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

Day 3: Write streams

Contents

1. Introduction:

For today's challenge, our goal is to run a Snowflake Cortex LLM using the snowflake, cortex, Complete Python API. We need to build a Streamlit app that lets a user select a model, enter a prompt, and then stream the response back. Once that's done, we will display the AI's response in real-time, word by word, as it's being generated.

2. How It Works: Step-by-Step

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

2.1. Imports and Session

```
import streamlit as st
from snowflake.cortex import Complete
import time

# 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 library needed to build the web app's user interface (UI).
- **from snowflake.cortex import Complete**: This is the key import. Instead of using the SQL function ai_complete, we are importing the direct Python Complete class from the Cortex SDK, which is designed for this kind of programmatic use.
- **import time**: Added for the custom generator method, which uses a small delay to smooth out streaming.
- **try/except block**: Automatically detects the environment and connects appropriately (SiS vs local/Community Cloud)
- **session**: The established Snowflake connection

2.2. Configure the User Interface

```
llm_models = ["claude-3-5-sonnet", "mistral-large", "llama3.1-8b"]
model = st.selectbox("Select a model", llm_models)

example_prompt = "What is Python?"
prompt = st.text_area("Enter prompt", example_prompt)

# Choose streaming method
streaming_method = st.radio(
    "Streaming Method:",
    ["Direct (stream=True)", "Custom Generator"],
    help="Choose how to stream the response"
)
```

- `llm_models = [...]`: Defines a Python list of the model names the user can choose from.
- `model = st.selectbox(...)`: Creates a drop-down menu in the UI with the label "Select a model". The user's choice is stored in the `model` variable.
- `prompt = st.text_area(...)`: Creates a multi-line text box for the user's prompt, and pre-populates it with the `example_prompt`. The user's final input is stored in the `prompt` variable.
- `streaming_method = st.radio(...)`: NEW - Adds a radio button to let users choose between the two streaming methods, so they can see the difference in behavior.

2.3. Stream the LLM Response

This app demonstrates **two methods** for streaming responses:

Method 1: Direct Streaming (stream=True)

```
if st.button("Generate Response"):
    with st.spinner(f"Generating response with `{model}`"):
        stream_generator = Complete(
                    session=session,
                    model=model,
                    prompt=prompt,
                    stream=True,   # Built-in streaming
                )

        st.write_stream(stream_generator)
```

- `stream=True`: The simplest approach. Tells `Complete` to return a generator that yields tokens as they arrive.
- **Works when**: The API's streaming is directly compatible with `st.write_stream()`.

Method 2: Custom Generator (Compatibility Mode)

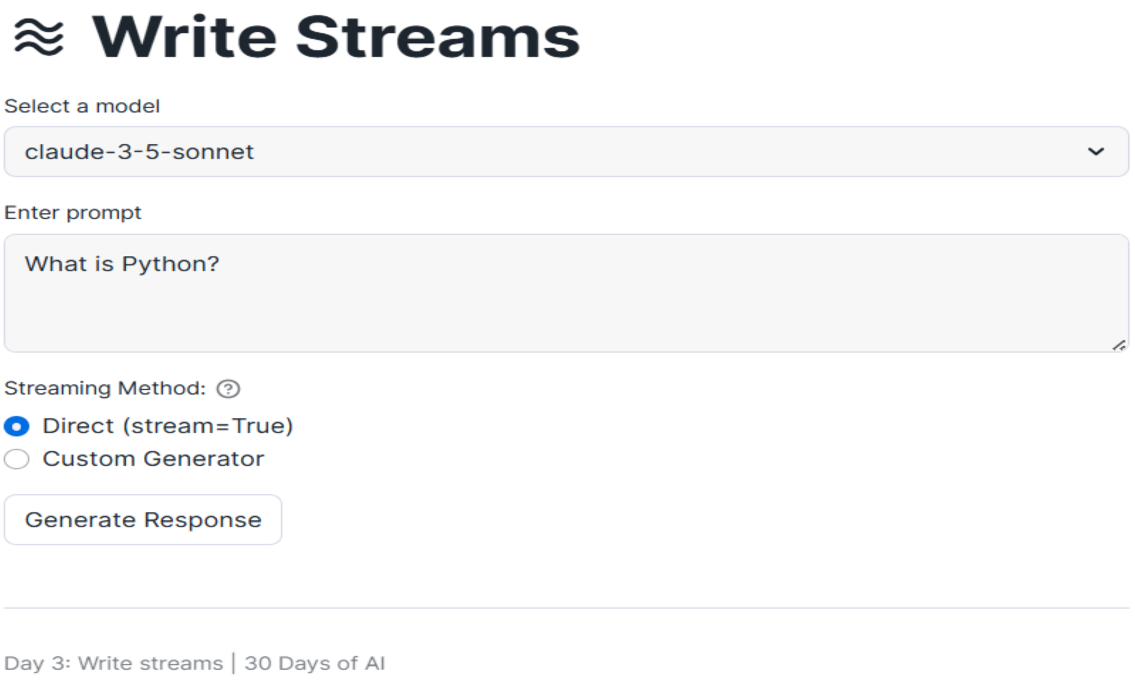
```
def custom_stream_generator():
    """
    Alternative streaming method for cases where
    the generator is not compatible with st.write_stream
    """
    output = Complete(
        session=session,
        model=model,
        prompt=prompt   # No stream parameter
    )
    for chunk in output:
        yield chunk
        time.sleep(0.01)   # Small delay for smooth streaming

with st.spinner(f"Generating response with `{model}`"):
    st.write_stream(custom_stream_generator)
```

- **When to use**: If stream=True doesn't work with st.write_stream() (e.g., compatibility issues with conversation history or certain API responses).
- **How it works**: Creates a Python generator function that manually yields chunks with a small delay for smooth visual streaming.
- **Docstring**: Documents why this alternative method exists—for compatibility when the direct method doesn't work.
- **Best practice**: This is the more reliable method for chatbots and complex prompts (as we'll see in later days)

3. Output:

The AI response appears word by word in real time, creating a smooth chat-like experience.



≋ Write Streams

Select a model

claude-3-5-sonnet

Enter prompt

What is Python?

Streaming Method:

[✓ Direct (stream=True)]
Custom Generator

Generate Response

Day 3: Write streams | 30 Days of AI

4. Resources:

- Cortex Complete Python API
- st.write_stream Documentation