30 Days of AI

The #30DaysOfAI is a coding challenge designed to help you get started in building AI apps with Streamlit.

Day 2: Hello, Cortex!

Contents

1. Introduction:

For today's challenge, our goal is to run a large language model (LLM) directly within Snowflake. We need to create a simple Streamlit interface that accepts a user's prompt, sends it to a Snowflake **Cortex AI_COMPLETE** function, and gets a response. Once that's done, we will display the AI's generated response back to the user in the app.

See the code:

```
import streamlit as st
from snowflake.snowpark.functions import ai_complete
import json

st.title(":material/smart_toy: Hello, Cortex!")

# Connect to Snowflake
try:
    # Works in Streamlit in Snowflake
    from snowflake.snowpark.context import get_active_session
    session = get_active_session()
except:
    # Works locally and on Streamlit Community Cloud
    from snowflake.snowpark import Session
    session = Session.builder.configs(st.secrets["connections"]["snowflake"]).

# Model and prompt
model = "claude-3-5-sonnet"
prompt = st.text_input("Enter your prompt:")

# Run LLM inference
if st.button("Generate Response"):
    df = session.range(1).select(
        ai_complete(model=model, prompt=prompt).alias("response")
    )

    # Get and display response
    response_raw = df.collect()[0][0]
    response = json.loads(response_raw)
    st.write(response)

# Footer
st.divider()
st.caption("Day 2: Hello, Cortex! | 30 Days of AI")
```

2. How It Works: Step-by-Step

Let's break down what each part of the code does.

Install prerequisite libraries

install the following prerequisite libraries:

snowflake-ml-python==1.20.0

snowflake-snowpark-python==1.44.0

Locally

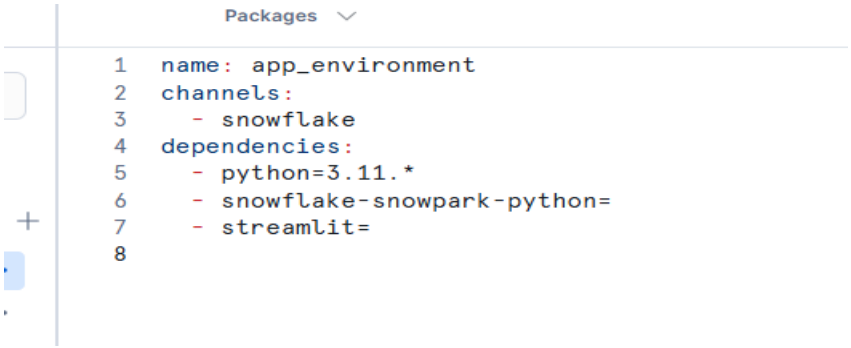
Save the above in **requirements.txt** and run **pip install -r requirements.txt**

Or you could also run

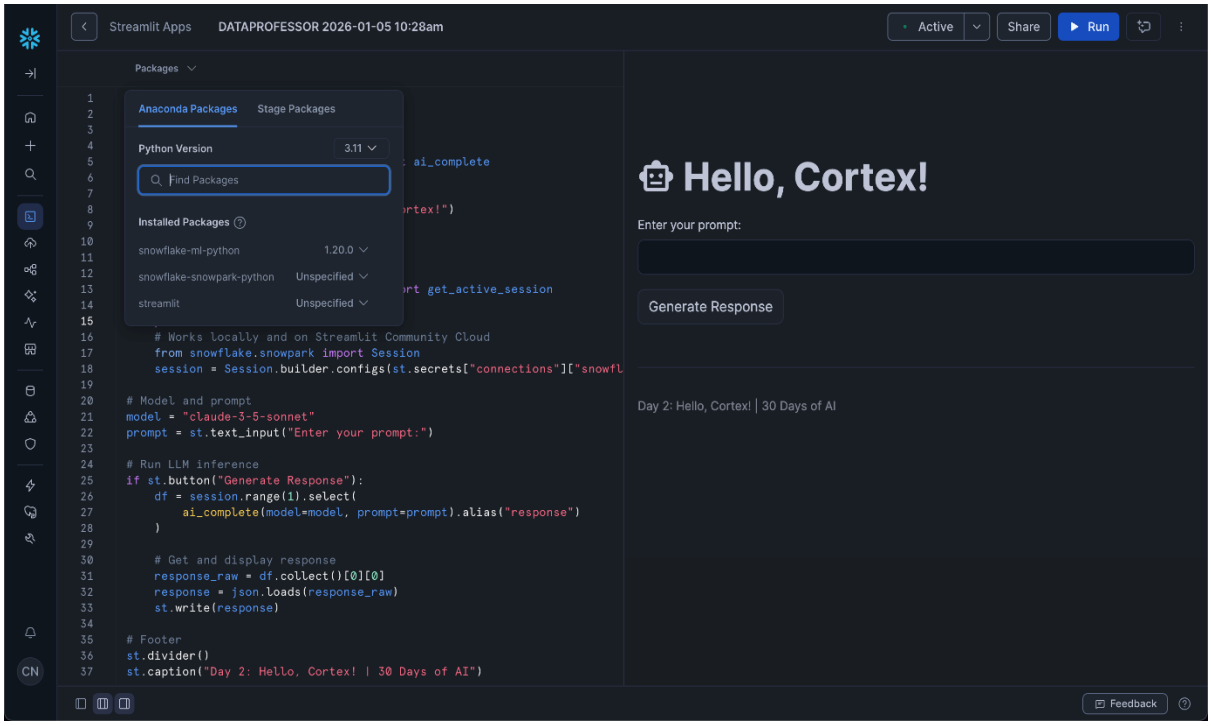
pip install snowflake-ml-python==1.20.0 snowflake-snowpark-python==1.44.0

Streamlit Community Cloud

Save the above in requirements.txt and include this in the GitHub repo of your app.



Streamlit in Snowflake Click on the **Packages** drop-down and enter the libraries name as shown:



2.1. Import Libraries & Connect

```
import streamlit as st
from snowflake.snowpark.functions import ai_complete
import json

# Connect to Snowflake
try:
    # Works in Streamlit in Snowflake
    from snowflake.snowpark.context import get_active_session
session = get_active_session()
except:
    # Works locally and on Streamlit Community Cloud
    from snowflake.snowpark import Session
    session = Session.builder.configs(st.secrets["connections"]["snowflake"]).
```

- **import streamlit as st**: Imports the Streamlit library, which is used to build the web app's user interface (UI).
- **from snowflake.snowpark.functions ...**: Imports the specific Cortex AI function ai_complete that will run the LLM inference.
- **try/except block**: Automatically detects the environment and uses the appropriate connection method:
    - **In Streamlit in Snowflake (SiS)**: Uses get_active_session() for automatic authentication
    - **Locally or on Streamlit Community Cloud**: Uses Session.builder with credentials from .streamlit/secrets.toml
- **session**: The established Snowflake connection, ready to execute queries and call Cortex AI functions

Why ai_complete()? We use the Snowpark ai_complete() function here because it integrates naturally with Snowpark DataFrames. This approach is ideal when you want to process data in a DataFrame pipeline or when you need the response as part of a SQL-like workflow. The trade-off is that it returns JSON that needs parsing, and it doesn't support streaming. In Day 3, we'll see the Python Complete() API which is simpler for direct calls and supports streaming.

2.2. Set Up Model and UI

```
# Model and prompt
model = "claude-3-5-sonnet"
prompt = st.text_input("Enter your prompt:")
```

- **model = "claude-3-5-sonnet"**: Sets a variable to specify which LLM we want to use from the models available in Snowflake Cortex.

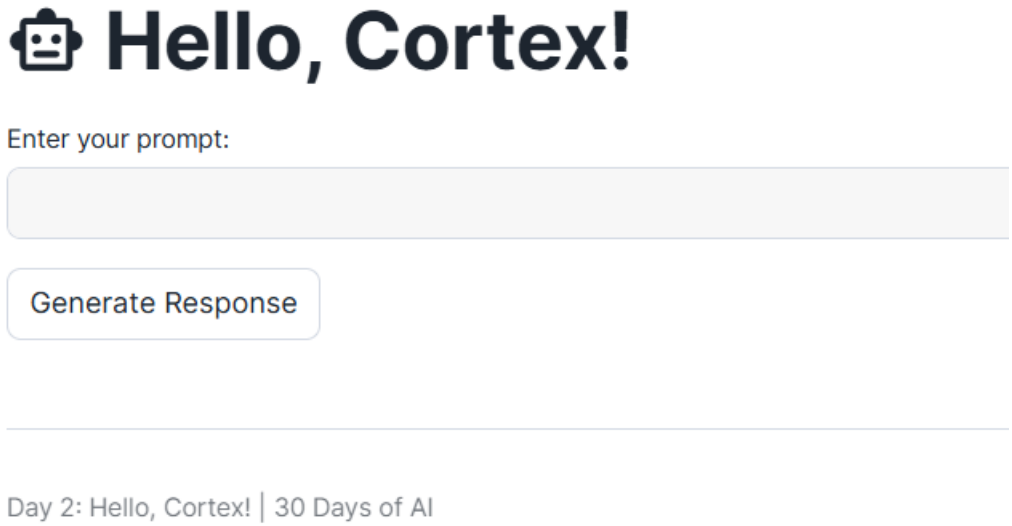
- **prompt = st.text_input(...)**: Creates a text input box in the Streamlit UI with the label "Enter your prompt:". Whatever the user types is stored in the prompt variable.

2.3. Run Inference on Button Click

```
# Run LLM inference
if st.button("Generate Response"):
    df = session.range(1).select(
        ai_complete(model=model, prompt=prompt).alias("response")
    )
```

- **if st.button(...)**: Creates a button in the UI. The code inside this if block only runs when the user clicks the "Generate Response" button.
- **df = session.range(1).select(...)**: This pattern might look strange! Think of it like creating a single-cell spreadsheet just to run a function and capture its output. session.range(1) creates that one-row "spreadsheet," and .select() runs our AI function and puts the result in a column.
- **ai_complete(...)**: This is the core call. It tells Snowflake to run the specified model using the user's prompt.
- **.alias("response")**: Renames the output column to "response" for easier access. The result is stored in a Snowpark DataFrame called df.

2.4. Fetch and Display the Result



- **response_raw = df.collect()[0][0]**: The .collect() command executes the query in Snowflake and pulls the data from the DataFrame df back into the app. [0][0] isolates the actual value from the first row and first column.
- **response = json.loads(response_raw)**: The raw response from ai_complete() looks like this: '{"choices":[{"messages":"Hello! How can I help you?"}]}'. That's a JSON

string, not plain text! This line parses it into a Python dictionary so we can easily access the actual message content.

- **st.write(response)**: Displays the final, parsed response in the Streamlit app for the user to see.

---

3. Resources

- Snowflake Cortex LLM Functions
- COMPLETE Function Reference
- Available LLM Models