

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

Day 3: Caching your App

Contents

1.	Introduction:	2
2.	How It Works: Step-by-Step	2
2.1.	Setup: Imports and Session	2
2.2.	Defining the Cortex LLM Call	2
2.3.	Building the Web App Interface	3
3.	Output/ result:	3
4.	Resources	4

1. Introduction:

For today's challenge, our goal is to create a Streamlit web application that calls a Snowflake Cortex Large Language Model (LLM). We need to build an interface where a user can enter a prompt, send it to a powerful AI model (like Claude 3.5 Sonnet) running securely inside Snowflake, and get a response. Once that's done, we will display the AI's answer directly in the web app, along with how long the request took.

2. How It Works: Step-by-Step

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

2.1. Setup: Imports and Session

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

# 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 ...:** These lines import all the necessary libraries: `streamlit` for the web app UI, `time` to measure response speed, `json` to parse the LLM's output, and `snowpark` components to connect to Snowflake and use its AI functions.
- **try/except block:** Automatically detects the environment and connects appropriately, working in all deployment scenarios

2.2. Defining the Cortex LLM Call

```
@st.cache_data
def call_cortex_llm(prompt_text):
    model = "claude-3-5-sonnet"
    df = session.range(1).select(
        ai_complete(model=model, prompt=prompt_text).alias("response")
    )

    # Get and parse response
    response_raw = df.collect()[0][0]
    response_json = json.loads(response_raw)
    return response_json
```

- **@st.cache_data:** This is a Streamlit "decorator" that cleverly saves the result of this function. The **first time** you submit a prompt, it calls the LLM and might take 3-5 seconds. If you submit the **exact same prompt** again, Streamlit instantly

returns the saved answer (often under 0.1 seconds). Change even one character in the prompt, and it's a cache miss, meaning the LLM runs again.

- `ai_complete(...)`: This is the core Snowpark function. It securely calls the specified `model` (Claude 3.5 Sonnet) running in Snowflake Cortex, passing it the user's `prompt_text`. The function is wrapped in a `select` statement, which is how Snowpark runs functions.
- `df.collect()[0][0]`: This runs the query. `df.collect()` fetches the result from Snowflake, which is a list containing one row (`[0]`) with one column (`[0]`). The result is a raw text string.
- `json.loads(response_raw)`: The raw text from the LLM is in a format called JSON. This line "parses" or "loads" that text into a structured Python dictionary so we can easily access its contents.

2.3. Building the Web App Interface

```
prompt = st.text_input("Enter your prompt", "Why is the sky blue?")  
  
if st.button("Submit"):  
    start_time = time.time()  
    response = call_cortex_llm(prompt)  
    end_time = time.time()  
  
    st.success(f"*Call took {end_time - start_time:.2f} seconds*")  
    st.write(response)
```

- `st.text_input(...)`: This function draws a text input box on the web page with a label ("Enter your prompt") and some default text ("Why is the sky blue?"). The user's text is stored in the `prompt` variable.
- `if st.button("Submit"):`: This draws a button labeled "Submit". The code indented underneath only runs when the user clicks this button.
- `start_time = ... / end_time = ...`: These lines capture the time just before and just after calling the LLM function, so we can calculate the duration.
- `response = call_cortex_llm(prompt)`: This is where we finally call our function from Step 2, passing it the user's text. The returned dictionary is stored in the `response` variable.
- `st.success(...)`: This displays a green box showing the formatted time elapsed.
- `st.write(response)`: This handy Streamlit command intelligently displays the entire `response` dictionary on the page for the user to see.

3. Output/ result:

When this code runs, you will see a simple webpage with a text box and a button. After entering a prompt and clicking "Submit," the LLM's response will appear below.

⟳ Caching your App

Enter your prompt

Why is the sky blue?

Submit

Day 4: Caching your App | 30 Days of AI

Result:

⟳ Caching your App

Enter your prompt

Why is the sky blue?

Submit

Call took 4.13 seconds

The sky appears blue due to a phenomenon called Rayleigh scattering. Here's a simple explanation:

1. Sunlight contains all colors of the visible spectrum
2. As sunlight enters Earth's atmosphere, it collides with gas molecules
3. These molecules scatter the light in all directions
4. Blue light is scattered more than other colors because it travels in shorter, smaller waves
5. This scattered blue light reaches our eyes from all directions making the sky appear blue

4. Resources

- [st.cache data Documentation](#)
- [Caching in Streamlit](#)
- [SiS Caching Limitations](#)