pinecone vector database is revolutionizing AI application development!

See Details → Start Course

See More →

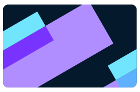
Related

BLOG

An Introduction to DuckDB: What is It and Why Should You...

BLOG

DuckDB makes SQL a first-class citizen on DataLab



TUTORIAL

A Comprehensive Guide to Databricks Lakehouse AI For...

See More →

LEARN

Learn Python

Learn R

Learn AI

Learn SQL

Learn Power BI

Learn Tableau

Learn Data Engineering

Assessments

Career Tracks

Skill Tracks

Courses

Data Science Roadmap

DATA COURSES

Python Courses

R Courses

SQL Courses

Power BI Courses

Tableau Courses

Alteryx Courses

Azure Courses

Google Sheets Courses

AI Courses

Data Analysis Courses

Data Visualization Courses

Machine Learning Courses

Data Engineering Courses

Probability & Statistics Courses

DATALAB

Get Started

Pricing

Security

Documentation

CERTIFICATION

Certifications

Data Scientist

Data Analyst

Data Engineer

SQL Associate

Power BI Data Analyst

Tableau Certified Data Analyst

Azure Fundamentals

AI Fundamentals

RESOURCES

Resource Center

Upcoming Events

Blog

Code-Alongs

Tutorials

Docs

Open Source

RDocumentation

Course Editor

Book a Demo with DataCamp for Business

Data Portfolio

Portfolio Leaderboard

PLANS

Pricing

For Business

For Universities

Discounts, Promos & Sales

DataCamp Donates

FOR BUSINESS

Business Pricing

Teams Plan

Data & AI Unlimited Plan

Customer Stories

Partner Program

ABOUT

About Us

Learner Stories

Careers

Become an Instructor

Press

Leadership

Contact Us

DataCamp Español

DataCamp Português

DataCamp Deutsch

DataCamp Français

SUPPORT

Help Center

Become an Affiliate

Privacy Policy Cookie Notice Do Not Sell My Personal Information Accessibility Security Terms of Use

TUTORIALS category

EN